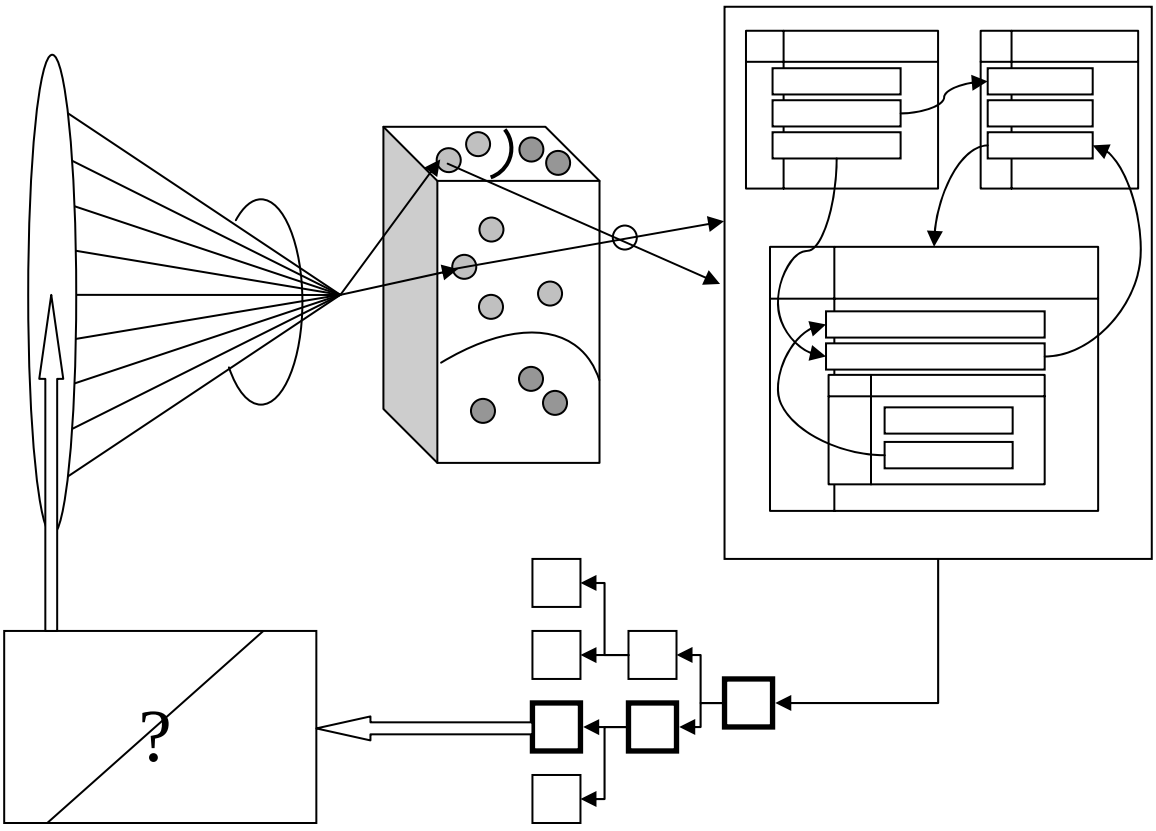
А.С. Потапов

ТЕХНОЛОГИИ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

учебное пособие



Санкт-Петербург

2010

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

А.С. Потапов

ТЕХНОЛОГИИ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Учебное пособие

Санкт-Петербург

2010

Потапов А.С. Технологии искусственного интеллекта – СПб: СПбГУ ИТМО, 2010. – 218 с.

Пособие содержит описание трех базовых проблем искусственного интеллекта – поиска в пространстве решений, представления знаний и машинного обучения. Описываются предпосылки возникновения каждой из проблем, их место в проблематике искусственного интеллекта, а также методы, разработанные для их решения. В частности, описываются методы эвристического программирования и эволюционных вычислений, логические представления знаний, формальные грамматики, семантические сети и фреймы, методы дискриминантного и синтаксического распознавания образов, восстановления наборов правил и деревьев решений.

Предназначено для студентов, обучающихся по направлению подготовки 200600 – «Фотоника и оптоинформатика».

Рекомендовано к печати УМО по образованию в области приборостроения и оптотехники в качестве учебного пособия для студентов высших учебных заведений, обучающихся по направлению подготовки 200600 – «Фотоника и оптоинформатика».

В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена Программа развития государственного образовательного учреждения высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» на 2009–2018 годы.

1. Структура области искусственного интеллекта

Термин «искусственный интеллект» используется для обозначения большого направления научных и прикладных исследований. Такое название, закрепившееся за этим направлением, у большинства людей скорее ассоциируется с разумными роботами или мыслящими компьютерами, многочисленные образы которых были созданы в научно-фантастических произведениях. Действительно ли специалисты по искусственному интеллекту ставят перед собой столь амбициозные задачи? Многие из них это отрицают.

Сами исследователи выделяют две основные цели своей работы – это (1) автоматизация человеческой деятельности, в особенности тех ее видов, которые традиционно считались интеллектуальными, и (2) создание компьютерных моделей, имитирующих процессы решения человеком тех или иных интеллектуальных задач в целях объяснения сущности этих процессов.

Еще одной возможной целью, о которой, однако, часто забывают, является создание усилителя интеллекта (УИ). Методология УИ-направления не сильно, но все же отличается от методологии ИИ-направления. Но что отличается существеннее – это прогнозируемые социальные последствия.

Сразу стоит отметить, что первым двум целям соответствуют и два различных подхода в ИИ, которые обычно называются техническим (или эвристическим) и бионическим. В рамках технического подхода психофизиологическая достоверность моделей мыслительных процессов приносится в жертву эффективности, с которой эти модели решают поставленные перед ними задачи, и интеллектуальность компьютерных программ определяется по тому, насколько хорошие результаты они получают по сравнению с человеком. При бионическом подходе, напротив, необходимым считается сходство самих процессов решения некоторой задачи компьютерной программой и человеком в ущерб качества конечного результата.

Зачастую эти два подхода противопоставляют, в чем нередко доходят до крайностей. Так, ярые приверженцы технического подхода приводят такие уже избитые аналогии как птица и самолет или колесо и ноги. Технически системы выполняют те же функции, что и биологические системы, но делают это совершенно другим образом, причем исследование, скажем, структуры перьев птиц вряд ли помогло создать самолет. Сторонники же бионического подхода утверждают, что бессмысленно пытаться создать искусственный интеллект, не познав естественного, который является единственным примером интеллекта, то есть того объекта, который хотят повторить исследователи ИИ. Да и зачем «с нуля» выдумывать то, что можно позаимствовать у природы? В действительности, эти два подхода просто преследуют две несколько разные цели и друг другу не противоречат.

Существует еще один подход к ИИ, называемый эволюционным, в рамках которого предлагается имитировать не мыслительные процессы уже сформировавшегося интеллекта взрослого человека, а сам процесс становления этого интеллекта в онто- и филогенезе, возможно, начиная с весьма ранних этапов эволюции.

Итак, специалисты по ИИ зачастую ставят перед собой не глобальную цель создания мыслящих машин, а более конкретные задачи поиска автоматического решения некоторых интеллектуально трудных задач либо моделирования отдельных аспектов мышления человека или животных. Тем не менее, свое название научное направление «искусственный интеллект» получило не случайно. Ведь одной из первых работ в этом направлении была основополагающая статья А. Тьюринга «Вычислительные машины и интеллект», опубликованной в 1950 г. в журнале «Mind». И основной темой этой статьи был вопрос: «Может ли машина мыслить?» (собственно, в русском переводе название статьи звучало именно так). Само же название «искусственный интеллект» распространилось в 1960-х годах (хотя появилось оно чуть раньше), и уже в 1969 году была проведена 1-я Международная объединенная конференция по искусственному интеллекту, которая официально и закрепила это название. Таким образом, видно, что ученые, определявшие лицо новой науки, все же в качестве конечной ее цели видели создание именно машинного разума.

Статья Тьюринга наиболее запомнилась предложенным в ней тестом разумности, названным автором «игрой в имитацию», но позднее получившим имя автора. Тест Тьюринга заключается в том, что некоторый человек («следователь») должен, общаясь с собеседником посредством текстовых сообщений и не видя своего собеседника, определить, является ли он компьютером или человеком. То есть компьютер должен быть отличен от человека только на основе того, как он ведет разговор, отвечает на те или иные вопросы.

Однако смысл этого теста спустя некоторое время стал пониматься слишком буквально, о чем говорит, к примеру, существование конкурса Лебнера, на который ежегодно представляются программы для прохождения теста Тьюринга.

Чтобы понять истинное значение этого теста, следует обратиться к социально-историческому контексту той статьи. В то время информация о принципах работы человеческого мозга (как на нейрофизиологическом, так и на психологическом уровне) была очень скудной. Высказывание: «мысль вырабатывается мозгом, как желчь – печенью», – могло показаться саркастическим далеко не всем, и некоторые ученые еще не так давно действительно пытались выделить мысль как некое вещество и установить его химическую формулу. И напротив, даже среди ученых гипотеза о том, что мышление обеспечивается некоторой нематериальной субстанцией, воспринималась все еще достаточно серьезно. Компьютеры только появились, были чрезвычайно громоздкими и медлительными и

использовались преимущественно для вычислений. В связи с этим сама постановка вопроса о том, может ли машина мыслить, вызывала массу споров.

В этой ситуации Тьюринг предложил свой тест не как конкретный практический рецепт проверки интеллектуальности компьютера, а скорее как призыв перейти от умозрительных и бесплодных рассуждений о возможности машинного интеллекта к экспериментальным исследованиям с использованием компьютеров. Ведь не столь важно, называть ли некоторую машину разумной или нет, важно, чтобы она решала поставленные перед ней задачи не хуже человека.

Споры на тему «может ли машина мыслить» не прекращаются и по сей день. Одна из позиций в этом споре была выражена Ньюэллом и Саймоном в гипотезе физической символьной системы, согласно которой для достижения интеллектуального поведения системой необходимо и достаточно, чтобы физическая система выполняла преобразование символьной информации. По сути, данная гипотеза говорит не более того, что искусственный интеллект может и должен быть реализован на физическом воплощении универсальной машины Тьюринга, например, на обычном компьютере.

Эта гипотеза подвергается жесткой критике. Один из наиболее известных аргументов против нее опирается на мысленный эксперимент, названный «Китайской комнатой». В этом эксперименте человек, не знающий китайского языка, помещается в некоторую комнату, полную корзинок с китайскими иероглифами. Человека также снабжают книгой, в которой описаны формальные правила манипуляции с этими символами без объяснения их смысла. Вне комнаты находятся люди (понимающие китайский), передающие наборы иероглифов человеку в комнате, который совершает манипуляции, описанные в книге, после чего выдает наружу ответ, причем этот ответ соответствует тому, который был бы дан человеком, понимающим китайский язык. Действия человека в комнате уподобляются действию компьютера по манипуляции символами. Сирл (автор этого мысленного эксперимента) говорит: человек в комнате, работающий как символьная физическая система, пройдет тест Тьюринга, но он совершенно не понимает смысла вопросов и ответов на них.

Мы не будем здесь обсуждать аргументы двух непримиримых сторон в этом до сих пор продолжающемся споре, заметим лишь, что отсутствие однозначного ответа на вопрос о возможности мыслящих машин не помешало специалистам по ИИ создать множество полезных приложений и оказать неоценимую помощь психологии, лингвистике и другим дисциплинам. В рамках данного курса ответ на этот вопрос также не будет иметь особого значения. Здесь основное внимание будет уделено методам и технологиям, разработанным в области ИИ, которые доказали свою ценность на практике при решении сложных задач. Эти методы также раскрывают вполне определенные аспекты умственной деятельности (чем и вызвана их успешность).

Область искусственного интеллекта является крайне неоднородной. В ней существуют различные направления исследований, которые выделяются либо по задаче (или предметной области), требующей интеллектуального анализа, либо по используемому инструментарию, либо по разрабатываемой модели мышления.

К направлениям, выделяемым на основе решаемой задачи, относятся:

- машинный перевод;
- автоматическое реферирование и информационный поиск;
- системы речевого общения;
- игровой интеллект, доказательство теорем и автоматизация научных исследований;
- компьютерное зрение;
- извлечение данных;
- сочинение текстов и музыки и др.

Большинство этих задач связано с автоматизацией различных видов деятельности человека, в связи с чем соответствующие направления исследований возникли вместе с возникновением компьютеров, и до середины 1950-х годов во многих из них были предложены первые решения. Доля, приходящаяся на исследования в разных направлениях, варьировалась во времени, однако все эти задачи в разной степени актуальны и сейчас.

Перечисленные направления характеризуются тем, что значительная часть проводимых в них исследований посвящена не процессам мышления, а предмету интеллектуального анализа. К примеру, для сочинения текстов изучение структуры литературных произведений имеет чуть ли не большее значение, чем изучение мыслительной деятельности писателя.

Направления в ИИ, выделяемые по развиваемому в них инструментарию, включают:

- искусственные нейронные сети;
- эволюционные вычисления;
- распознавание образов;
- экспертные системы;
- эвристическое программирование;
- мультиагентный подход и т.д.

Сюда также можно отнести и ряд других направлений, включающих исследование более частных групп методов.

Отличие данных направлений в том, что в них развивается аппарат решения большого класса задач. К примеру, методы распознавания образов могут применяться при категоризации текстов в информационном поиске, в системах речевого общения, компьютерного зрения и многих других направлениях первого типа. В то же время, для решения одной задачи могут применяться разные методы. К примеру, нейронные сети способны решать те задачи, которые решаются методами распознавания

образов, а эволюционные вычисления могут заменять методы эвристического программирования и наоборот.

Эта группа направлений более неоднородна, чем первая. В ней существуют направления (например, ИНС), которые претендуют на то, чтобы называться отдельным подходом к искусственному интеллекту в целом. Однако за время своего существования эти подходы не приблизили нас к понимания процесса мышления, поэтому их следует рассматривать именно как совокупность методов, объединенных некоторой общей идеей или математическим аппаратом. В отличие от первой группы направлений исследований эти направления появились в разные моменты времени и момент их зарождения часто связан с выходом какой-то конкретной работы.

К направлениям третьего типа можно отнести:

- поиск в пространстве решений;
- представление знаний;
- машинное обучение.

Каждое из этих направлений сосредотачивает внимание на одном из аспектов интеллекта. Их истоки лежат в области философии. Ведь не зря Г.С. Поспелов сказал: «За спинами специалистов по искусственному интеллекту стоят тени великих философов». В связи с этим, момент возникновения каждого из этих направлений назвать затруднительно, но можно выделить этап, когда каждое из них доминировало. При этом смена этих этапов определяет логику развития области ИИ в целом.

Первые исследователи в области ИИ вначале были вынуждены заимствовать идеи из других дисциплин, занимающихся изучением интеллекта естественного. В психологии в первой половине XX века при исследовании поведения животных была обнаружена чрезвычайно большая роль поиска, который начинает осуществляться в ответ на ситуацию, для которой нет готового решения. При этом, если для низших животных этот поиск происходит во внешнем пространстве, то для высших животных такой поиск, сначала тоже выражающийся в перепроизводстве движения, может внезапно перейти в какой-то сложный внутренний процесс, реализующий поиск в ментальном пространстве, «лабиринте» состояний, достижимых путем выполнения доступных действий. В результате была сформирована лабиринтная гипотеза мышления.

«Лабиринтная» гипотеза была развита первыми специалистами по ИИ, предмет исследования которых составляли интеллектуальные игры и доказательство теорем, где концепция поиска также играла ключевую роль. При этом полагалось, что суть интеллекта состоит в решении проблем, а сам процесс решения может быть представлен как поиск пути от исходных данных к ответу в пространстве возможных решений (или как поиск пути от имеющихся средств к конечной цели через достижимые подцели). Однако со временем обнаружилась ограниченность подобных систем: для них формализованное описание задачи должно было

составляться человеком. Возникла проблема формирования самого «лабиринта», для решения которой необходимо было, чтобы машинная система могла использовать знания о предметной области. Более того, не удалось найти универсального алгоритма, эффективно решающего любую задачу поиска в произвольном «лабиринте».

Проблема представления знаний стала доминирующей с середины 1970-х годов, чему также способствовала бурно развивающаяся в то время отрасль компьютерной лингвистики. Системы, основанные на знаниях, нашли широкое применение в виде экспертных систем, с которыми одно время отождествлялась чуть ли не вся область ИИ. Экспертные системы, используя знания о предметной области, оказались способными строить формальные описания задач, сформулированных на ограниченном естественном языке для одной узкой предметной области. Поиск решения проблемы в системах, основанных на знаниях, превратился в проблему манипулирования знаниями, в чем и усматривалась теперь суть мышления. Важно отметить, что проблема манипулирования знаниями является более узкой и более конкретной, чем проблема поиска вообще.

Однако по мере развития данного направления была выявлена другая проблема, а именно проблема автоматического приобретения знаний. Более широко эта проблема была сформулирована как проблема машинного обучения. Хотя и раньше важность обучения понималась многими учеными, только после исследований в области представления знаний стало ясно, насколько большой объем информации получает человек в процессе обучения и насколько трудоемко закладывать эти знания в машинные системы вручную. Выделившись в самостоятельное направление, машинное обучение в 1980-е годы стало привлекать все большее внимание и в результате стало центральным в области ИИ. Интеллект перестал пониматься как некий готовый продукт, который можно воспроизвести, или как фиксированная способность к решению задач или манипулированию знаниями.

Смена парадигмы привела к постановке новых проблем в ранее исследованных направлениях. В частности, в задачах поиска была сформулирована проблема автоматического построения эвристик поиска (оптимизации поиска), которая, однако, все еще остается мало исследованной. Методы машинного обучения в задачах приобретения знаний применяются к неопределенным и противоречивым наблюдательным данным, поэтому и порождаемые в результате знания не обладают полной достоверностью. Как следствие, возникает проблема представления нечетких знаний, а проблема манипулирования знаниями превращается в проблему рассуждений в условиях неопределенности.

Еще одна группа проблем связана с применением методов обучения к самой проблеме обучения, т.е. с метаобучением. В частности, если в задачах приобретения знаний подразумевается, что представления знаний являются заданными априори и нужно лишь построить систему знаний в рамках этих представлений, то в задачах метаобучения ставится вопрос об

автоматическом построении самих представлений, детали которых могут сильно меняться в зависимости от предметной области. Подобные проблемы являются очень сложными и мало изученными. Однако их решение необходимо для снятия следующего ограничения для машинных систем – их способности функционировать только в узкой предметной области.

Итак, исследования в области ИИ начались с парадигмы «мышление как поиск» и с разработки методов решения формально поставленных задач. Дальнейшая смена парадигм была связана с увеличением универсальности машинных систем, благодаря уменьшению объема информации, подготавливаемой для них человеком. Если на первом этапе развития ИИ описание каждой задачи формировалось человеком, то на втором этапе человек уже задавал описание некоторой (достаточно узкой) предметной области, включающей целый комплекс задач. На третьем этапе машинная система получает возможность, по крайней мере частично, строить описание предметной области самостоятельно в рамках заданного человеком представления.

На рис. 1 представлена схема, условно изображающая структуру базового уровня области искусственного интеллекта (той ее части, которая к настоящему времени является сравнительно устоявшейся).

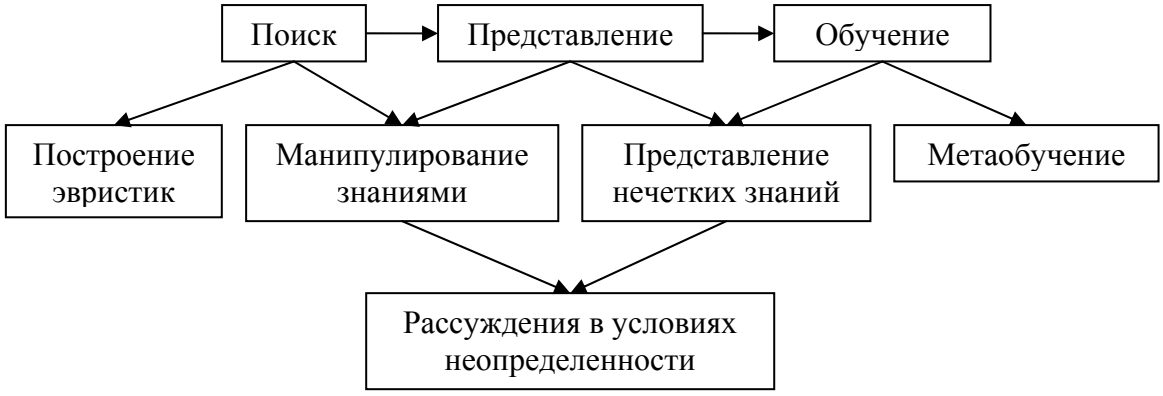


Рис. 1. Структура базового уровня области искусственного интеллекта

Последующее развитие области ИИ, по-видимому, связано с дальнейшей универсализацией машинных систем и получением ими более широкого доступа к информации. Последнее может быть связано с направлением, в рамках которого исследуются воплощенные системы, т.е. системы, помещенные в конкретное информационное, физическое, социальное окружение. Так, согласно позиции Р. Брукса, интеллект не может возникнуть в невоплощенных системах, таких как традиционные системы доказательства теорем или экспертные системы (помимо самого этого тезиса ученый также предлагает и конкретную многослойную архитектуру, состоящую из взаимодействующих более простых образований и названную им категориальной архитектурой, для управления роботами). Это направление не является принципиально

новым, так как многое заимствует из робототехники. Однако в воплощенных системах подразумевается, что поступающая сенсорная информация должна служить основой обучения, в результате которого должна формироваться система знаний с целью их использования для последующего решения поставленных задач. Проблема обучения «с нуля» на основе сенсорной информации ставит множество дополнительных проблем, решение которых еще предстоит искать в будущем. Вероятно, современное состояние в области ИИ можно охарактеризовать как этап синтеза, на котором происходит объединение методов, полученных ранее в рамках изолированных направлений исследований.

Выше выделены три базовых проблемы (и несколько производных от них), которые обуславливали развитие области ИИ до настоящего момента (хотя, возможно, их число больше). Таким образом, структуру области ИИ можно условно разделить на три уровня (см. рис. 2).

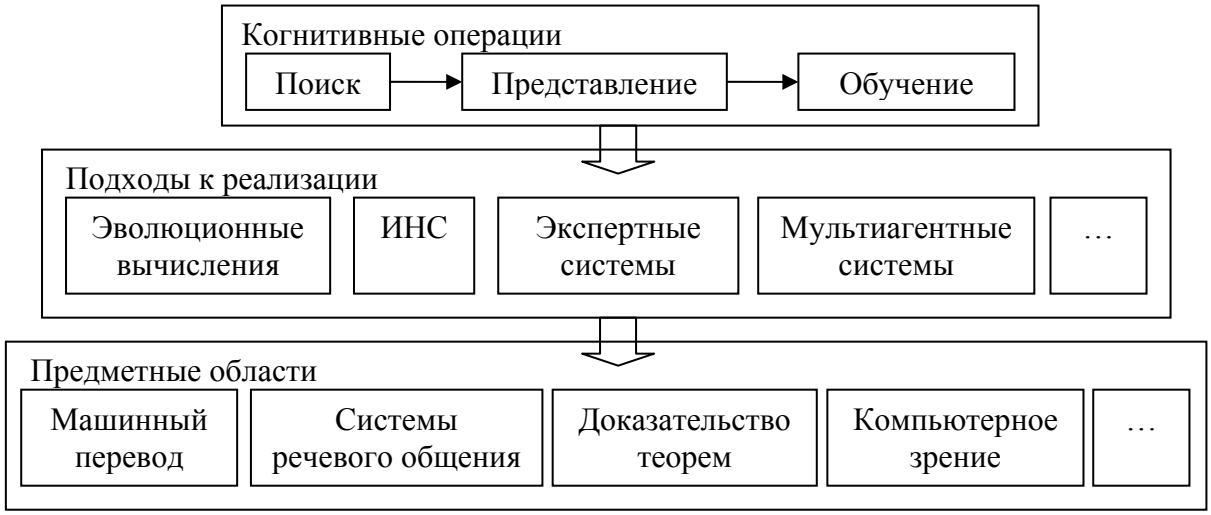


Рис. 2. Общая структура области искусственного интеллекта

Вопросы и упражнения

1. Что такое искусственный интеллект?
2. Какие цели ставят исследователи в области ИИ? Какие подходы выделяются по этим целям?
3. В чем заключается тест Тьюринга?
4. Что говорит гипотеза символьной физической системы?
5. В чем заключается парадокс китайской комнаты? Какие возражения можно предложить против этого парадокса?
6. Из каких разделов состоит область ИИ? Как в общем виде можно представить структуру этой области?
7. Какие наиболее существенные ограничения есть у существующих интеллектуальных систем? Какое возможно дальнейшее развитие данной области?

2. Основные понятия эвристического программирования

Первым доминирующим направлением в области ИИ было направление, обычно обозначаемое термином «эвристическое программирование», в котором была принята метафора, что мышление – это поиск в пространстве решений. Данное направление тесно связано с лабиринтной гипотезой мышления в психологии и исследованиями формализации процесса доказательства теорем в математике.

Математика занимается исследованием формальных (символьных) систем. На основе некоторой системы аксиом, определяющей интересующий математика класс объектов, далее доказываются различные теоремы о свойствах этих объектов. Таким образом, из истинных утверждений дедуктивно выводятся другие истинные утверждения. Однако сам процесс поиска математического доказательства является неформальным, то есть лежит вне самой математики: между формальными аксиомами и выведенными из них утверждениями находится неформальный человеческий разум. Для ученых было естественным желание избавиться от этого неформального компонента, для чего было необходимо попытаться строго описать процесс решения математических проблем.

В результате этих попыток в середине 1930-х годов было разработано формальное понятие алгоритма как последовательности операций над символьными выражениями. Механическое применение алгоритма, описывающего решение некоторой математической задачи, к цепочке символом, задающей условие этой задачи, в результате дает новую цепочку символов – ответ. Поиск решения – это поиск подходящего алгоритма.

Интересно отметить, что одной из первых формализаций понятия алгоритма является машина Тьюринга, представляющая собой некоторое воображаемое механическое приспособление, выполняющее преобразование цепочки символов, записанных в ячейках входной ленты, в соответствии с заложенным в нее набором правил. С одной стороны, современные компьютеры, по сути, воплощают собой концепцию машины Тьюринга. С другой стороны, машина Тьюринга воплощала наиболее общие аспекты мышления…

Итак, мышление сводилось к решению задач, что представлялось как поиск в пространстве цепочек допустимых операций. То, что эвристическое программирование считалось не просто одним из вопросов ИИ, а именно подходом к созданию ИИ в целом, говорит о том, насколько важной в то время виделась концепция поиска.

На то, что сам процесс мышления суть поиск, указывали и психологические исследования того времени. Особенно отчетливо значение поиска было видно в поведении животных. Так, если на пути муравья, идущего по привычному маршруту, положить препятствие, муравей начинает беспорядочно бегать во все стороны. Сходным образом

ведет себя курица, обнаружив, что отверстие в ограде, через которое она обычно подходила к пище, оказывается закрытым. Курица в сильном возбуждении начинает метаться вдоль ограды, беспорядочно тычась во все отверстия. И даже голодная собака, видя за решеткой мясо, делает множество попыток просунуть морду и лапы в отверстия решетки. Подобное поведение возникает у животного, когда оно сталкивается с ситуацией, для которой нет готового решения и заученные схемы поведения не работают. Это поведение кажется бессмысленным только на первый взгляд. Случайный поиск позволяет найти новое решение, обходной путь. И даже человек, например, собирая головоломку, осуществляет такой поиск, что имело большое значение для первобытного человека при изготовлении и использовании орудий труда, у которого доминировало так называемое ручное мышление.

Поведение обезьяны при появлении препятствий отличается от поведения других животных. На проблемную ситуацию обезьяна сначала тоже реагирует перепроизводством движения, гиперкинезом. Однако если это не приводит к успеху, достаточно быстро поведение обезьяны резко меняется: она неподвижно замирает, фиксируя глазами цель. Вместо перепроизводства движения нервное возбуждение расходуется на какой-то сложный внутренний процесс. Животное осуществляет поиск решения не во внешнем, физическом пространстве, а в некотором внутреннем, ментальном его отражении.

Эти наблюдения показали, что мыслительная деятельность возникает в ответ на проблемную ситуацию, для которой нет готового решения, и заключается в поиске ее решения. У низших животных этот поиск осуществляется непосредственно в физическом пространстве. У человека, напротив, поиск приобретает еще более сложную форму, чем у обезьян, благодаря развитию языка как символьной системы. Таким образом, поиск с использованием символьных представлений выглядел хорошим кандидатом на объяснение сути мышления, что было выражено в лабиринтной гипотезе мышления, название которой очевидным образом связано с лабиринтами, используемыми зоопсихологами для оценки интеллектуальных способностей животных. Однако здесь лабиринт не обязательно является физическим; это лабиринт возможных путей решения, в котором производится поиск пути, ведущего из исходной точки – условий задачи, к конечной точке – ее решению.

Лабиринтная гипотеза мышления была с энтузиазмом поддержана исследователями ИИ, поскольку эта гипотеза допускала вполне прозрачную формализацию и позволяла исследовать проблему решения на задачах, имеющих простое описание. В качестве таких задач выступали преимущественно различные интеллектуальные игры и доказательство теорем, а также некоторые прикладные задачи. Достаточность лабиринтной гипотезы неявно признавалась не только сторонниками ИИ. К примеру, со стороны критиков ИИ часто слышались утверждения, что компьютер никогда не сможет играть в шахматы лучше человека. Это

говорит о том, что критика ИИ основывалась не на более глубоком понимании природы человеческого разума, а на субъективных предпочтениях.

Итак, была сформирована первая парадигма ИИ: мышление как решение задач путем поиска в пространстве вариантов. Уже первые работы в этом направлении позволили выявить некоторые трудности. Как оказалось, даже для сравнительно простых задач пространство решений чрезвычайно велико и исследование всего пространства невозможно. Для получения реально работающих программ необходимо использование неких правил или приемов, которые позволяют повысить эффективность программ, осуществляющих решение сложных задач. Эти приемы получили название эвристик, а создание программ, реализующих поиск на основе эвристик, – эвристическим программированием.

Слово «эвристический» означает «способствующий открытию». Таким образом, проводится параллель между эвристическим программированием и научным поиском, что еще раз подчеркивает нетривиальность проблемы поиска и ее значимость для понимания природы интеллекта.

Работы в направлении эвристического программирования привели к формированию собственной терминологии и методов исследования. Центральным понятием в эвристическом программировании является понятие дерева вариантов (пространства состояний), которое обычно принимает форму *дерева игры* либо *дерева целей*. Корень дерева представляет собой исходное состояние, из которого выходят ветви, соответствующие тому, как это состояние может быть изменено. Листьям дерева, не имеющим потомков, соответствуют состояния, для которых выполняется *критерий окончания*. В дереве целей корню, как правило, соответствует цель, а дочерним узлам – подцели, достижение которых необходимо или достаточно для достижения цели.

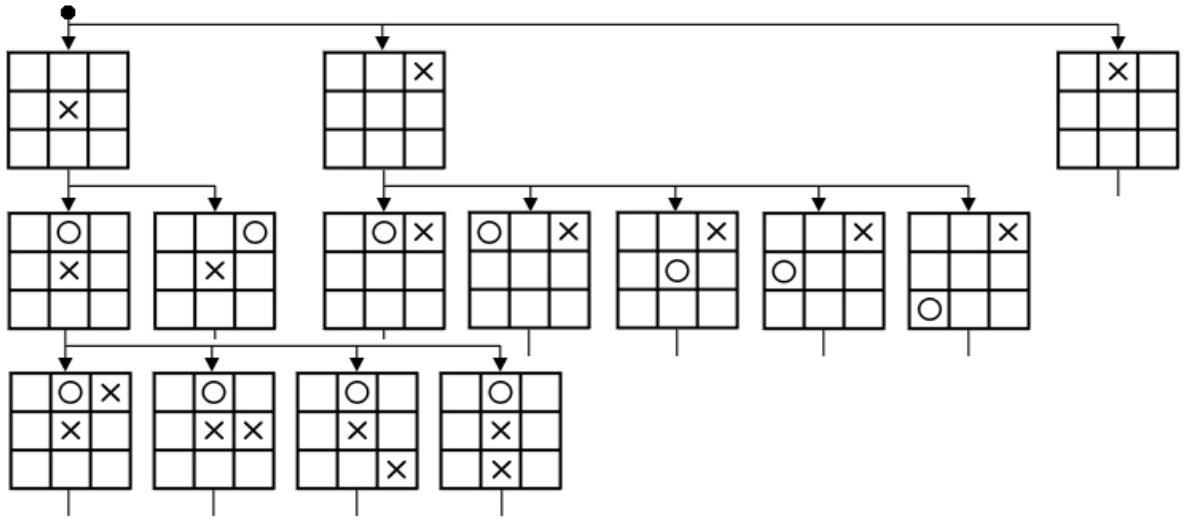


Рис. 3. Пример фрагмента дерева игры

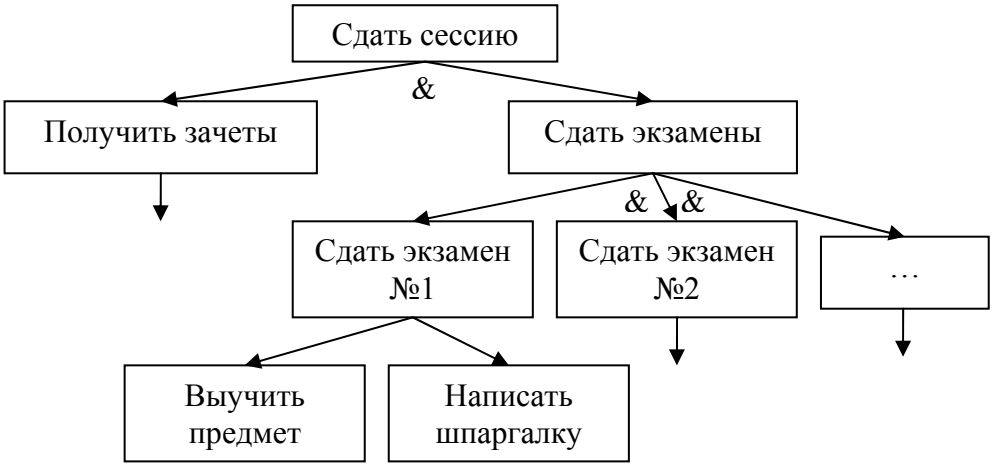


Рис. 4. Пример фрагмента дерева целей

На рис. 3 и 4 приведены примеры фрагментов дерева игры и дерева целей соответственно.

Решение некоторой задачи сводится к нахождению удовлетворяющего условиям задачи листа и построению пути от корня дерева до этого листа. Все возможные пути от корня до листьев составляют *пространство решений*.

Альтернативным представлением является граф состояний. Если в дереве состояний одному и тому же состоянию могут соответствовать разные узлы в зависимости от способа достижения этого состояния, то в графе состояний каждому состоянию соответствует только один узел, однако в него может входить несколько дуг, причем в таком графе могут присутствовать циклы. Представление в виде графа является более компактным, но его анализ значительно сложнее.

Деревья игры и целей могут быть *эксплицитными* и *имплицитными*. Эксплицитное дерево задается в явном виде путем указания всех его узлов и дуг, в то время как имплицитное дерево задается путем указания исходной позиции и правил формирования дерева (правил игры или допустимых операций по изменению текущего состояния).

Решение задачи поиска на эксплицитном дереве тривиально: достаточно лишь из листьев дерева выбрать подходящее. Однако эксплицитные деревья в реальных задачах встречаются редко.

Процедура, которая превращает имплицитное дерево в эксплицитное, называется *порождающей процедурой*. В классическом эвристическом программировании полагается, что такая порождающая процедура существует (и детерминирована), и она известна. Иными словами, конкретное действие в конкретной ситуации всегда приводит к одному и тому же известному изменению этой ситуации.

Если такая порождающая процедура существует, то в чем же тогда проблема? Несмотря на свое существование, эта процедура далеко не всегда позволяет сформировать эксплицитное дерево из-за того, что оно

может получиться чрезвычайно больших размеров. К примеру, размер дерева для шашек 10^{40}, для шахмат 10^{120}, а для го – 10^{400}.

Поскольку дерево в имплицитной форме является как бы невидимым для программы и программа может построить («увидеть») только часть эксплицитного дерева при фиксированных затратах вычислительных ресурсов, этот процесс также называют поиском. Здесь несложно проследить аналогию между поиском решения на имплицитном дереве и поиском пути в реальном лабиринте, где обзор также ограничен.

Поскольку все дерево порождено быть не может, становится принципиальным, в каком порядке происходит порождение узлов (или в каком порядке осуществляется обход узлов имплицитного дерева). В зависимости от этого порядка порождающие процедуры можно разделить на несколько типов. Две наиболее сильно отличающиеся стратегии поиска – это поиск в глубину и поиск в ширину.

При поиске в глубины на каждом шаге посещается один из еще не рассмотренных потомков текущего узла (обычно самый левый). Если не посещенных потомков нет, то производится возврат к родительскому узлу, а текущий узел помечается, как изученный. При поиске в ширину сначала посещаются все узлы, находящиеся на одной глубине с текущим узлом, и лишь затем осуществляется переход к их потомкам, находящимся на следующем уровне глубины. На рис. 5 представлены схемы обхода в глубину и ширину.

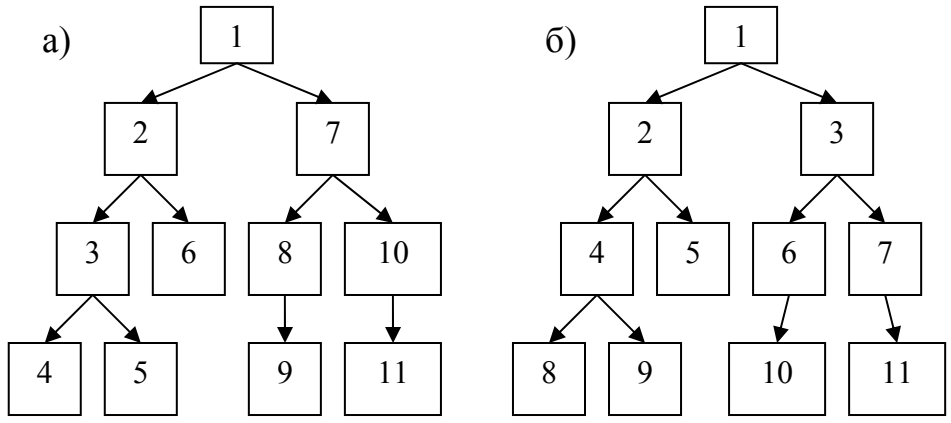


Рис. 5. Порядок обхода узлов дерева при поиске (а) в глубину; (б) в ширину

Поскольку порядок обхода дерева вариантов при поиске в глубину и ширину различается, при наличии ограничения на число посещенных узлов с использованием этих двух стратегий поиска разные узлы останутся не посещенными.

Еще один часто используемый способ перебора – это порождение комбинаторных объектов в лексикографическом порядке, т.е. введение линейного порядка на множестве вариантов и перечисление всех вариантов в этом порядке. Этот способ удобен тем, что если дерево вариантов удается свести к одному из типов комбинаторных объектов, то

это может существенно упростить процедуру поиска. Рассмотрим несколько типичных задач.

1) Перечисление всех бинарных строк длины N.

Пример перечисления в лексикографическом порядке для N=3:

000, 001, 010, 011, 100, 101, 110, 111.

2) Перечисление всех перестановок символов в строке длины N.

Пример перечисления в лексикографическом порядке для N=3.

123, 132, 213, 231, 312, 321.

3) Перечисление всех K-элементных подмножеств множества из N элементов.

Пример для N=5, K=3.

00111, 01011, 01101, 01110, 10011, 10101, 10110, 11001, 11010, 11100.

Сами алгоритмы порождения комбинаторных объектов состоят из процедуры генерации первого объекта и процедуры перехода от n-го к (n+1)-му объекту.

Однако при порождении объектов в лексикографическом порядке проблематичным оказывается введение эвристик, отсекающих неперспективные ветви, поскольку эти объекты, как правило, соответствуют листьям на дереве вариантов, и процедура порождения переходит от листа к листу, минуя неконечные узлы на дереве (см. рис. 6).

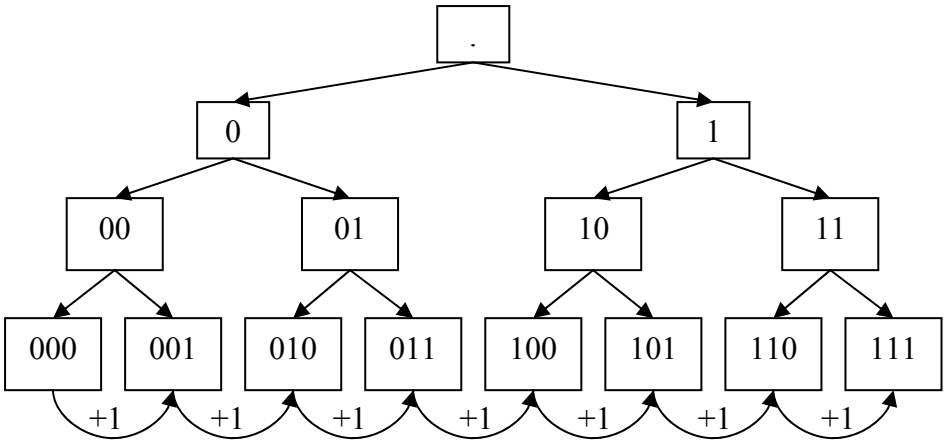


Рис. 6. Отличие процедуры обхода дерева и порождения конечных объектов в лексикографическом порядке

Поскольку зачастую все листья достигнуты быть не могут, оказывается необходимым определять узлы, из которых нет необходимости порождать дерево вариантов. Здесь мы подходим к сути эвристического программирования, центральным понятием которого является *эвристика* – способ или прием сокращения перебора, отсечения неперспективных ветвей на дереве вариантов.

Примером простейшей эвристики является использование симметрий: позиции, получаемые в результате поворотов и отражений уже рассмотренных позиций рассматривать не нужно. Например, в крестиках-ноликах на поле 3x3 существует 3 разных первых хода (см. рис. 3). Сколько существует вариантов игры, если не учитывать симметрии? Их

количество – не более чем факториал числа 9. Благодаря использованию симметрий это число значительно уменьшается. Однако для больших полей выигрыш оказывается значительно меньше, так как уже после выполнения нескольких ходов симметричные продолжения перестают существовать. Понятие эвристики находит выражение в оценивающих функциях, которые позволяют оценивать качество позиций, не следующее непосредственно из правил игры или решаемой задачи.

Рассмотрим процедуру поиска ходов на имплицитном дереве решений с использованием оценивающей функции. Процедура поиска включает порождающую процедуру, (статическую) оценивающую функцию и процедуру формирования рабочих оценок. По определению *(статическая) оценивающая функция* устанавливает вес позиции без порождения каких-либо ее преемников, а *процедура формирования рабочих оценок* присваивает позиции вес, исходя из весов преемников этой позиции.

Зачем нужна процедура формирования рабочих оценок в дополнение к оценивающей функции? Представим, что веса, присваиваемые оценивающей функцией, позволяют в каждой ситуации выбирать наилучший ход. В этом случае задача будет решаться без перебора, то есть будет существовать явная стратегия выигрыша. Такая ситуация действительно возможна. Приведем в качестве примера две задачи, не требующие перебора:

1) Есть кучка из 2007 монеток. Два игрока по очереди берут из нее от одной до шести монеток. Выигрывает тот, кто делает последний ход. Какова выигрышная стратегия?
2) Можно ли замостить доску 8x8 клеток с двумя вырезанными уголками (*A*1 и *H*8), то есть доску из 62 клеток, костяшками домино 1x2?

Очевидно, обе задачи могут быть решены полным перебором, деревья вариантов для которых несложно построить. Однако для обеих задач есть беспереборные решения. В первой задаче в качестве инварианта может выступать остаток от деления числа монеток на 7, на чем и строится выигрышная стратегия. Во второй задаче нужно заметить, что при шахматной раскраске доска с вырезанными клетками имеет неравное число белых и черных клеток, а костяшка домина всегда занимает одну белую и одну черную клетку. Итак, не любую задачу, для которой можно построить дерево вариантов, следует решать перебором.

Однако задачи, решаемые в области ИИ, зачастую относятся к т.н. *NP*-полным. Для таких задач не известно точное решение, которое не требовало бы обхода дерева вариантов, количество узлов в котором экспоненциально растет с глубиной. Этот эффект называется *комбинаторным взрывом* или *проклятием размерности* (почему это проклятие, понять несложно: именно невозможность полного перебора на экспоненциально растущем дереве вариантов лишает переборные алгоритмы интеллектуальности, а исследователей – легкого решения проблемы искусственного интеллекта). В связи с этим, для таких задач

также не существует и оценивающих функций, которые бы позволяли на каждом шаге гарантированно выбирать лучший ход. В то же время, конечным позициям (листьям) оценивающая функция должна назначать правильные веса. Таким образом, оценивающая функция должна при приближении к листьям сходиться к «правильному ответу», однако эта сходимость может быть немонотонной.

Рассмотрим простой пример оценивающей функции: суммарную силу фигур в шахматах. Для начала отметим эвристичность этой функции: из правил игры в шахматы напрямую не следует, что большое количество фигур способствует выигрышу. Эта оценивающая функция, очевидно, немонотонна (вернее, не строго возрастает), что проявляется в двух фактах. Во-первых, веса двух ходов будут одинаковы, если эти ходы не приводят к изменению количества фигур, то есть с использованием данной оценивающей функции нельзя отдать предпочтение какому-либо ходу без съедения фигуры. Во-вторых, часто съедение какой-либо фигуры приводит к потере более значимой фигуры на следующем ходе, в результате чего значение данной оценивающей функции сначала увеличится, а затем уменьшится.

Уже на этом примере видно, что немонотонность оценивающей функции может быть компенсирована перебором, на основе которого и формируются рабочие оценки.

Вопросы и упражнения

1. Чем отличается дерево игры от дерева целей?
2. По какой причине оказывается необходимым вводить понятие эвристики?
3. Оцените примерные размеры полного дерева вариантов для игры «пять в ряд» (крестики-нолики) на поле 15x15.
4. Какие деревья вариантов бывают?
5. Какие основные стратегии порождения дерева вариантов вы знаете?

3. Процедуры формирования рабочих оценок. Общий решатель задач

При вычислении оценивающей функции ни для одного из узлов не производится порождение потомков. Статическая оценивающая функция практически полностью зависит от задачи, и в нее вкладывается знание экспертов. К примеру, в игре в шашки и в поддавки стратегии полностью различны, что выражается в разных оценивающих функциях. То есть любое изменение правил тут же сказывается на том, какую оценивающую функцию следует использовать. Как мы видели раньше, качество оценивающей функции очень сильно влияет на эффективность работы программы. Однако хоть сколько-нибудь общего подхода к автоматическому построению оценивающих функций не известно.

Некоторые существующие методы базируются на накоплении статистики. В простейшем случае игровая позиция описывается некоторой совокупностью признаков. Далее просматривается большая база примеров партий (или решений каких-то задач) и определяется, какие значения каких признаков приводили к успеху или неудаче. Тогда качество новой позиции вычисляется, исходя из того, соответствуют ли текущие значения признаков позициям, которые привели к выигрышу или проигрышу. Хотя такой подход и дает некоторые результаты, он сильно ограничен. К примеру, человеку не нужно играть сотни игр для того, чтобы понять, что потеря фигур обычно нежелательна, то есть он способен делать некоторые заключения об оценивающей функции, анализируя сами правила игры. Другое ограничение подхода заключается в том, что задание релевантных признаков необходимо выполнять человеку.

В отличие от оценивающей функции процедуры формирования рабочих оценок могут быть достаточно проблемно-независимые и поддаются теоретическому анализу. Назначение этих процедур заключается в уточнении качества позиции, соответствующей текущему узлу в дереве, на основе значений оценивающей функции в дочерних узлах. Различия между этими процедурами заключается в том, какие именно дочерние узлы просматриваются и как именно комбинируются значения оценивающей функции, полученные для них.

В простейшем случае просмотр осуществляется на фиксированную глубину одинаково для всех потомков. Далее оценки, полученные для наиболее глубоких узлов, распространяются на родительские узлы. В игровых задачах способ такого распространения, как правило, связан с принципом *минимакса*, который заключается в следующем. Для узла, в котором осуществляется ход игрока, помещается рабочая оценка, равная рабочей оценке, максимальной среди всех непосредственных потомков. Для узла, соответствующего ходу противника, напротив, рабочая оценка формируется как минимум среди рабочих оценок непосредственных потомков.

Суть минимаксной оценки в том, что использующая ее компьютерная программа ожидает от противника наиболее сильного (причем по оценке самой программы) хода. Минимаксная оценка на первый взгляд может показаться единственно правильной оценкой. Однако нужно понимать, что данная оценка не является логически обоснованной и содержит элемент эвристичности. В действительности, в минимаксную оценку заложено предположение, согласно которому противник «думает» так же, как и компьютерная программа. Это не что иное, как модель противника. При отсутствии дополнительной информации о противнике такая модель должна быть принята по умолчанию. Однако она не учитывает ни стиль игры противника, ни его силу. Это наиболее ярко проявляется при игре с форой против как более сильного, так и более слабого противника. В утрированном случае, компьютерная программа, давшая фору противнику

и способная осуществить полный перебор, должна была бы сразу сдаться, если бы она руководствовалась минимаксной оценкой.

Несмотря на все упрощения, минимаксная оценка весьма удобна. При этом она может быть модифицирована, в частности, за счет учета числа и степени различия потомков данного узла. Так, позиция, предоставляющая малое количество альтернатив, может считаться предпочтительной, так как позволяет вести более глубокий поиск. Учет числа потомков может быть отнесен и к статической оценивающей функции, однако он непосредственно связан со стратегией поиска, а не с решаемой задачей, поэтому он чаще относится именно к процедуре формирования рабочих оценок.

Поиск на фиксированную глубину (или при фиксированном числе просматриваемых потомков) по всей ширине дерева вариантов является наиболее простым. В зависимости от глубины поиск варьируется от полного перебора до так называемых *жадных алгоритмов*, глубина просмотра в которых равна единице (то есть рабочая оценка совпадает со статической). Условно можно считать, что жадные алгоритмы просматривают лишь одну ветвь дерева до самого конца, выбирая в каждом узле локально оптимальную ветвь. Жадные алгоритмы могут быть весьма эффективными, однако, как отмечалось ранее, оценивающая функция может изменяться весьма немонотонно, то есть жадные алгоритмы могут приводить к очень плохим результатам.

Другой проблемой для жадных алгоритмов являются «плато» в оценивающих функциях, то есть такие позиции, в которых нет ходов, сколько-нибудь значащим образом меняющих ситуацию. К примеру, в шахматах оценивающая функция, базирующаяся только на силе оставшихся фигур, будет изменяться только при выполнении ходов со съеданием фигур. Очевидно, жадный алгоритм, использующий такую оценивающую функцию, будет играть абсолютно хаотично, когда нет возможности съесть какую-нибудь фигуру, и есть фигуры противника при первой же возможности. Включение оценки позиционного преимущества позволяет сделать оценивающую функцию более гладкой, а игру на основе жадного алгоритма более разумной. Здесь видно, что улучшение (и усложнение) оценивающей функции заменяет углубление перебора при формировании рабочих оценок.

Однако улучшать можно не только оценивающую функцию. Процедура поиска также может быть улучшена без увеличения числа просматриваемых потомков. Следующими по сложности после простого просмотра на фиксированную глубину процедурами является процедуры *направленного сокращения*. Это класс процедур формирования рабочих оценок, в которых исследуются не все потомки данной позиции, а только некоторые из них, причем выбор рассматриваемых потомков осуществляется на основе их статических оценок. За счет того, что неперспективные ветви не рассматриваются, перспективные ветви могут быть исследованы глубже. В *процедуре n-наилучшего направленного*

сокращения для каждого узла просматривается *n* потомков с максимальными (или минимальными при ходе соперника) значениями статической оценивающей функции.

Важность такого сокращения достаточно очевидна. К примеру, если в игре го на каждом ходе есть выбор между, скажем, ста вариантами постановки камня, то при просмотре лишь десяти наиболее перспективных ходов глубину поиска можно увеличить в два раза (например, с десяти до двадцати) без увеличения числа посещаемых вершин. При этом, однако, возникает опасность исключить хороший ход. Например, в шахматах часто возникают позиции, при которых можно отдать ферзя (или другую фигуру), чтобы следующим ходом поставить мат. Отдача ферзя существенно уменьшает значение статической оценивающей функции, так что этот ход, вероятно, будет исключен из рассмотрения.

Для уменьшения риска пропуска хорошего решения могут использоваться процедуры *суживающего n-наилучшего направленного сокращения*, в которых значение просматриваемых ветвей *n* уменьшается по мере продвижения вглубь по дереву вариантов. Выбор закона, по которому уменьшается значение *n* с ростом глубины рассматриваемых узлов, производится эмпирически, то есть на основе экспериментов.

Существует и более изощренный способ уменьшить риск пропуска важных узлов. Можно заметить, что процедура формирования рабочих оценок призвана улучшить статическую оценивающую функцию за счет перебора. Однако в *n*-наилучшем направленном сокращении отсечение ветвей ведется на основе значений (неулучшенной) оценивающей функции. Вместо этого можно осуществлять *неглубокий поиск* в целях получения более адекватных оценок для отсечения неперспективных ветвей. Итак, сначала осуществляется неглубокий, но широкий поиск для уточнения значений оценивающей функции, затем эти значения используются для осуществления более узкого и более глубокого поиска, который позволяет далее уточнить рабочие оценки и т.д. Возможно формирование целой иерархии поисков, хотя обычно ограничиваются лишь двумя уровнями.

Помимо выбора ветвей для отсечения, эффективность поиска можно повышать и за счет более адекватного выбора числа отсекаемых ветвей. Действительно, в процедуре *n*-наилучшего направленного сокращения, как и в других рассмотренных процедурах, значение *n* является заданным априори и не зависит от того, из какого именно узла начинается перебор и какие промежуточные результаты получены в процессе этого перебора.

Некоторые задачи допускают более эффективные процедуры отсечения ветвей, в которых при выборе значения *n* учитывается дополнительная информация. Наиболее широко известной является *процедура альфа-бета отсечения* (также называемая *методом ветвей и границ*). Идея этой процедуры заключается в том, чтобы принятие решения об отсечении некоторой ветви на дереве вариантов основывалось на сравнении качества соответствующей позиции с качеством ранее

найденного решения. Качество уже найденного (но, возможно, не лучшего) решения – это текущее значение параметра α. Если оптимистическая оценка (то есть оценка сверху) для данного узла меньше значения α, то рассматривать ветвь, выходящую из этого узла, не имеет смысла. Второй параметр, β, используется в игровых задачах для ограничения ветвей, соответствующих ходам противника (то есть этот параметр характеризует пессимистическую оценку качества позиции).

Для многих задач (особенно, игровых) получить точные оценки сверху и снизу для качества некоторой позиции оказывается затруднительно, в связи с этим в процедуре альфа-бета отсечения могут использоваться и нестрогие (эвристические) оптимистические и пессимистические оценки. Рассмотрим пример задачи, допускающей использование строгих оценок сверху в процедуре альфа отсечения. Это т.н. задача о рюкзаке.

Эта задача заключается в следующем. Дано N предметов, для каждого из которых определен вес w_i и ценность v_i, i=1,…,N. Необходимо выбрать такой набор предметов, который бы обладал максимальной суммарной ценностью, и при этом общий вес предметов не превосходил некоторого значения W. Данная задача решается перебором. Предметы обычно сортируются по удельной ценности v_i/w_i (это является очевидной для человека эвристикой, но ее нужно закладывать в алгоритм априорно).

Жадный алгоритм работает следующим образом: выбирается предмет максимальной ценности, который не нарушает ограничения по весу, после чего производится переход к выбору следующего предмета. Этот алгоритм не гарантирует нахождения оптимального решения, что хорошо видно на следующем примере: N=3, w_1=5, v_1=10, w_2=4, v_2=7, w_3=4, v_3=7, W=8. В данной ситуации сначала будет выбран первый предмет, после чего ни один предмет больше взят быть не может.

Жадный алгоритм, однако, может дать неплохое решение, которое может использоваться для установки исходного значения параметра α. Далее осуществляется полный перебор. При этом в каждом узле может быть определена оптимистичная оценка: набор предметов, соответствующий данному узлу дополняется следующими по ценности предметами до тех пор, пока не нарушается ограничение по весу; если следующий по ценности предмет это ограничение нарушает, то берется часть этого предмета (то есть при сохранении удельной стоимости вес предмета уменьшается так, чтобы суммарный вес предметов оказался равным W). Несложно убедиться, что это строгая оценка сверху. Если для какого-то узла эта оценка оказывается меньше текущего значения α, то этот узел не рассматривается. Если в процессе перебора получается решение, которое лучше решения, полученного жадным алгоритмом, то значение α обновляется.

Альфа-бета процедура часто совмещается с другими процедурами поиска. К примеру, исходная оценка значений α и β может быть получена процедурой n-наилучшего направленного сокращения с малым значением

n, что позволяет все еще достаточно быстро получить ограничение, более жесткое, чем при использовании жадного алгоритма. В результате, при последующем переборе будет отсекаться больше ветвей, и он будет более эффективным (но если значение *n* взять большим, то поиск первого решения будет долгим и выигрыша не будет). Для перебора, осуществляемого после определения начальных значений α и β, могут также использоваться разные процедуры, что порождает широкий спектр возможностей по реализации поиска в рамках эвристического программирования.

Хотя процедуры формирования рабочих оценок сравнительно универсальны, сам поиск опирается на статическую оценивающую функцию или другие предметно-зависимые эвристики. В результате, системы, построенные на основе идей эвристического программирования, оказываются неспособными самостоятельно решать любую новую (даже весьма простую) задачу независимо от того, сколько задач они уже могут решать. Вряд ли подобные системы можно назвать действительно интеллектуальными. В связи с этим перед исследователями возникла проблема создания универсальных средств решения интеллектуальных задач.

Исходно такие системы разрабатывались в рамках еще доминировавшей парадигмы эвристического программирования. Одна из первых и наиболее известных систем такого рода – это «Общий решатель задач» (GPS, General problem solver), первая версия которого была разработана Аланом Ньюэллом и Гербертом Саймоном (при участии Кристофера Шоу) в 1957 году (развитие и исследование программы осуществлялось до 1969 года) на основе ранее разработанной ими программы «Логик-теоретик».

Основной особенностью GPS является то, что помимо описания конкретной задачи этой программе также в явном виде задается описание и проблемной среды. GPS решает задачи следующего типа: поиск цепочки допустимых действий, приводящих заданную начальную ситуацию к желаемой конечной ситуации (цели). Каждая ситуация описывается как совокупность объектов, находящихся в определенных состояниях. То есть задача для GPS описывается как совокупность имеющихся и желаемых состояний набора объектов. Допустимые действия трактуются как операторы. Чтобы GPS был способным решать задачи такого рода, он должен обладать описанием проблемной среды, которая включает всю общую для задач определенного типа информацию, например, информацию о проблеме интегрирования.

Для понимания условия конкретной задачи необходима информация следующего рода: описание операторов (например, при решении шахматных задач операторы должны описывать разрешенные правилами игры ходы) и описание различий между объектами (GPS должен иметь возможность определять, чем именно отличается одна ситуация от другой). Помимо этого описание проблемной среды для GPS включает

дополнительную информацию, которая, по сути, задает эвристики поиска. Сюда относится таблица связей, которая содержит информацию о том, какие операторы могут использоваться для уменьшения каких различий, и информация об упорядочении различий по степени значимости (или трудности их устранения). Хотя GPS тоже требует задания эвристик, они являются внешними по отношению к программе, в связи с чем применение GPS к новой предметной области не требует его перепрограммирования.

На основе всей указанной информации GPS решает задачи, используя общий метод, называемый *анализом целей и средств*. Этот метод заключается в направленном построении дерева целей, где подцели выбираются таким образом, чтобы уменьшить различие между имеющейся и желаемой ситуацией. С каждой подцелью связывается некоторая величина, характеризующая трудность в ее достижении, которая определяется по трудности устранения соответствующих различий. В реализации GPS присутствует множество технических деталей, которые здесь рассматриваться не будут.

GPS был способен решать широкий круг задач, включая аналитическое интегрирование, некоторые шахматные и геометрические задачи, задачи планирования и т.д., хотя он решал только корректно поставленные задачи. В то же время, GPS оказался не столь универсальным, как предполагали его создатели, что имеет несколько причин. Во-первых, язык описания проблемной среды оказался сравнительно бедным, и некоторые задачи не поддавались формализации в его рамках. Во-вторых, помимо собственно описания предметной среды требовалось также снабжать GPS и некоторыми предметно-зависимыми эвристиками, часть из которых все же приходилось задавать в виде программного кода. В-третьих, универсальные процедуры поиска оказались не достаточно эффективными при решении сложных задач, характеризующихся большими деревьями вариантов. В частности, GPS не мог играть в шахматы, поскольку не удавалось разбить конечную цель (выигрыш в партии) на непосредственно достижимые подцели.

Тем не менее, хотя GPS и другие аналогичные программы потерпели неудачу в качестве универсальных решателей задач, они позволили глубже понять проблему ИИ и послужили одним из факторов, приведших к смене парадигм. А именно, в них была выявлена необходимость исследования способов представления и манипуляции знаниями о предметной области.

Вопросы и упражнения

1. В чем заключаются отличия процедуры формирования рабочих оценок от статической оценивающей функции?
2. Если для каждого хода в некоторой игре есть возможность выбора одного из 64 вариантов, то во сколько раз более глубокий поиск обеспечит процедура *n*-наилучшего направленного сокращения при

n=4 по сравнению с процедурой неполного перебора на фиксированную глубину без отсечения ветвей?
3. Для каких целей используется неглубокий поиск?
4. Каково основное отличие алгоритма альфа-бета отсечения от процедуры n-наилучшего направленного сокращения?
5. Какой недостаток классических методов эвристического программирования был призван преодолеть Общий Решатель Задач?
6. Какие выводы для области искусственного интеллекта позволил сделать общий решатель задач?

4. Методы градиентного спуска и имитации отжига

Поиск на дереве вариантов является универсальным средством решения задач, однако в силу своей универсальности он оказывается малоэффективным для многих частных классов задач поиска. Рассмотрим в качестве одного примера такого класса задач проблему поиска пути.

Пусть дана доска MxN клеток. В каждой клетке указана некоторая стоимость ее посещения $c_{i,j}$, i=1,…,M, j=1,…,N. Требуется найти такой путь из клетки с координатами (1,1) в клетку с координатами (M,N), стоимость которого была бы минимальной. Здесь путь – это такая последовательность клеток, что расстояние между последующей и предыдущей клеткой равно единице (т.е. клетки являются 4-связанными).

Для этой задачи несложно сформировать дерево перебора: корень дерева соответствует начальной клетке, а из каждого узла дерева исходят ветви, соответствующие всем возможным перемещениям из текущей клетки в следующую. Количество листьев на дереве вариантов равно количеству всех возможных путей от начальной клетки до конечной. Чтобы число листьев было конечно, можно искать только пути, в которых любая клетка посещается не более одного раза. Однако даже с учетом этого ограничения количество путей растет чрезвычайно быстро с ростом размера доски, что делает невозможным их полный перебор даже для сравнительно небольших значений M и N.

К счастью, поиск наилучшего пути и не требует полного перебора. Существует следующий простой алгоритм:

1. Создать массив **D** размера MxN и присвоить его элементам $d_{i,j}$ значения $+\infty$.

2. Изменить значение элемента (1, 1) на $c_{1,1}$.

3. Для каждого элемента (i, j), значение которого было изменено на предыдущем шаге алгоритма, просмотреть его соседей и сравнить величины $d_{i,j}+c_{i\pm1,j\pm1}$ с соответствующими текущими значениями $d_{i\pm1,j\pm1}$. В случае если $d_{i\pm1,j\pm1} > d_{i,j}+c_{i\pm1,j\pm1}$ заменить значение элемента ($i\pm1$, $j\pm1$) на $d_{i,j}+c_{i\pm1,j\pm1}$. Обычно среди всех измененных элементов сначала рассматривают элементы с наименьшим значением $d_{i,j}$.

4. Повторять шаг 3 алгоритма до тех пор, пока значения каких-либо элементов массива **D** меняются.

5. Массив **D** будет содержать минимальные стоимости путей от точки (1, 1) до каждой из точек доски, откуда будет несложно восстановить и сам маршрут от начальной до конечной точки.

На рис. 7 представлено пояснение к алгоритму: исходные стоимости посещения клеток и изменение матрицы **D** при выполнении одного шага.

1	3	2	6
7	6	1	8
4	2	1	5
6	1	8	6
7	2	4	5

а)

1	4	6	12
8	10	7	15*
12	12	8*	+∞
18	13*	+∞	+∞
25*	+∞	+∞	+∞

б)

1	4	6	12
8	10	7	15
12	10*	8	13*
18	13	16*	+∞
25	15*	+∞	+∞

в)

Рис. 7. Поиск кратчайшего пути: а) матрица стоимости посещения клеток; б) матрица **D** после пяти итераций; в) матрица **D** после шести итераций. Отмечены клетки, значения $d_{i,j}$ в которых менялись на предыдущей итерации

Несложно убедиться, что массив **D** будет содержать стоимости наилучших путей до каждой клетки. При этом время работы алгоритма линейно зависит от площади доски.

За счет чего возникает такое эффективное решение? Причина заключается в том, что при посещении некоторой клетки нам не важно, каким путем мы в нее пришли, важна лишь стоимость этого пути. Иными словами, длина последующего пути из этой клетки не зависит от уже пройденного пути. Для человека это кажется совершенно очевидным фактом, но его использование не возникает естественным путем в компьютерной программе, а должно быть специально запрограммировано.

Подобное свойство независимости встречается во многих задачах, однако, даже в некоторых разновидностях задачи поиска пути оно может нарушаться. К примеру, если на некоторой местности выполняют перемещение сразу несколько интеллектуальных агентов, которые могут преграждать друг другу путь, то стоимость посещения каждой клетки может меняться со временем, причем зависеть не только от конкретного момента времени, но и от самих траекторий, проделанных агентами (поскольку каждый агент может выбирать свой путь в зависимости от траектории других агентов). В этом случае задача поиска пути

превращается в т.н. задачу отслеживания пути, которая является несравненно более сложной и уже не вполне формализуемой.

Хотя описанный алгоритм поиска пути гораздо быстрее полного перебора всех возможных путей, однако и он может оказаться неэффективным. Представим, что мы уменьшаем размеры клеток на нашей доске, увеличивая их количество (так чтобы линейный размер доски оставался неизменным), пока клеток не станет чрезвычайно много. Тем самым мы получим ситуацию, близкую к реальному миру, в котором из некоторой точки можно попасть в другую точку по произвольной траектории. Осуществлять поиск пути описанным выше алгоритмом (не говоря уже об универсальных алгоритмах поиска на дереве вариантов) в таком пространстве становится практически бессмысленным.

Проблемы поиска в непрерывных пространствах можно выделить в отдельный широкий класс проблем. Для этих проблем можно было бы использовать методы эвристического программирования, однако для того, чтобы эти методы оказались хоть сколько-нибудь эффективными, в них необходимо закладывать особые эвристики, опирающиеся на свойства непрерывных пространств, которые столь сильно влияют на организацию процесса поиска, что методы поиска в непрерывных пространствах удобнее рассматривать отдельно.

Задачи поиска в непрерывном пространстве зачастую сводятся к поиску экстремума некоторой целевой функции (поиск наилучшего пути также можно свести к этой задаче).

Методы поиска экстремума одномерной функции хорошо известны, но они плохо переносятся на многомерный случай, то есть обладают достаточно узким применением, поэтому мы их рассматривать не будем. Отметим лишь, что в ряде случаев поиск экстремума может производиться покоординатно: сначала фиксируются значения всех аргументов функции, кроме первого, и осуществляется поиск экстремума для такого одномерного сечения функции, далее фиксируется следующий аргумент и т.д. При этом могут использоваться методы поиска экстремумов одномерных функций.

Классический метод поиска минимума дифференцируемой многомерной функции с аргументами, принимающими вещественные значения, – это метод градиентного спуска, который также называется методом «подъема в гору» при его использовании для поиска максимума функции. Данный метод, как правило, применяется для многомерных функций, поскольку в одномерном случае существуют более эффективные методы поиска.

Как известно, градиент некоторый функции $f(\mathbf{x})$ в некоторой точке показывает направление локального наискорейшего увеличения функции. Этот факт используется в методах градиентного спуска (подъема).

Эти методы описываются следующей последовательностью действий:

1. Выбрать начальную точку $\mathbf{x}_0 = (x_{1,0},...x_{n,0})$. Установить номер итерации: i=0.
2. Для текущей точки определить значение градиента:

$$\nabla f(\mathbf{x}_i) = \left(\frac{\partial f}{\partial x_1}(\mathbf{x}_i),..., \frac{\partial f}{\partial x_n}(\mathbf{x}_i) \right). \quad (1)$$

   В случае если градиент не может быть вычислен аналитически, его компоненты могут быть оценены численно:

$$\frac{\partial f}{\partial x_j}(\mathbf{x}_i) \approx \frac{f(x_{1,i},...,x_{j-1,i},x_{j,i}+\Delta x,x_{j+1,i},...,x_{n,i}) - f(\mathbf{x}_i)}{\Delta x}. \quad (2)$$

3. Определить положение следующей точки:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - d\frac{\nabla f(\mathbf{x}_i)}{|\nabla f(\mathbf{x}_i)|}, \quad (3)$$

   где d – параметр, определяющий скорость спуска, и положить i=i+1.
4. Перейти к шагу 2, если не выполнен критерий останова.

Существует несколько способов ввода критерия останова. Самый простой – это наложить ограничение на количество итераций. Другие способы связаны с проверкой того, что текущее положение точки или значение функции f меняются мало. При фиксированном шаге d изменение положения текущей точки происходит всегда на одну и ту же величину. Однако в этом случае можно проверять изменение за несколько итераций и сравнивать с шагом d: $\frac{|\mathbf{x}_i - \mathbf{x}_{i-k}|}{kd}$.

Существует также возможность адаптивного выбора шага d. Для этого на каждой итерации осуществляется выбор такого значения из трех величин шага $d_0 = d, d_1 = d/w, d_2 = dw$ (где w – некоторый параметр, как правило $w \in (1,2]$), что значение функции в точке $\mathbf{x}_{i+1}^{(j)} = \mathbf{x}_i - d_j \frac{\nabla f(\mathbf{x}_i)}{|\nabla f(\mathbf{x}_i)|}$ минимально. Таким образом, если при большом шаге d метод градиентного спуска «проскакивает» минимум, то d будет уменьшаться. Уменьшение d ниже заданного порога также служит критерием остановка. Напротив, на пологих участках значение d будет увеличиваться.

При условии существования глобального минимума функции f метод градиентного спуска обычно сходится (за исключением случаев, когда вдоль некоторого направления функция, монотонно убывая, стремится к некоторому конечному пределу при $|\mathbf{x}| \to \infty$). Сходимость метода обеспечивается тем, что на каждой итерации выбирается такая точка $\mathbf{x}_i$, что $f(\mathbf{x}_i) < f(\mathbf{x}_{i-1})$. Метод, однако, не гарантирует нахождения глобального минимума, поскольку при достижении любого локального минимума метод не в состоянии определить направление на более

глубокий минимум (и вообще обнаружить его существование) и останавливается в соответствии с выбранным критерием останова.

В связи с этим, выбор начальной точки может существенным образом сказываться на получаемом результате, поэтому часто градиентный спуск осуществляется из нескольких начальных точек, и выбирается наиболее глубокий найденный минимум. Выбор начальных точек может осуществляться различными способами. В простейшем случае начальные точки выбираются равномерно распределенными в некоторой области пространства, в которой предполагается наличие глобального минимума. В более сложных модификациях градиентный спуск осуществляется иерархически: положения и величины нескольких найденных локальных минимумов аппроксимируются некоторой непрерывной функцией, к которой также применяется метод градиентного спуска (значение градиента для этой функции указывает направление наискорейшего углубления локальных минимумов исходной функции). Для избегания локальных минимумов может также использоваться т.н. стохастический градиентный спуск, основная идея которого заключается в том, чтобы в дополнение к перемещению текущей точки в направлении наискорейшего локального убывания функции также использовать редкие случайные перемещения в произвольном направлении.

Несложно проследить аналогию между градиентным спуском и жадными алгоритмами в эвристическом программировании: в градиентном спуске также выбирается «ход», приводящий к максимальному локальному выигрышу. Однако вместо того, чтобы перебирать все возможные точки в окрестности текущей точки и выбирать из них лучшую (в жадных алгоритмах просматривались все непосредственные потомки текущего узла), в градиентном спуске оказывается достаточным знать лишь значение градиента. Это оказывается возможным благодаря предположению о непрерывности и дифференцируемости целевой функции (стоит заметить, что дифференцируемые функции – это, вообще говоря, очень узкий класс функций). Как и в случае жадных алгоритмов, при применении градиентного спуска возникают проблемы, связанные с «застреванием» в локальном минимуме в случае немонотонных функций, а также с «плато», на котором значение градиента равно нулю, и направление на минимум функции определить невозможно.

Для решения этих проблем был предложен метод моделирования отжига. Этот метод предназначен для поиска глобального минимума некоторой функции $f : S \to R$, где S – некоторое пространство (необязательно непрерывное), элементы которого интерпретируются как состояния некоторой воображаемой физической системы, а значения самой функции – как энергия этой системы $E=f(x)$ в состоянии $x \in S$.

В методе моделирования отжига система в каждый момент времени находится в некотором состоянии x_i, а также обладает некоторой температурой T, которая является управляемым параметром.

На каждой итерации система случайным образом переходит в новое состояние $x_i \to x_{i+1}$. Механизм выбора нового состояния состоит из двух частей:

1. Сначала выбирается состояние x_{i+1} в соответствии с некоторой функцией распределения $g(x_{i+1}, x_i, T)$. Как правило, эта функция зависит только от расстояния $|x_{i+1} - x_i|$, причем с увеличением этого расстояния вероятность перехода понижается.
2. После случайного выбора x_{i+1} проверяется вероятность перехода в это новое состояние $h(\Delta E, T)$, исходя из разности энергий $\Delta E = f(x_{i+1}) - f(x_i)$ и текущей температуры T. Здесь $h(\Delta E, T)$ показывает вероятность перехода в состояние с другой энергией. Проверка производится следующим образом: выбрасывается случайное число из диапазона [0, 1]. Если это число оказывается меньше, чем значение вероятности $h(\Delta E, T)$, то новое состояние x_{i+1} принимается, в противном случае шаг 1 повторяется. Функция $h(\Delta E, T)$, как правило, стремится к 1 при ΔE, стремящемся в минус бесконечность, и стремится к 0 при ΔE, стремящемся в плюс бесконечность (то есть предпочтение в среднем отдается состояниям с меньшей энергией).

Поскольку метод моделирования отжига базируется на физических принципах, функции распределения вероятностей $g(x_{i+1}, x_i, T)$ и $h(\Delta E, T)$ также часто заимствуются из физики. В частности, достаточно популярен больцмановский отжиг, в котором распределения задаются в форме

$$g(x_{i+1}, x_i, T) = (2\pi T)^{-D/2} \exp(-|x_{i+1} - x_i|^2 / 2T), \qquad (4)$$

где D – размерность пространства S;

$$h(\Delta E, T) = \frac{1}{1 + \exp(\Delta E / T)} \approx 1 - \Delta E / T. \qquad (5)$$

Таким образом, температура T определяет, насколько в среднем может меняться текущее состояние x_i, а также то, насколько в среднем может меняться энергия системы при переходе в новое состояние.

Поскольку переход в состояния с меньшей энергией более вероятен, чем переход в состояния с более высокой энергией, то система будет больше времени проводить именно в низкоэнергетических состояниях.

Чтобы обеспечить сходимость системы к некоторому состоянию с наименьшей энергией, температуру системы понижают с переходом к следующей итерации. В больцмановском отжиге применяется следующий закон понижения температуры:

$$T_i = \frac{T_0}{\ln(1 + i)}, \qquad (6)$$

где номер итерации $i > 0$. Больцмановский отжиг гарантирует нахождение глобального минимума для широкого класса целевых функций. Закон (6)

может, однако, потребовать большое число итераций, особенно при больших значениях начальной температуры T_0 и высокой требуемой точности результата. В этом несложно убедиться: за 10^6 итераций происходит уменьшение температуры всего в 14 раз. В связи с этим зачастую используется закон более быстрого понижения температуры:

$$T_i = \frac{T_0}{1+i}, \qquad (7)$$

который, хотя и не гарантирует нахождения глобального минимума, на практике оказывается предпочтительнее.

Начальная температура неявно задает область, в которой будет осуществляться поиск глобального минимума, а также определяет необходимое для сходимости число итераций.

Несложно заметить, что число итераций в больцмановском отжиге растет экспоненциально с увеличением пространства поиска или требуемой точности, что, в некотором смысле, аналогично полному перебору. Использование закона более быстрого убывания температуры можно соотнести с просмотром только части ветвей на дереве вариантов, как в процедуре *n*-наилучшего направленного сокращения.

Метод имитации отжига находит глобальный минимум функции гораздо надежнее метода градиентного спуска, однако последний гораздо быстрее и точнее находит минимум функции в области, в которой функция является вогнутой. На практике зачастую встречаются функции, которые имеют небольшое число локальных минимумов и максимумов, и пространство поиска разбивается на небольшое число областей, в которых функция вогнута. Для таких функций может быть эффективным совместное использование методов градиентного спуска и имитации отжига.

Вопросы и упражнения

1. Какая основная эвристика используется в методах поиска в непрерывных пространствах решений, которая позволяет существенно повысить эффективность этих методов?
2. Какому методу поиска из области эвристического программирования аналогичен градиентный спуск?
3. В чем преимущество адаптивного выбора шага в градиентном спуске?
4. Какие ограничения классического градиентного спуска позволяет преодолеть стохастических или иерархический градиентный спуск?
5. За сколько итераций при Больцмановском отжиге начальная температура уменьшится в 20 раз?
6. Если количество итераций в методе имитации отжига фиксировано, то на что влияет выбор начальной температуры?
7. В каких случаях метод имитации отжига предпочтительнее градиентного спуска?

5. Эволюционные вычисления

Эволюционные вычисления – это совокупность подходов к решению проблемы поиска (оптимизации), в основе которых лежат идеи, почерпнутые из эволюции в живой природе. Для начала, установим связь между эволюцией и процессом поиска.

Существующие методы эволюционных вычислений берут за основу идеи Дарвина. Одна из этих идей гласит: выживает наиболее приспособленный. В действительности, это тавтология (хотя, возможно, не вполне очевидная применительно к природе), поскольку более приспособленный – это, по определению, тот, кто имеет больше шансов выжить. Однако в сочетании с другой идеей – идеей изменчивости видов, первая идея превращается в идею естественного отбора. То, что виды изменяются со временем, не самоочевидно и может быть установлено только из эмпирических данных, поскольку изменчивость должна обеспечиваться какими-то конкретными механизмами.

Еще одна идея – это идея наследственности. Если бы виды возникали произвольно, то со временем могли бы появляться все более приспособленные виды, однако появление новых видов не зависело бы от степени приспособленности уже существующих, ведь при возникновении нового вида заранее нельзя сказать, насколько приспособленным он будет (если этот вид не конструируется сознательно). Наследственность же обеспечивает последовательное улучшение существующих решений за счет изменчивости и естественного отбора (стоит отметить, что здесь неявно присутствует предположение непрерывности: небольшое изменение вида обычно не сильно сказывается на степени его приспособленности).

Таким образом, как хорошо известно, базовые элементы эволюции – это наследственность, изменчивость и естественный отбор. Поскольку в процессе эволюции появляются все более приспособленные виды, эволюцию можно трактовать как поиск максимума некоторой *фитнесс-функции* выживания. Если предположить, что в процессе эволюции в некоторый момент создан совершенный вид, то на нем эволюция прекращается, так как изменение этого вида может привести только к ухудшению его выживаемости (то есть к уменьшению значения фитнесс-функции).

Стоит заметить, что эволюция представляет собой гораздо более сложный процесс, поскольку сам критерий приспособленности меняется в этом процессе. Однако несомненное сходство эволюции с поиском имеется, и это сходство было замечено исследователями в области ИИ.

Перечисленные выше идеи добавляют немногое к классическим методам поиска. Чтобы показать это, достаточно переформулировать метод градиентного спуска в эволюционных терминах. Пусть текущая точка в методе градиентного спуска – это некоторый вид. На каждом шаге градиентного спуска проверяются некоторые точки вблизи текущей

(наследственность и изменчивость) и выбирается из них лучшая (отбор). Стохастический градиентный спуск из многих точек обладает еще большим сходством с эволюцией видов, если ее ограничить лишь идеями наследственности, изменчивости и отбора. На самом деле, еще сам Дарвин называл отбор в сочетании с изменчивостью «спуском с модификацией». Таким образом, обычный градиентный спуск содержит все базовые элементы эволюции. С одной стороны, это еще раз подчеркивает сходство эволюции с поиском, но с другой стороны, ставит вопрос, что же исследователи ИИ нашли нового в эволюции?

Как оказывается, существует множество механизмов эволюции, список которых все еще продолжает пополняться благодаря новым открытиям. Эти механизмы выступают в роли своего рода эвристик эволюционного поиска, обеспечивая его чрезвычайную эффективность. Таким образом, не базовые идеи эволюции, а конкретные ее механизмы (не выводящиеся из этих идей) и представляли интерес для специалистов в области ИИ.

Эти механизмы являются основой нескольких направлений исследований в рамках эволюционных вычислений:

- генетические алгоритмы (ГА);
- эволюционные стратегии;
- эволюционное (генетическое) программирование.

Эти направления тесно связаны и во многом похожи, хотя возникли практически независимо. Концепция генетических алгоритмов была предложена Холландом в начале 1960-х годов (хотя широко известной она стала только с середины 1970-х годов). Идею эволюционных стратегий предложил Реченберг (а чуть позже Шефель) в середине 1970-х годов. Концепция эволюционного программирования была предложена Фогелем, Оуэнсом и Уолшем в середине 1960-х годов.

Мы рассмотрим сначала генетические алгоритмы, а затем укажем сходство и различия других направлений с ГА.

ГА предназначены для нахождения экстремумов функций от произвольных объектов. Сами объекты трактуются как некоторые организмы, а оптимизируемая функция – как приспособленность организмов, или фитнесс-функция. Множество возможных объектов отображается (предпочтительно, взаимно однозначно) на некоторое подмножество множества битовых строк (обычно фиксированной длины). Эти строки трактуются как *хромосомы* (или геномы).

Особенность генетических алгоритмов заключается в том, что они работают с битовыми строками, не опираясь на структуру исходных объектов, что позволяет применять ГА без модификации для любых объектов. Единственно, что требуется для такого применения, – это процедура перекодирования объектов в геномы.

Двумя основными механизмами эволюции (наряду с идеей отбора), наиболее часто моделируемыми в генетических алгоритмах, являются скрещивание и мутации. Алгоритмическая реализация этих механизмов

называется *генетическими операторами*, к которым относится *оператор скрещивания*, *оператор мутации* и *оператор редукции* (селекции или отбора). При этом последовательность шагов в генетическом алгоритме выглядит следующим образом.

1. Сгенерировать начальную популяцию (случайную совокупность объектов).
2. Выбрать родительские пары.
3. Для каждой родительской пары с использованием оператора скрещивания породить потомство.
4. В хромосомы порожденного потомства внести случайные искажения оператором мутации.
5. Произвести отбор особей из популяции по значению их фитнесс-функции, применив оператор редукции.
6. Повторять шаги 2-4, пока не выполнится критерий остановки.

Рассмотрим каждый из шагов чуть подробнее.

1. Генерация начальной популяции обычно производится равномерно по пространству генов (или по пространству описаний объектов). Размер популяции – установочный параметр.

2. Выбор родительских пар может осуществляться различными способами. Этот выбор осуществляется в два этапа: выбор первого родителя и формирование пары. При выборе одного родителя обычно используется один из следующих способов:

- с равной вероятностью выбирается любая особь из имеющейся популяции;
- особь выбирается случайно с вероятностью, пропорциональной значению фитнесс-функции; то есть в этом случае значение фитнесс-функции сказывается не только на том, какие особи останутся в популяции в результате отбора, но и на то, сколько потомства они произведут.

Выбор второго родителя осуществляется по одному из следующих критериев:

- независимо от уже выбранного родителя (то есть второй родитель выбирается абсолютно так же, как и первый); этот вид отбора называется *неселективным*;
- на основе *ближнего родства*;
- на основе *дальнего родства*.

В последних двух случаях выбор одного родителя влияет на выбор другого родителя: с большей вероятностью формируются пары, состоящие из особей, которые больше похожи друг на друга (то есть ближе находятся в пространстве геномов или описаний объектов) в случае использования ближнего родства и меньше похожи – в случае дальнего родства. В генетических алгоритмах в качестве меры близости обычно используется расстояние Хемминга.

3. Оператор скрещивания – это оператор, который определяет, как из хромосом родителей формировать хромосомы их потомства. Часто применяется следующий оператор скрещивания: хромосомы делятся в некоторой случайной точке и обмениваются этими участками (то есть, все, что идет до этой точки, берется от одного родителя, а все, что после, – от другого). Это одноточечный кроссинговер. В многоточечном кроссинговере таких участков обмена больше.

При равномерном скрещивании каждый бит хромосомы берется от случайного родителя или родители случайным образом обмениваются некоторыми битами (рис. 8).

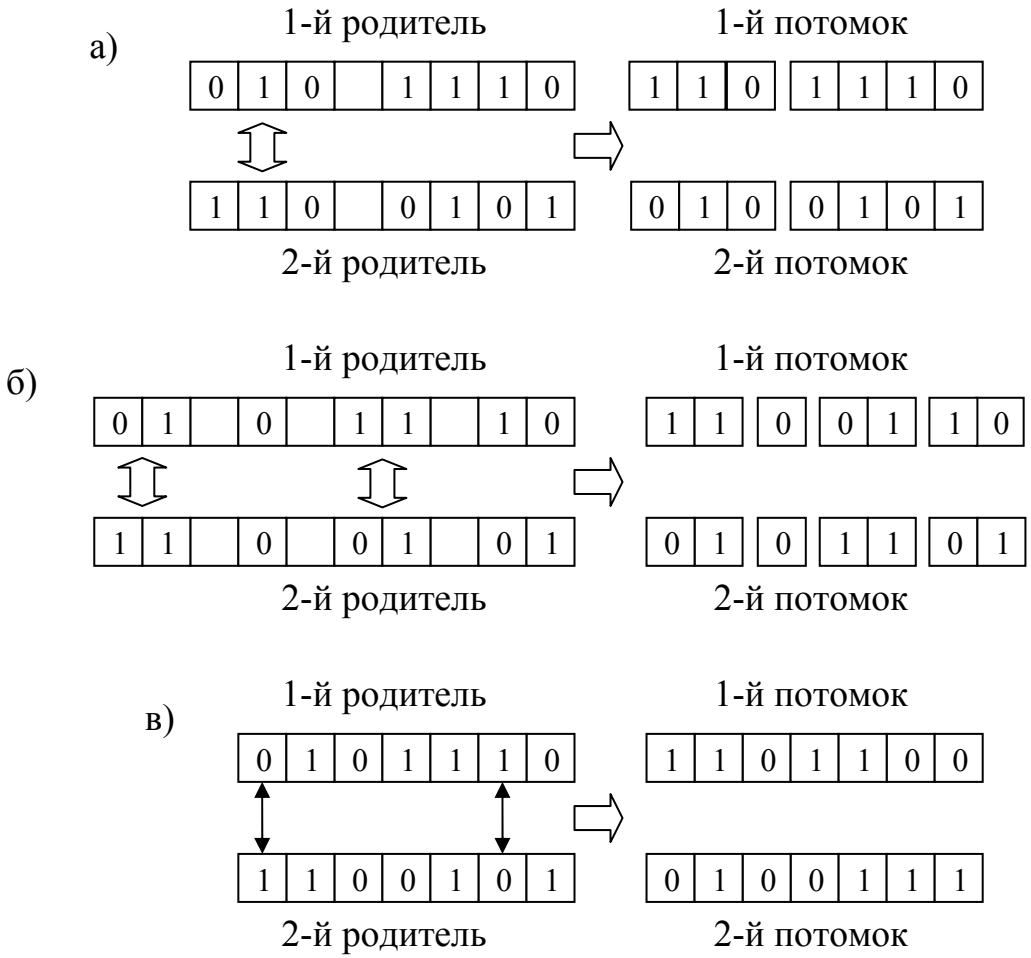


Рис. 8. Примеры работы оператора скрещивания: а) одноточечного; б) многоточечного; в) обмена случайными битами

4. Мутации обычно осуществляются, как случайная замена одного бита хромосомы. Скорость мутаций определяется тем, как часто они происходят. Это управляемый параметр, который влияет на скорость сходимости и вероятность попадания в локальный экстремум.

5. Отбор особей в новую популяцию чаще всего осуществляется одной из двух стратегий:

- пропорциональный отбор, при котором вероятность того, что некая особь останется в следующей популяции, пропорциональна значению фитнесс-функции этой особи;
- элитный отбор, при котором из популяции отбираются лучшие по значению фитнесс-функции особи, и они детерминированным образом переходят в следующую популяцию.

Формирование новой популяции может осуществляться как на основе потомков и родителей, так и на основе только потомков в зависимости от конкретной реализации.

6. Основные критерии останова базируются либо на числе сменившихся поколений (количестве выполненных итераций), либо на некотором условии стабильности популяции. Число поколений не является адаптивным по отношению к виду фитнесс-функции, поэтому используется обычно в качестве вспомогательного критерия. Проверка стабильности популяции в общем виде, как правило, требует значительных вычислений, поэтому чаще используется проверка того, что максимальное по популяции значение фитнесс-функции перестает заметно расти от поколения к поколению. Все эти критерии останова соответствуют критериям останова в методе градиентного спуска, но учитывают ту специфику генетических алгоритмов, что в них на каждой итерации одновременно рассматривается не одно, а много решений.

Отличие эволюционных стратегий от ГА заключается в том, что в них не используются битовые представления. Вместо этого все генетические операторы реализуются в исходном пространстве объектов с учетом их структуры. Это ведет к потере универсальности, но повышению гибкости. Рассмотрим особенности реализации генетических операторов в эволюционных стратегиях на примере объектов, описаниями которых являются двухкомпонентные векторы: (x,y).

1. Генерация начальной популяции может осуществляться путем выбора случайных векторов из области $[x_{\min}, x_{\max}]\times[y_{\min}, y_{\max}]$, где величины $x_{\min}, x_{\max}, y_{\min}, y_{\max}$ задают ожидаемые минимальные и максимальные значения переменных x и y искомого положения экстремума фитнесс-функции. В случае генетических алгоритмов эта область задается неявно, и она зависит от способа отображения вектора (x,y) в битовую строку.

2. При выборе родителей особенность эволюционных стратегий выражается в способе задания меры родства. В данном случае мерой родства двух особей (x_1, y_1) и (x_2, y_2) может служить евклидово расстояние: $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$, которое будет заметно отличаться от расстояния Хемминга, используемого в ГА.

3. Результатом скрещивания двух особей в рассматриваемом случае будет являться особь $((\varepsilon x_1 + (1-\varepsilon)x_2, \varepsilon y_1 + (1-\varepsilon)y_2))$, где $\varepsilon \in [0,1]$ –

случайная величина, что, опять же, отличается от результата скрещивания в пространстве геномов.

4. Результатом мутации для особи (x, y) будет являться особь $(x + \delta_x, y + \delta_y)$, где δ_x, δ_y – случайные величины. Их распределение вероятностей может быть выбрано гауссовым или, для простоты программной реализации, равномерным в некотором интервале. Дисперсия этих величин определяет скорость мутаций.

5, 6. Операторы отбора и критерии останова в эволюционных стратегиях не имеют особых отличий от тех, которые используются в генетических алгоритмах.

Как видно, эволюционные стратегии допускают гораздо более гибкую настройку генетических операторов и позволяют учитывать особенности конкретных объектов, что позволяет повысить их эффективность по сравнению с ГА. Это, однако, происходит за счет потери универсальности: реализация операторов в ГА не зависит от того, фитнесс-функцию от каких именно объектов необходимо оптимизировать, в то время как для эволюционных стратегий каждый тип объектов требует индивидуальной реализации операторов.

Еще одно направление в области эволюционных вычислений – это эволюционное (или генетическое) программирование, которое отличается тем, что в нем рассматриваются вопросы поиска (с использованием принципов эволюции) программ, наилучшим образом удовлетворяющих некоторому критерию. При этом для вычисления фитнесс-функции для некоторой программы необходимо ее исполнять.

Эта проблема выделяется в отдельное направление исследований, поскольку в ней возникают специфические трудности в реализации операторов скрещивания и мутации. Здесь могут применяться как эволюционные стратегии, так и генетические алгоритмы. В случае ГА программа кодируется в виде битовой строки. При этом возникает проблема, заключающаяся в том, что в результате скрещивания или мутации может получиться некорректная программа. Для эволюционных стратегий программы тоже должны быть представлены некоторым образом, удобным для выполнения операторов над ними.

Часто используются не программы на каком-то обычном языке программирования, а разрабатывается некий язык со специфическим синтаксисом, упрощающий работу в рамках эволюционных методов. Зачастую этот язык не является алгоритмически полным. Популярным также является представление программ в виде нейронных сетей или в рамках других графовых представлений.

Наиболее простым в реализации является случай, когда структура нейронной сети фиксирована, и требуется лишь настроить веса ее связей. В геноме тогда кодируются только веса связей, так что длина генома оказывается одинаковой для всех особей, и не возникает опасности порождения некорректного генетического кода в результате применения

генетических операторов. Это, однако, приводит к сильному ограничению пространства поиска. Поиск на менее ограниченном множестве программ является все еще плохо изученной проблемой.

Схема работы генетического алгоритма (и других эволюционных методов) достаточно проста и понятна с точки зрения классических методов поиска. Естественная эволюция гораздо сложнее и содержит в себе многие эвристики поиска, которые еще не были перенесены в эволюционные вычисления.

К примеру, идеи ближнего и дальнего родства реализуются в эволюции одновременно: скрещивание возможно только между представителями одного вида (ближнее родство), но внутри одного вида из-за существования летальных и полулетальных генов скрещивание близких родственников зачастую приводит к порождению потомков с низким значением фитнесс-функции (дальнее родство). На примере эволюционных вычислений хорошо видно, почему это должно быть так. Если представить, что каждый вид располагается на своей моде фитнесс-функции, то межвидовое скрещивание бессмысленно, так как результат попадет между модами. Скрещивание же практически идентичных особей внутри вида также не приводит к оптимизации.

Идея рецессивных и доминантных генов также не нашла еще применения в эволюционных вычислениях. Существование большого количества генов, не кодирующих фенотипические признаки, а управляющих экспрессией других генов, лишь недавно заинтересовало специалистов в области ИИ. Интересно отметить, что в природе и скорость мутации разных генов различна и зависит от многих факторов, в частности, гены, изменение которых мало сказывается на значении фитнесс-функции, мутируют быстрее. Иными словами, изменения в генотипе не столь независимы от фенотипа, как это принято считать при проектировании ГА.

Этим список эвристик поиска в естественной эволюции далеко не ограничивается. Более того, чрезвычайно интересным является тот факт, что даже те эвристики, которые принимаются в ГА априори, например скрещивание, появились в процессе эволюции не сразу, а «изобретались» с течением времени (в эвристическом же программировании проблема автоматического построения эвристик поиска до сих пор не имеет сколько-нибудь удовлетворительного решения).

Возвращаясь к метафоре «мышление как поиск» можно прийти к выводу, что между мышлением и эволюцией имеется много общего: эволюция представляет собой крайне эффективный направленный поиск (не являющийся ни полным перебором, ни чисто случайным поиском), хотя и функционирующий лишь в единственном пространстве поиска (тогда как интеллект способен выполнять поиск для совершенно разных задач).

В связи с тем, что даже крайне упрощенные, и возможно не самые важные, механизмы эволюции оказались весьма полезными при разработке

систем ИИ, эволюция (в дополнение к естественному интеллекту) сама по себе является достойным предметом изучения в рамках исследований в области ИИ. Более того, и сам интеллект возник в ходе эволюции, причем это стало возможным именно благодаря чрезвычайной эффективности эволюционного поиска и его кардинальному отличию от чисто случайного поиска, полного перебора или жадного поиска, в ходе которых возникновение живых организмов было бы вообще вряд ли возможно.

Интересно отметить, что дарвиновскую теорию эволюции, неверно отождествляя с чисто случайным поиском, часто упрекают именно в том, что случайное возникновение жизни или ее сложных форм было бы крайне маловероятно. Утверждение о невозможности случайного развития жизни верно, но к теории эволюции отношения не имеет, поскольку эволюция не случайна, а характеризуется сложными законами. Исследованию законов эволюции в рамках ИИ посвящено направление, называемое «искусственная жизнь».

Вопросы и упражнения

1. Какие основные подходы существуют в области эволюционных вычислений?
2. Какие типичные генетические операторы используются в генетических алгоритмах?
3. В чем основное отличие между генетическими алгоритмами и эволюционными стратегиями?
4. Какой генетический оператор влияет на возможность нахождения глобального экстремума, находящегося вне области, в которой задавалась начальная популяция?
5. Как скорость мутаций влияет на скорость сходимости генетического алгоритма?
6. При каком виде скрещивания скорость сходимости максимальна?

6. Искусственная жизнь и аниматы

К генетическим алгоритмам, эволюционным стратегиям и эволюционному программированию примыкает направление исследований, получившее название «Искусственная жизнь» (artificial life, AL). Оно оформилось под таким названием в конце 1980-х годов, однако большое число работ, которые можно к нему отнести, были выполнены гораздо раньше.

В рамках этого направления создаются своего рода искусственные существа, которые помещаются в некий специально сконструированный «мир». В этом мире искусственные существа «живут» и «эволюционируют». Как правило, это небольшой виртуальный (цифровой) мир с достаточно простыми законами. Изредка в качестве полигона для

функционирования этих существ выбирается реальный цифровой мир, в роли которого чаще всего выступает интернет.

В рамках направления «искусственная жизнь» существа обычно не помещаются в реальный физический мир, так как в нем на настоящий момент невозможно организовать эволюцию искусственных существ. В другом близком направлении исследований, называющимся «Аниматы» или «Адаптивное поведение», в котором реализация эволюции является второстепенной задачей, напротив, искусственные существа достаточно часто реализуются в реальном мире или в виртуальной модели реального мира.

Основной целью исследований в направлении «искусственная жизнь» является раскрытие, формализация и моделирование принципов организации биологической жизни и процесса ее развития в ходе эволюции. При этом в рамках данного направления часто ставится вопрос об исследовании не только жизни в той форме, в которой она есть в конкретных земных условиях, но и той жизни, какой она могла бы быть в принципе.

Работы в направлении «искусственная жизнь» имеют также и инженерные применения (например, решение задач конструирования механизмов с системами управления путем моделирования их эволюции в виртуальной среде для получения оптимальных параметров). Однако такие инженерные приложения тесно примыкают к ранее рассмотренным методам оптимизации на основе эволюционных вычислений и не составляют предмета исследования в рамках направления «искусственная жизнь». Также для моделей искусственной жизни (в отличие от ГА) характерно отсутствие задаваемой в явном виде фитнесс-функции.

Рассмотрим простейший типичный для «искусственной жизни» виртуальный мир с существами. Как и во других многих случаях, этот мир представляет собой прямоугольное поле, разбитое на клетки. В каждой клетке может присутствовать травоядное животное, растение или хищник, или клетка может быть пустой (см. рис. 9).

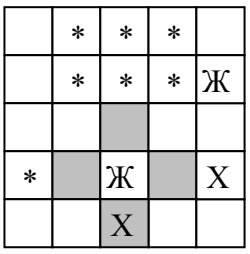


Рис. 9. Пример фрагмента «искусственного мира», состоящего из клеток, в которых может располагаться животное (Ж), хищник (Х) или растение (*); закрашенными отмечены клетки, содержимое которых видит животное

Животное располагается в некоторой клетке и через «сенсоры» получает информацию о содержимом некоторого количества соседних клеток. При этом животное может совершить одно из доступных

действий: остаться на месте или переместиться, а если находится в одной клетке с другим объектом, то произвести взаимодействие с ним, например, съесть его, или напасть на него. Обычно вводится возможность размножения. Это может быть как размножение делением, так и половое размножение. Любое действие требует энергии, которая пополняется за счет поглощения пищи.

Эволюционировать в этом мире могут либо только животные, либо как животные, так и хищники. Эволюционирующими, как правило, делаются не «физические» параметры животных (если, конечно, не решается оптимизационная задача в целях конструирования некоторого реального механизма), а их программы управления (иногда также эволюционным изменениям подвергаются сенсоры). Разнообразие возможных программ управления, способ их кодирования в геноме и реализация генетических операторов, изменяющих программы управления у потомков, полностью зависят от разработчика.

Помимо миров, состоящих из отдельных клеток, в которых организмы занимают целиком одну клетку, также распространено создание миров, в которых положение животных описывается вещественными координатами, а сами животные имеют ненулевой размер и, возможно, некоторую форму. Здесь могут возникать сложные проблемы управления непрерывным движением.

Из-за большого произвола в том, какой именно создавать мир, как описывать программы управления животными и их эволюцию, а также из-за отсутствия какой-либо четкой методологии в данной молодой области исследований большинство работ здесь являются интересными, но не связанными друг с другом экспериментами, интерпретации результатов которых весьма нестрогие.

Надеяться на возникновение разума в подобных искусственных мирах наивно, и мало кто рассматривает в качестве цели создание искусственной жизни как таковой. Так в чем же смысл их конструирования? Как правило, каждый из подобных экспериментов ставится в целях проверки какой-то конкретной идеи или гипотезы об эволюционных механизмах. Именно отталкиваясь от конкретной проверяемой идеи, исследователь выбирает параметры виртуального мира и способ описания управляющих программ. Если «искусственная жизнь» создается без четко осознаваемых целей, то это, хотя и может быть весьма увлекательным, не является исследованием в области ИИ.

Вопросы, которые можно исследовать с помощью «искусственной жизни» весьма разнообразны. Это и влияние априорной информации о мире на выживаемость, и значение случайного поиска, и проверка моделей инстинктов, и возникновение системных явлений при возможности передачи информации между особями, и многое другое.

Например, путем моделирования была проверена концепция эволюционно устойчивых стратегий, согласно которой в результате эволюции возникает не лучшая (для вида в целом) форма поведения

особей, а эволюционно устойчивая, то есть такая, что отклонение поведения отдельной особи от нее оказывается невыгодным для самой особи, а не для вида. К примеру, пусть особям доступны две поведенческие стратегии: кооперация и агрессия. Если обе встретившиеся особи выбрали кооперацию, то они получают небольшой выигрыш. Если одна особь выбирает кооперацию, а другая – агрессию, то первая особь получает большой убыток, а вторая – выигрыш. Если обе особи выбирают агрессию, то получают небольшой убыток. Всеобщая кооперация была бы оптимальна для вида в целом. Но агрессивная особь, случайно появившаяся в такой популяции, будет получать очень большой выигрыш, то есть всеобщая кооперация не является эволюционно устойчивой (в рамках данной модели). В то же время, моделирование показывает, что в конечном итоге образуется популяция со смешанной стратегией: часть особей проявляют агрессию, а часть предпочитают кооперацию (конкретные доли тех и других типов поведения определяются конкретными значениями выигрышей и потерь), что подтверждает тезис об эволюционно устойчивых стратегиях.

Однако нужно еще раз подчеркнуть, что интерпретация результатов таких экспериментов должна быть осторожной. В частности, в этом эксперименте не предусматривалось возможности существования нескольких эволюционно устойчивых стратегий. Если же реализовать несколько конкурирующих социумов, то из них «выиграет» тот, у которого эволюционно устойчивая стратегия также и лучше для вида в целом.

Еще один интересный пример – это проверка с помощью моделей искусственной жизни эффекта Балдвина. Суть этого эффекта в том, что навыки, приобретаемые организмами в течение жизни в результате обучения, через некоторое число поколений оказываются записанными в геном. Этот эффект, на первый взгляд, противоречащий дарвиновской концепции эволюции, предположительно, объясняется следующим образом.

Полагается, что этот эффект работает в два этапа. На первом этапе организмы приобретают некоторый полезный навык в результате обучения и передачи его из поколения в поколение без участия генов. Поскольку навык полезен, особи, способные ему обучиться, начинают превалировать в популяции. На втором этапе элементы этого навыка в результате мутаций появляются в геноме. Если какой-то элемент навыка оказался в геноме, то организму нужно будет в течение жизни уже меньше обучаться для полного формирования навыка, то есть организм этот навык получит раньше, при меньших энергетических и временных затратах. Через некоторое (вероятно, очень большое) количество популяций навык будет полностью переведен в геном. В результате чисто случайных мутаций формирование сложного навыка было бы крайне проблематично, но в рамках данной схемы навык может постепенно изобретаться эволюцией, под «руководством» результатов обучения. Ряд моделей

искусственной жизни подтвердили принципиальную реализуемость эффекта Балдвина на основе классических генетических операторов.

К направлению искусственная жизнь также примыкают такие направления, как «Адаптивное поведение» и клеточные автоматы. В рамках первого из них изучаются более сложные формы поведения, которые часто воплощаются не только в виртуальных мирах, но и в физических устройствах. Эти виртуальные организмы или физические устройства называются аниматами (animat = animal + robot).

Здесь гораздо меньший упор делается на эволюцию, но больший – на адаптацию и обучение. Точнее говоря, предметом исследования в рамках данного направления выступает адаптивное поведение: что оно собой представляет, какова должна быть архитектура систем управления, чтобы они могли обладать способностью приспосабливаться к изменяющейся внешней среде и т.д. Происхождение адаптивного поведения в процессе эволюции – лишь один из вопросов в данном направлении (хотя этот вопрос, возможно, является ключевым). Моделирование эволюции может здесь также использоваться для нахождения оптимальных структур управления аниматами, однако при этом находятся не сами структуры, а их конкретизации, так как пространство поиска (способ описания этих структур) задается исследователем.

Наиболее общая схема взаимодействия анимата со средой представлена на рис. 10.

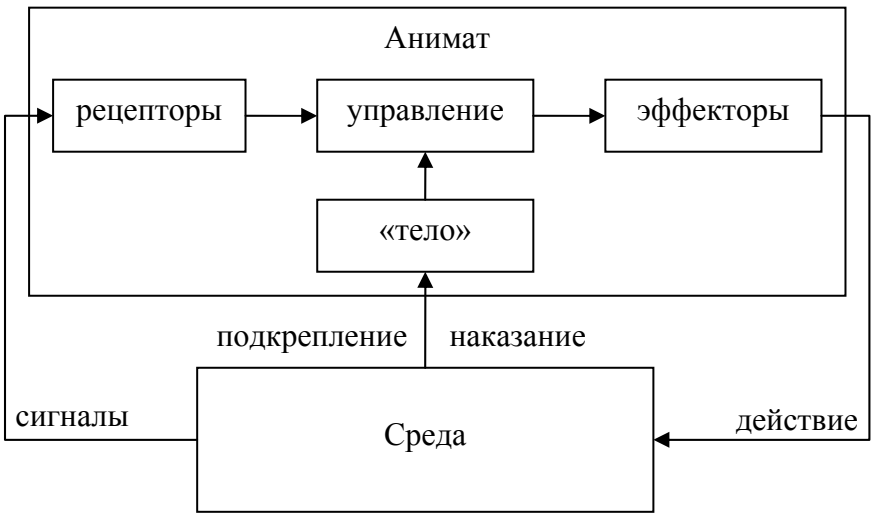


Рис. 10. Схема взаимодействия анимата со средой

Анимат обладает набором эффекторов, с помощью которых может совершать действия, а также «телом», получающим подкрепление и наказание от среды, чем оно и отличается от остальных сенсоров, на которые осуществляется только информационное воздействие. Далее ставится вопрос о конкретизации этих общих блоков (в особенности, системы управления).

Такое уточнение может браться как из работ по исследованию естественных систем адаптивного поведения (к примеру, в этих целях широко используется теория функциональных систем Анохина), так и подбираться искусственно для проверки какой-либо идеи. В качестве примера такой идеи можно привести проверку значимости мотивационных центров, блоков планирования или долговременной памяти.

Клеточные автоматы, в отличие от аниматов, напротив, состоят из максимально простых элементов и функционируют по простым правилам. В классических клеточных автоматах эти правила заключаются в следующем. Есть поле, разбитое на клетки. Каждая клетка поля может находиться в одном из двух состояний: живая и мертвая. Живая клетка умирает, если у нее меньше двух или больше трех живых соседей. Мертвая клетка оживает, если у нее ровно три соседа. Все клетки меняют свои состояния одновременно. Эти правила в какой-то степени отталкиваются от свойств живого: чрезмерная концентрация или разреженность населения не способствует жизни. Существуют также и различные модификации этих правил, однако общие идеи клеточных автоматов от этого не изменяются.

Рассмотрим примеры некоторых интересных конфигураций:

а) исчезающие конфигурации, которые через несколько тактов пропадают (рис. 11);

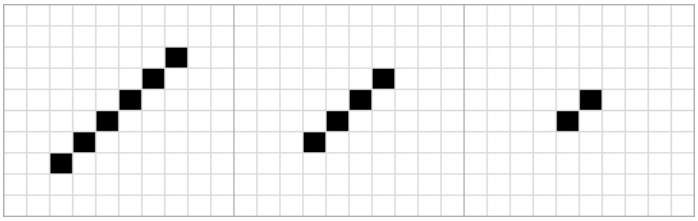


Рис. 11. Конфигурация, исчезающая через три такта

б) статические конфигурации, которые не меняются с течением времени (рис. 12);

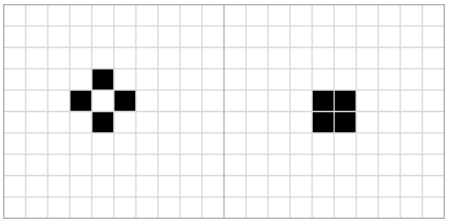


Рис. 12. Две статические конфигурации

в) осцилляторы, которые через несколько тактов возвращаются в исходное состояние (рис. 13);

г) перемещающиеся конфигурации или «бумеранги» (рис. 14);

д) сверхстабильные конфигурации или «пожиратели», которые поглощают «бумеранги» и при этом сами не изменяются;

е) самовоспроизводящиеся конфигурации (они состоят из достаточно большого числа элементов и здесь не приводятся).

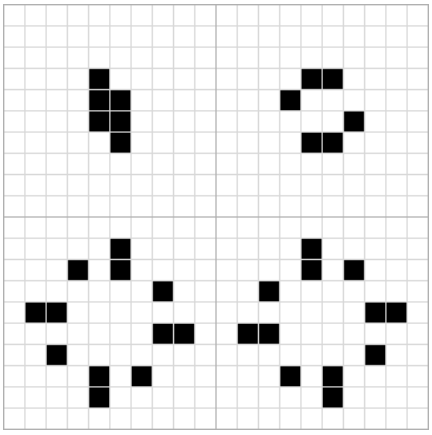


Рис. 13. Две конфигурации, осциллирующие с периодом в 2 такта

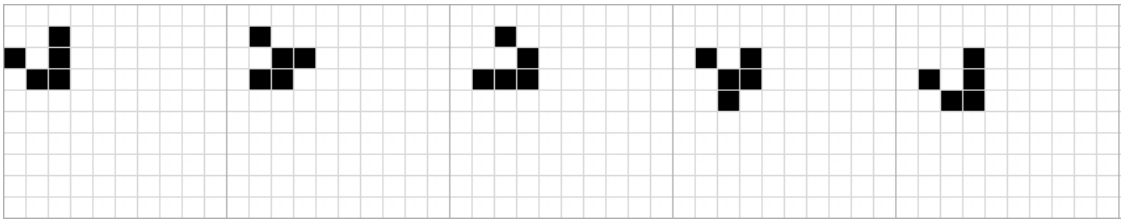


Рис. 14. Конфигурация «бумеранг», вращающаяся и перемещающаяся с периодом в 5 тактов

На примере клеточных автоматов исследуются проблемы самоорганизации, которая характерна для живой (и, на самом деле, мертвой) материи. Простые локальные правила позволяют существовать весьма сложным и протяженным конфигурациям и даже самовоспроизводящимся конфигурациям, что, на первый взгляд, кажется весьма удивительным. Однако это не означает, что происходит настоящая самоорганизация. Как не сложно убедиться на практике, случайные конфигурации обычно достаточно быстро вырождаются в совокупность несвязанных и очень простых статических или осциллирующих конфигураций. В действительности, возможность создавать интересные конфигурации в клеточных автоматах вряд ли удивительнее возможности писать произвольные программы для машины Тьюринга.

Вопросы и упражнения

1. В чем цель направления «искусственная жизнь»?
2. Чем отличаются фитнесс-функции, используемые в моделях «искусственной жизни», от фитнесс-функций, используемых в генетических алгоритмах?
3. В каком направлении исследований используется понятие «анимат»?

4. Каковы основные компоненты аниматов, обеспечивающих их взаимодействие со средой?
5. Какие конфигурации в клеточных автоматах являются исчезающими, статическими, осциллирующими, перемещающимися?

7. Логические системы представления знаний

Исследования в рамках первой парадигмы ИИ «мышление как поиск» позволили убедиться в том, что процесс решения любой задачи может быть представлен как поиск в пространстве состояний, коль скоро задача описана в терминах начального состояния, доступных операциях, изменяющих текущее состояние, и конечного состояния. Однако также было установлено, что формирование пространства состояний и направление процесса поиска должны управляться знаниями о предметной области, к которой относится задача.

Некие экспертные сведенья о задаче всегда закладывались в эвристические программы. Однако зачастую они присутствовали в программах в неявном виде: эвристиках сокращения перебора, способе вычисления оценивающей функции, процедурах проверки особых ситуаций (например, возможного шаха), специфичных для данной задачи, и т.д. При этом эти сведенья «жестко» кодировались в теле программы. Попытки создания универсальных программ, наподобие Общего Решателя Задач, привели к пониманию необходимости отделения знаний от механизмов поиска и явном представлении знаний. В результате стал актуальным вопрос, в каком виде должны представляться знания (и как они представляются в человеческом мозге, как работает человеческая память).

Проблема представления знаний выделилась в отдельную область исследований в рамках ИИ в 1970-х годах. Эта область тесно связана с исследованиями формальных систем в математике, с изучением проблемы смысла языковых высказываний в философии, а также с исследованиями в области когнитивной психологии и лингвистике. Несмотря на такую связь с перечисленными науками, основным критерием в области представления знаний выступает не математическая строгость или психологическая достоверность, а эффективность реализации некоторого представления знаний на практике (что не следует, однако, путать с сугубо прикладным значением).

Поскольку знания стали описываться независимо от исполняемых модулей, а полученные описания затем использовались программно, эти описания должны были осуществляться в рамках некоторого формального языка с достаточно четким синтаксисом и семантикой, чтобы программа, осуществляющая «рассуждения» на основе этих знаний, могла их однозначно интерпретировать и использовать для решения конкретной задачи. Этим (а также тем, что они образуют некую единую систему)

знания и отличаются от обычных данных. Стоит, однако, отметить, что четкого определения понятия «знания» нет. Сами представления знаний и выступают в роли наиболее полного определения этого понятия.

Противоречивые требования к удобству манипулирования знаниями и выразительной силе (а также связь области представления знаний с разнообразными науками) привели к возникновению некоторого количества различных языков представления, среди которых наиболее распространены

- логические модели;
- системы продукций;
- семантические сети;
- фреймы;
- объектно-ориентированные представления;
- сценарии и ряд других представлений.

Логические модели были, пожалуй, исторически первыми моделями представления знаний и сохранили свою популярность и по сей день. Логичные рассуждения выглядели эталоном мышления, а манипулирование знаниями в рамках логических моделей как раз и означает логический вывод, который может быть достаточно эффективно реализован. Сначала будут рассмотрены логические модели и достаточно близкие к ним системы продукций, а затем – другие представления знаний.

Для начала вспомним основные элементы логики высказываний и исчисления предикатов. В логике высказываний высказывание является базовым элементом, который может быть либо истинным (1), либо ложным (0). Логически неделимое высказывание (простой факт) называется *пропозициональной переменной*. Из таких высказываний строятся сложные высказывания, или *пропозициональные формулы*, путем их соединения с помощью логических операций.

К основным логическим операциям относят конъюнкцию “$\wedge$” (другие обозначения: “&”, “and”), дизъюнкцию “$\vee$” (другое обозначение: “or”), отрицание “$\neg$” (другие обозначения: “$\sim$”, “$^{-}$”, “not”) и импликацию “$\Rightarrow$” (другие обозначения: “$\rightarrow$”, “$\supset$”, “then”). Таким образом,

- всякая пропозициональная переменная есть пропозициональная формула;
- если A – пропозициональная формула, то $\neg(A)$ – также пропозициональная формула;
- если A и B – пропозициональные формулы, то $(A)\wedge(B)$, $(A)\vee(B)$, $(A)\Rightarrow(B)$ – также пропозициональные формулы.

Каждая логическая операция описывается таблицей истинности, примером которой может служить таблица 1.

Таким же образом можно описать и любую формулу. То есть формула, включающая n пропозициональных переменных – это

отображение $\{0,1\}^n \to \{0,1\}$, которое однозначно определяется таблицей истинности, содержащей 2^n строк.

Таблица 1.

A	B	$A \vee B$	$A \wedge B$	$A \Rightarrow B$
0	0	0	0	1
1	0	1	0	0
0	1	1	0	1
1	1	1	1	1

Некоторые формулы истинны независимо от значений входящих в них переменных. Эти формулы выражают логические законы, также называемые *тавтологиями*. В чем же интерес формул, истинных вне зависимости от значений и смысла их параметров? Эти формулы могут описывать схемы логического мышления.

Например, если мы знаем, что два высказывания эквивалентны: «некоторое число рационально» и «некоторое число представимо в виде дроби целых чисел», и если мы знаем, что число 3,(3) представимо в виде дроби, то мы можем заключить, что число 3,(3) – рационально. Таким образом, зная истинность некоторого правила вида $A \Leftrightarrow B$ и зная факт A, мы может сделать заключение об истинности B.

Рассмотрим более подробно следующую тавтологию: $\big((A \Rightarrow B) \wedge A\big) \Rightarrow B$. Она означает следующее: если из A следует B, и A истинно, то истинно также и B. К примеру, пусть истинна импликация «Некто является человеком (A) $\Rightarrow$ он смертен (B)». И пусть Сократ – человек (то есть A применительно к Сократу истинно), тогда истинно и B, то есть Сократ смертен. Это правило, называемое *modus ponens*, было примером правильных логических рассуждений еще со времен Аристотеля.

Здесь, правда, видно, что при применении логики высказываний к реальным высказываниям приходится идти на некоторую хитрость, связанную с тем, что мы говорили об истинности высказывания A применительно к Сократу, но могли говорить и про любого другого человека, то есть делали неявную подстановку. Если говорить строго, то нужно было бы сказать, что A = «Сократ – человек», B = «Сократ смертен», а тогда наш логический вывод превратился бы в следующее: мы знаем, что если «Сократ – человек», то «Сократ смертен». Мы также знаем, что «Сократ – человек». Следовательно, «Сократ смертен». Подобный вывод не представляет особого интереса в отличие от случая, когда правило $A \Rightarrow B$ означает «Любой человек смертен».

Эта проблема решается в исчислении предикатов. Рассмотрим основные понятия исчисления предикатов первого порядка. Пусть M – некоторое непустое множество. Множество M^k состоит из всех последовательностей (m_1,…, m_k) длины k. Назовем *k-местной функцией*

(или функцией k аргументов) на множестве M любое отображение множества M^k в множество M.

Назовем *k-местным предикатом* на множестве M отображение множества M^k в множество $\{0,1\}$. То есть предикат истинен для одних последовательностей вида (m_1,…, m_k) и ложен для других.

Из переменных, обозначающих элементы множества M, а также из функций формируются *термы*. *Формулы* в исчислении предикатов формируются на основе логических операций и предикатов от термов. Например, если f_1 и f_2 – двуместные функции (точнее, функциональные символы), то $f_1(f_2(m_1,m_2),m_3)$ – терм. Если A, B – двуместные предикаты (точнее, предикатные символы), то $A(f_1(m_1,m_2),m_3) \Rightarrow B(f_2(m_1,m_2),m_3)$ – формула.

Помимо этого в исчислении предикатов используются *кванторы* существования "$\exists$" и всеобщности "$\forall$". Поясним смысл этих кванторов на примере. Пусть M – множество произвольных объектов. И пусть Человек(m), где $m \in M$, – это одноместный предикат, который определяется как истинный, если m – человек, и ложный в противном случае. Пусть Смертен(m) – это тоже одноместный предикат, который может быть ложным для «вечных» объектов, а для других объектов он истинен.

Теперь запишем: Человек(m) $\Rightarrow$ Смертен(m). Это формула с одной свободной переменной m, то есть истинность формулы зависит от конкретного значения m. Если же мы хотим сказать, что данная импликация истинна для любого m, то в записи следует использовать квантор всеобщности: $(\forall m)$ Человек(m) $\Rightarrow$ Смертен(m). Теперь на языке предикатов приведенный вывод на основе правила «*modus ponens*» можно представить более строго: выражение

$$((\forall m)\,\text{Человек}(m) \Rightarrow \text{Смертен}(m)) \wedge \text{Человек}(\text{Сократ}) \Rightarrow \text{Смертен}(\text{Сократ})$$

имеет истинное значение.

Поскольку это не просто логическое выражение, а пример логического рассуждения (то есть получение одного истинного утверждения из других истинных утверждений), правильнее его представлять в следующей форме:

1. Общий факт: $(\forall m)$ Человек(m) $\Rightarrow$ Смертен(m).
2. Частный факт: Человек(Сократ).
3. Вывод: Смертен(Сократ).

На языке исчисления предикатов можно представлять достаточно разнообразные знания. Рассмотрим следующий пример. Пусть M – это множество людей. И пусть заданы следующие предикаты: Родитель(m_1,m_2), который истинен, если m_1 – родитель m_2, и Мужчина(m), который истинен, если m – мужчина, и ложен, если m – женщина. Тогда можно определить истинность для некоторых других предикатов:

1. $(\forall m_1, m_2)$ (Родитель(m_1,m_2) $\wedge$ Мужчина(m_1) $\Leftrightarrow$ Отец(m_1,m_2)).
2. $(\forall m_1, m_2)$ (Родитель(m_1,m_2) $\wedge \neg$ Мужчина(m_1) $\Leftrightarrow$ Мать(m_1,m_2)).

3. $(\forall m_1, m_2)\,((\exists m_3)\,\text{Родитель}(m_1,m_3) \wedge \text{Родитель}(m_3,m_2) \wedge \text{Мужчина}(m_1) \Leftrightarrow \text{Внук}(m_2,m_1))$.
4. Аналогичным образом можно определить и все другие родственные отношения: Брат(m_1,m_2), Тетя(m_1,m_2) и т.д.

Имея подобную систему утверждения, а также зная базовые факты Родитель(m_1,m_2) и Мужчина(m) о конкретных людях, можно логически вывести между ними прочие родственные отношения. Такой вывод обычно осуществляется методом резолюций, идея которого заключается в том, чтобы к имеющемуся набору фактов добавить отрицание утверждения, истинность которого нужно доказать, и попытаться вывести из получившегося расширенного набора фактов очевидно ложное утверждение (то есть прийти к противоречию). Резолюцией же называется тавтология

$$(A \vee C) \wedge (B \vee \neg C) \Rightarrow A \vee B, \qquad (8)$$

на основе которой формируется следующее правило логического вывода: если в базе правил есть два факта вида $A \vee C$ и $B \vee \neg C$, то в базу нужно добавить правило $A \vee B$, которое далее следует использоваться для построения новых резолюций. Это позволяет выводить новые правила в надежде получить противоречие.

Подробно метод резолюций здесь разбирать не будем, а отметим лишь следующий момент. Процесс доказательства утверждения методом резолюций – это поиск на дереве целей, поскольку заранее неизвестно, какие факты нужно объединить с отрицанием доказываемого утверждения, чтобы получить ложное следствие. Как любая задача поиска, поиск доказательства может оказаться нетривиальной задачей, особенно при большом числе исходных фактов. Естественно, здесь часто используются методы эвристического программирования.

Таким образом, исчисление предикатов может служить основой систем представления знаний и манипуляции ими, которые, однако, допускают различные реализации. Можно представить реализацию приведенных в предыдущем примере правил на некотором языке общего назначения, например, Си:

```
1. bool Father(Human m1, Human m2)
     { return Parent(m1,m2) && Male(m1); }
2. bool Mother(Human m1, Human m2)
     { return Parent(m1,m2) && !Male(m1); }
```

Однако уже реализация предиката Внук(m_2,m_1) не столь очевидна. Здесь либо требуется перебирать всех людей в поисках такого m_3, для которого верно Родитель(m_1,m_3) и Родитель(m_3,m_2), либо хранить в переменных, описывающих людей, некоторую дополнительную информацию, например, ссылки на родителей и детей. Иными словами, здесь оказывается необходимо в явном виде описывать процедуру получения конкретного результата.

Такой стиль представления знаний называется *процедурным*, так как конкретные элементы знания представлены в виде процедур в программном коде. А поскольку компьютерная программа, как правило, не имеет возможности «осмысленно» интерпретировать собственный код, при процедурном представлении знаний программа не может инспектировать имеющиеся знания, хотя и может осуществлять логический вывод на их основе. Иными словами, программа «знает, как», но «не знает, что».

Альтернативой процедурному представлению знаний является декларативное представление. Именно такой способ представления знаний использовался в системе GPS. Примером реализации декларативного способа представления знаний в рамках исчисления предикатов первого порядка служит язык программирования Пролог. В этом языке знания описываются подобно тому, как в примере с описанием родственных отношений вводились связи между предикатами.

Декларативное представление знаний обладает следующими особенностями, обуславливающими его преимущество перед процедурным представлением.

- Собственно знания во многом отделены от механизмов их использования (например, от системы логического вывода), что позволяет применять единожды реализованную систему манипулирования знаниями (например, реализацию метода резолюций) для разных баз знаний, описывающих независимые предметные области.
- Декларативные представления, как правило, обладают более простыми синтаксисом и семантикой, чем процедурные представления, и наполнение баз знаний, построенных на их основе, является более доступным для пользователей, не являющихся программистами, а также легче автоматизируется.
- Декларативные представления обычно описывают знания как совокупность однотипных элементов (например, логических формул), то есть воплощают модульный принцип. Это позволяет быстро создавать прототипы систем и достаточно свободно включать и исключать из них элементы без существенной перестройки всей системы. Проблема же проектирования систем, основанной на процедурном представлении, является очень серьезной, а ошибки проектирования могут повлечь необходимость трудоемкого рефакторинга всей системы.

В то же время, декларативные представления являются менее выразительными и не всегда достаточно прозрачными, поскольку их интерпретация может зависеть от конкретной реализации системы манипулирования знаниями.

Помимо декларативного представления знаний в языке Пролог реализован метод резолюций. Естественно, эта реализация не универсальна и сталкивается с рядом трудностей при ее применении в

реальных задачах. Помимо увеличения сложности поиска при увеличении базы фактов существуют и другие трудности, в частности, проблема неполных знаний. В классической логике предполагается, что любое высказывание может быть только истинным или ложным, однако статус некоторых высказываний может быть просто неизвестен. В частности, для некоторого человека в базе знаний может отсутствовать информация о возрасте или росте. Однако механизм логического вывода такой возможности неполной информации не предусматривает. В зависимости от того, как интерпретатор обрабатывает такие ситуации, возможны разные ответы на один и тот же запрос.

Как правило, вводится так называемое *предположение о замкнутости мира* (базы знаний), в рамках которого полагается, что все знания о предметной области имеются. Обычно для воплощения этого предположения любое предположение, статус которого неизвестен, полагают ложным, что позволяет сделать логическую систему знаний полной и непротиворечивой, а вывод над ней – *монотонным*. Это означает, что может быть определена истинность любого утверждения, причем ранее выведенные утверждения остаются истинными вне зависимости от того, какие утверждения будут выведены в дальнейшем. Однако за это приходится платить тем, что подобная база знаний не может расширяться: добавление в нее нового факта, ранее отсутствовавшего в базе и предполагавшегося ложным, меняет истинность ранее выведенных утверждений.

Предположение о ложности неизвестных фактов также является не вполне адекватным. Рассмотрим следующие два утверждения, которые могут использоваться в двух разных базах знаний.

1. $(\forall x)(\text{Птица}(x) \wedge \text{Обычный}(x) \Rightarrow \text{Летает}(x))$
2. $(\forall x)(\text{Птица}(x) \wedge \neg\text{Небычный}(x) \Rightarrow \text{Летает}(x))$

Если для некоторой птицы установлено значение предикатов «Обычный» или «Необычный», то вывод в этих базах знаний будет совпадать. Однако если для какой-то птицы, например, воробья, эта информация не указана, то заключение о том, летает ли эта птица или нет, будет разным. В первой базе будет предполагаться, что Обычный(воробей) является ложным, откуда будет заключено, что воробей не летает. Во второй же базе будет предполагаться, что Необычный(воробей) является ложным, поэтому Летает(воробей) будет истинным. Во второй базе окажется необходимым лишь для некоторых необычных птиц указать истинность соответствующего предиката. В первой базе окажется необходимым перечислять значение Обычный(x) для всех летающих птиц.

Помимо исчисления предикатов существует множество других неклассических логик, в частности, многозначных логик, в которых к значениям истинности и ложности добавлены другие значения (например, степень уверенности). В таких логиках проблемы с отсутствием информации могут иметь решения, однако для них хуже разработаны

механизмы вывода (в частности, ввиду их большого многообразия). Многозначные логики могут быть монотонными: в них в результате вывода для утверждения, опирающегося на отсутствующий в базе факт, устанавливается значение «неизвестно», которое, однако, в дальнейшем остается неизменным. Существуют также и немонотонные логики, которые допускают изменение значений утверждений со временем.

Близкими к логическим представлениям, но несколько отличающимися от них, являются *наборы правил*, или *продукционные системы*. Хотя продукционные системы часто относят к представлениям знаний, под ними все же обычно подразумевают наборы правил, дополненные механизмами вывода (манипулирования знаниями). Наборы правил бывают разными, но их объединяет то, что знания в них представляются в форме продукций вида: $A \Rightarrow B$. Здесь знак "$\Rightarrow$" может означать логическое следование, но чаще он означает связку типа «условие-действие». При этом как условие, так и действие могут задаваться в весьма различной форме.

Рассмотрим простой пример набора правил, позволяющего переводить слово из именительного падежа в дательный. Для начала включим в этот набор одно правило:

1. $X \Rightarrow X$ у.

Это правило гласит, что для перевода в дательный падеж к данному слову нужно приписать букву "у": дом$\Rightarrow$дому; сад$\Rightarrow$саду. Однако это правило не работает для существительных женского рода, заканчивающихся на букву "а". Для них можно ввести новую продукцию:

2. X а$\Rightarrow X$ е.

Это правило говорит нам, что при встрече произвольной цепочки букв X, справа от которой стоит буква "а", эту цепочку нужно оставить неизменной, а букву "а" заменить буквой "е": лиса$\Rightarrow$лисе; нора$\Rightarrow$норе. Однако для слов женского рода, заканчивающихся на мягкий знак, эти два правила будут работать неверно. То есть нужно ввести следующую продукцию.

3. X ь$\Rightarrow X$ и.

Теперь для конкретных значений X получим: боль$\Rightarrow$боли; соль$\Rightarrow$соли. Но и здесь есть исключения, например, слово «рожь». Для этого слова необходимо ввести отдельную продукцию:

4. рожь$\Rightarrow$ржи.

Представим, что продукционной системе подается на вход слово «рожь». Какое из правил с истинной посылкой: первое, третье или четвертое, – к этому слову применять? Чтобы избежать неоднозначности, применяют самое первое подходящее правило из заданного набора. Однако если правило с наиболее общим условием находится выше всего, то правило с более частным условием никогда не будет применено. В связи с этим часто используется следующая эвристика: чем более частное условие у продукции, тем раньше ее нужно помещать в списке. В нашем примере порядок продукций следовало бы изменить на обратный.

Существование на каждом шаге вывода нескольких правил, которые могут быть применены в текущих условиях, является типичным для продукционных систем. Устранение неоднозначности выбора правила носит название *разрешения конфликтов*. Помимо эвристики *специфичности*, то есть применения в первую очередь наиболее частного правила, существуют также и другие эвристики (также называемые стратегиями разрешения конфликтов), например эвристика, согласно которой следует применять то правило, которое в процессе текущего вывода еще не применялось. Если база правил является открытой (незамкнутой), то согласно еще одной эвристике следует использовать наиболее новое правило, то есть правило, которое было либо добавлено как можно позднее, либо условие применимости которого оказалось выполненным только что. Помимо общих эвристик могут также использоваться *метаправила*, которые являются частью базы знаний и описывают (предметно-зависимые) правила разрешения конфликтов между обычными продукциями. При использовании метаправил продукции могут также расширяться *предусловиями*, в которых описываются условия, предпочтительные для применения данных продукций.

В продукционных системах не обязательно на каждом шаге применяется только одно правило. Здесь также возможен и поиск: при формировании дерева вариантов из каждого узла исходит такое количество ветвей, сколько разных правил может быть применено в данной ситуации. Тогда стратегии разрешения конфликтов превращаются в классические эвристики поиска: они указывают предпочтительный порядок анализа подветвей дерева вариантов. Однако построение дерева вариантов возможно не во всех задачах.

Наиболее типичным является использование продукционных систем в задачах диагностики и классификации, причем первые не допускают построения полного дерева вариантов, а требуют «поиска» в реальном пространстве. К примеру, система помощи поиска неисправности автомобиля, может содержать такие правила:

1. Автомобиль не заводится и есть бензин $\Rightarrow$ проверить зажигание.
2. Автомобиль не заводится $\Rightarrow$ проверить наличие бензина.
3. …

Подобные системы часто начинают работу при минимуме информации, поэтому применяются общие правила. Стоящие в правых частях этих правил действия зачастую являются физическими действиями, направленными на изменение состояния диагностируемого объекта или на получение дополнительной информации, которая позволит выбрать из большой совокупности более частных правил. Здесь еще раз можно отметить, что поиск решения в физическом мире по структуре своей организации не отличается от «рассуждений» (манипуляции знаниями, представленных в символьном виде). В очень примитивной форме диагностирующие продукционные системы достаточно распространены в

виде компьютерных помощников, предназначенных (по крайней мере, по замыслу разработчиков) для помощи пользователям в настройке программного обеспечения. Пожалуй, наиболее серьезное применение продукционные системы имеют в диагностике заболеваний.

Выполнение классификации с использованием наборов правил сходно с выполнение диагностики, за тем исключением, что в этом случае продукционная система обычно не получает новой информации в процессе работы. Здесь правила принимают такой вид:

1. Зеленый, полосатый, вкусный $\Rightarrow$ арбуз.
2. Желтый, кислый $\Rightarrow$ лимон.
3. …

Иными словами, в этом случае наборы правил служат для описания понятий из некоторой предметной области через совокупность дискретных признаков. Условия в левых частях правил могут принимать и более сложный вид, приближая эти правила к формулам исчисления предикатов. В частности, широко используются операции конъюнкции и дизъюнкции. Например, первое правило в предыдущем примере можно представить в виде: $(\forall x)(\text{Зеленый}(x) \wedge \text{Полосатый}(x) \wedge \text{Вкусный}(x) \Rightarrow \text{Арбуз}(x))$. Однако произвольные предикаты в продукционных системах не реализуются, в частности, обычно не используется квантор существования.

Вопросы и упражнения

1. Перечислить основные типы представления знаний (не менее пяти).
2. В чем основные различия между знаниями и данными?
3. Какое основное преимущество имеет исчисление предикатов в качестве системы представления знаний?
4. Чем логика предикатов отличается от логики высказываний?
5. К какой проблеме искусственного интеллекта сводится проблема доказательства в рамках исчисления предикатов?
6. Какие задачи могут решаться с помощью наборов правил?

8. Формальные грамматики и семантические сети

В логических представлениях основное внимание уделяется не столько удобству и выразительной силе языков представления знаний, сколько обеспечению эффективного манипулирования этими знаниями, при котором сохраняется правильность рассуждений благодаря совершенству и полноте правил дедуктивного вывода. Однако из-за подобной идеализированности возникает множество проблем при отображении рассуждений о реальном мире в формальную логику.

Рассмотрим следующий пример. Пусть имеется гипотеза, выраженная в виде логической формулы: $(\forall m)\,\text{ворон}(m) \Rightarrow \text{черный}(m)$. Это утверждение логически эквивалентно следующему утверждению:

$(\forall m) \neg$черный$(m) \Rightarrow \neg$ворон(m). Однако представим, что какие-то два ученых хотят эту гипотезу проверить, причем один из них руководствуется первой логической формулой, а второй – эквивалентной ей второй формулой. Первый будет искать воронов, чтобы убедиться, что они черные. Второй же просто обнаружит вокруг себя множество нечерных предметов, ни один из которых не является вороном, и будет считать это подтверждением своей гипотезы. С точки зрения здравого смысла, действия второго ученого будут выглядеть нелепыми.

Причина такого несоответствия заключается в том, что логическая импликация выражает лишь связь между значениями истинности своих операндов, но она не устанавливает причинно-следственную связь или связь типа класс-экземпляр. В то же время, в естественном языке связка «если…, то…» подразумевает именно такого рода отношение. Формальная логика дает примеры правильных рассуждений вне зависимости от их смысла. В этом заключается ее сила, но в то же время в этом и ее слабость, поскольку, как оказывается, смысл естественноязыковых (ЕЯ) высказываний играет большую роль в рассуждениях.

Проблема смысла ЕЯ высказываний являлась центральной еще для одного направления в области представления знаний, в котором большую роль сыграли исследования психологов и лингвистов. Для понимания смысла высказываний на естественном языке необходимо обладать как лингвистическими знаниями, так и знаниями о мире. Первый тип знаний относится к грамматике естественного языка, а второй – к его семантике. Хотя грамматика и семантика принадлежат одному и тому же языку, их исследование осуществляется сравнительно независимо, и для каждой из них предлагаются собственные средства описания, наиболее известными из которых являются формальные грамматики и семантические сети.

Рассмотрим кратко вопрос представления лингвистических знаний на примере формальных грамматик. Теорию формальных грамматик, которая стала одним из основных разделов математической лингвистики, в середине прошлого века разработал Ноам Хомский, именно исходя из потребностей лингвистики. Однако этим сфера применения данной теории не ограничивается. Формальные грамматики оказались удобным средством при создании компиляторов и трансляторов для языков программирования, для классификации динамических систем и при исследовании классов алгоритмов, а также в структурном распознавании образов.

Введем основные определения.

Под *алфавитом* будем подразумевать некоторое конечное множество, элементы которого будем называть *символами*.

Предложением в данном алфавите будем называть произвольную (конечную) цепочку символов этого алфавита.

Языком над данным алфавитом будем называть произвольное (возможно, бесконечное) множество предложений в этом алфавите.

Порождающей (формальной) грамматикой будем называть четверку $G = (V_T, V_N, P, S)$, где

V_T – алфавит *терминальных* (основных) *символов*;

V_N – алфавит *нетерминальных* (вспомогательных) *символов*, причем $V_N \cap V_T = \varnothing$;

$V_N \cup V_T = V$ – алфавит грамматики G;

P – множество *правил подстановки* (или продукций);

S – *начальный* (корневой) *символ*, $S \in V_N$.

Через V^* будем обозначать все возможные цепочки символов (предложения) алфавита V. Через V^+ обозначим множество $V^* \setminus \{\Lambda\}$, где Λ – пустая цепочка. Тогда правило подстановки (элемент множества P) будет иметь вид: $\alpha \to \beta$, где $\alpha \in V^+, \beta \in V^*$, причем хотя бы один символ предложения α должен быть нетерминальным. Это правило говорит о том, что в любом предложении вхождение цепочки символов α *может быть* заменено цепочкой β: $(\forall \gamma, \delta \in V^*)\gamma\alpha\delta \Rightarrow \gamma\beta\delta$. Символ $\Rightarrow$, в отличие от символа $\to$, который применяется при записи правил вывода, используется для обозначения возможности вывода одного предложения из другого в результате применения некоторого количества (одного или нескольких) правил грамматики. Вывод в грамматике начинается с корневого символа S и заключается в последовательном применении правил подстановки.

Предложение называется *терминальным*, если оно состоит только из терминальных символов. Поскольку в любом правиле вывода слева стоит цепочка, содержащая, по крайней мере, один нетерминальный символ, то к терминальному предложению не могут быть применены никакие правила вывода. Могут существовать и нетерминальные предложения, вывод из которых не может быть продолжен.

Через $\Gamma(G)$ обозначим язык, порождаемый грамматикой G и содержащий все терминальные предложения (и только их), выводимые из начального символа S, то есть

$$\Gamma(G) = \left\{\alpha \middle| (\alpha \in V_T^*) \& (S \Rightarrow \alpha)\right\}. \qquad (9)$$

Рассмотрим пример. Пусть даны алфавиты $V_T = \{a, b\}$, $V_N = \{S, A, B\}$ и правила подстановки $P = \{S \to AS, S \to SB, S \to \Lambda, A \to ab, B \to ba\}$. В грамматике с такими компонентами могут быть выведены любые цепочки вида $(ab)^n (ba)^m$. Приведем одну из бесконечного множества возможных последовательностей подстановок:

$$S \Rightarrow AS \Rightarrow AAS \Rightarrow AASB \Rightarrow AAB \Rightarrow abAB \Rightarrow ababB \Rightarrow ababba.$$

На основе этого простого примера можно сделать ряд наблюдений.

- На каждом шаге вывода может существовать выбор из нескольких возможностей применения грамматических правил. Таким образом,

грамматика не задает (детерминированный) алгоритм порождения некоторой цепочки символов, в отличие, например, от машины Тьюринга, последовательность операций которой предопределена содержимым входной ленты. Грамматика не указывает последовательность действий по порождению цепочек символов, а устанавливает ограничения на возможные действия (и, как следствие, на порождаемые цепочки). В то же время, каждая порождаемая цепочка описывается в грамматике последовательностью правил постановки, соответствующей генеративной модели набора символьных данных. В этом и заключается смысл грамматики, как символьного представления, задающего пространство структурированных объектов.

- Конечный набор правил может порождать язык, содержащий бесконечное количество предложений.
- Различные грамматики могут порождать одинаковые языки (такие грамматики называются *слабо эквивалентными*). Например, грамматика, состоящая из элементов $V_T = \{a, b\}$, $V_N = \{S\}$, $P = \{S \to abS, S \to Sba, S \to \Lambda\}$, порождает тот же язык, что и грамматика, рассмотренная выше.

В приведенном выше примере нетерминальные символы использовались для обозначения определенных цепочек терминальных символов. Нетерминальные символы могут использоваться также для обозначения некоторого подмножества терминальных символов. Например, если даны алфавит терминальных символов $V_T = \{a, b, c, d\}$ и алфавит нетерминальных символов $V_N = \{S, A, B\}$, и существуют правила подстановки $A \to a, A \to b$ и $B \to c, B \to d$, то нетерминальные символы A и B будут обозначать соответствующие классы терминальных символов $\{a, b\}$ и $\{c, d\}$. В более сложных случаях одни нетерминальные символы могут обозначать классы других нетерминальных символов или их цепочек, а также смешанные классы и цепочки, что приведет к сложным структурам предложений.

Итак, терминальные символы представляют собой непроизводные элементы, из которых формируются языковые объекты, а нетерминальные символы являются различного рода обобщениями этих непроизводных элементов, описывая их классы или сконструированные из них цепочки.

Рассмотрим пример простой грамматики, описывающей небольшую часть естественного языка. Пусть алфавит V_T содержит следующие терминальные символы (слова): *быстрый, быстрые, медленный, медленные, большой, большие, компьютер, компьютеры, калькулятор, калькуляторы, число, числа, величина, величины, умножает, умножают, складывает, складывают*. И пусть алфавит V_N содержит следующие нетерминальные символы: S – понятие правильно построенного предложения, C_1 – существительное единственного числа, C_2 –

существительное множественного числа, $П_1$ – прилагательное единственного числа, $П_2$ – прилагательное множественного числа, $Г_1$ – глагол единственного числа, $Г_2$ – глагол множественного числа, C'_1, C'_2 – группы существительного, $Г'_1, Г'_2$ – группы глагола. И пусть есть следующий набор правил:

$S \to C'_1 Г'_1$, $S \to C'_2 Г'_2$,

$C'_1 \to П_1 C_1$, $C'_1 \to C_1$, $C'_2 \to П_2 C_2$, $C'_2 \to C_2$,

$Г'_1 \to Г_1 C'_1$, $Г'_1 \to Г_1 C'_2$, $Г'_1 \to Г_1$, $Г'_2 \to Г_2 C'_1$, $Г'_2 \to Г_2 C'_2$, $Г'_2 \to Г_2$,

$C_1 \to$ *компьютер*, $C_1 \to$ *калькулятор*, $C_1 \to$ *число*, $C_1 \to$ *величина*,

$C_2 \to$ *компьютеры*, $C_2 \to$ *калькуляторы*, $C_2 \to$ *числа*, $C_2 \to$ *величины*,

$Г_1 \to$ *умножает*, $Г_1 \to$ *складывает*,

$Г_2 \to$ *умножают*, $Г_2 \to$ *складывают*,

$П_1 \to$ *быстрый*, $П_1 \to$ *медленный*, $П_1 \to$ *большой*,

$П_2 \to$ *быстрые*, $П_2 \to$ *медленные*, $П_2 \to$ *большие*.

Такая грамматика порождает некоторые предложения русского языка, имеющие сходную синтаксическую структуру. Пример допустимого предложения представлен на рис. 15 с соответствующим *структурным деревом*. Нетерминальные символы здесь обозначают части речи и синтаксические типы. В равной степени нетерминальные символы могут обозначать общие понятия, устойчивые словосочетания, парадигмы слов (как обобщения словоформ) и т.д. Это показывает, каким именно образом формальная грамматика может использоваться для описания структуры естественного языка.

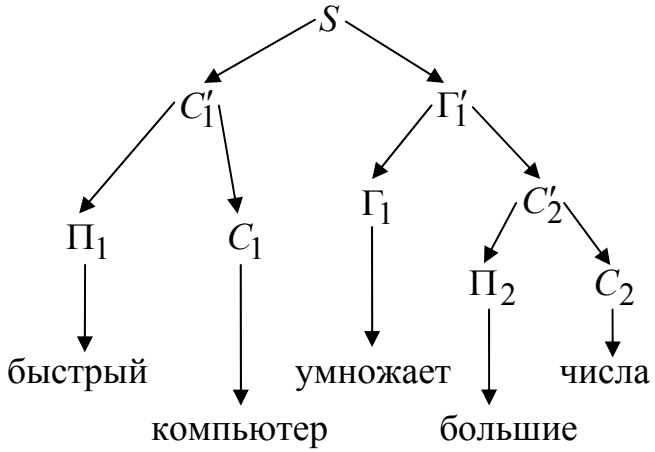


Рис. 15. Структурное дерево фразы «быстрый компьютер умножает большие числа», порожденной с помощью некоторой простой формальной грамматики

Хотя приведенная в примере грамматика порождает только правильно построенные предложения, не все из них оказываются вполне осмысленными. В частности, допустимым является такое предложение: «Медленные величины складывают большой калькулятор». Иными словами, грамматические правила не описывают семантику языка.

Некоторая грамматика G называется *однозначной*, если для каждого слова из порождаемого ею языка $\alpha \in \Gamma(G)$ верно, что все возможные варианты его вывода имеют идентичные структурные деревья. Вместо сравнения структурных деревьев двух выводов можно воспользоваться следующим приемом: заменить правила $\alpha \to \beta$ правилами $\alpha \to (\beta)$, где скобки играют роль служебных символов и не учитываются при выводе. Если в результате двух выводов формируются цепочки символов, совпадающие вплоть до расстановки скобок, то такие выводы не считаются существенно различными. Грамматика называется неоднозначной, если хотя бы для одной цепочки $\alpha \in \Gamma(G)$ существуют существенно различные выводы.

Часто подчеркивается неоднозначность естественных языков: в них существуют предложения, имеющие несколько альтернативных синтаксических структур при одинаковом написании. Классическим примером неоднозначного предложения для английского языка является предложение «They are flying planes» (которое может иметь переводы «Это летящие самолеты» или «Они летят на самолетах»). Несмотря на такую неоднозначность, обычно ограничиваются рассмотрением однозначных грамматик, так как они существенно проще в использовании.

Выражения естественного языка характеризуются не только неоднозначностью. В значительно большей степени им присуща избыточность: одна и та же мысль может быть выражена многими способами. Например, для высказывания «Лишь большое количество специальных терминов в данном тексте не позволит Смиту перевести его» существует несколько миллионов предложений, соответствующих ему по смыслу. Из-за избыточности и неоднозначности естественных языков проблема семантики оказывается очень трудной. Хотя в рамках формальных грамматик предпринимались попытки разрешить эту проблему, но все же для этих целей себя больше зарекомендовали другие средства.

Грамматики с правилами подстановки произвольного вида оказываются неудобными в применении на практике. Вместо них используются грамматики некоторых специальных типов. В зависимости от формы правил постановки принято выделять четыре типа грамматик, предложенных Н. Хомским (в последствии были выделены многие промежуточные подтипы). Эти типы грамматик вводятся путем наложения все более сильных ограничений на вид правил подстановки. С точки зрения использования формальных грамматик в целях представления знаний, существование различных типов грамматик является очень удобным.

Неограниченные грамматики (грамматики типа 0). Эти грамматики характеризуются правилами подстановки вида $\alpha \to \beta$, где на цепочки $\alpha \in V^+, \beta \in V^*$ не накладывается никаких дополнительных ограничений.

Грамматики непосредственно составляющих (НС-грамматики, контекстно-зависимые грамматики, грамматики типа 1). Эти грамматики состоят из правил вида $\gamma A\delta \to \gamma\beta\delta$, где $A \in V_N; \gamma,\delta \in V^*; \beta \in V^+$. Наложение ограничений на вид правил делает языки, порождаемые НС-грамматиками, более узкими, чем языки, порождаемые грамматиками типа 0. Условие $\beta \in V^+$ (или $\beta \neq \Lambda$) делает НС-грамматики *неукорачивающими*: в них в процессе вывода применение любого правила приводит к тому, что цепочка увеличивает или, по крайней мере, не уменьшает свою длину.

Контекстно-свободные грамматики (КС-грамматики, бесконтекстные грамматики, грамматики типа 2). Эти грамматики состоят из продукций вида $A \to \beta$, где $A \in V_N; \beta \in V^+$. Поскольку в НС-грамматиках возможно выполнение условия $\gamma = \delta = \Lambda$, очевидно, КС-грамматики являются частным случаем НС-грамматик, в которых при замене нетерминального символа выбор подставляемой цепочки может производиться в зависимости от контекста.

Регулярные грамматики (автоматные грамматики, грамматики типа 3). Эти грамматики содержат правила вида $A \to aB$ и $A \to b$, где $A,B \in V_N; a,b \in V_T$. То есть в правилах подстановки фигурируют лишь отдельные символы, а не их цепочки. В связи с этим количество различных правил в грамматиках этого типа ограничивается размером алфавитов. В грамматиках типов 0-2 такого ограничения нет: при фиксированных алфавитах количество правил может быть любым.

Отметим, что в грамматиках типов 1-3 на каждом шаге подстановки заменяется лишь один нетерминальный символ. Это позволяет представлять структуру предложения в виде дерева, подобного тому, которое представлено на рис. 15. Некорневые узлы и листья этого дерева называют *составляющими*, а узлы, являющиеся непосредственными потомками некоторого узла, называются его *непосредственными составляющими*. Отсюда и происходит название НС-грамматик. Например, на рис. 15 C_1' и Γ_1' (группа существительного и группа глагола) являются непосредственными составляющими S (предложения).

В таком представлении предложения в форме структурного дерева не фиксируется последовательность правил вывода, за исключением того, что узлы должны рассматриваться позже своих родителей. В случае наличия правил, при применении которых производится замена нескольких символов, структурное дерево невозможно. В то же время, в неукорачивающей грамматике правило вывода, содержащее замену нескольких символов, может быть преобразовано в некоторое количество правил, в каждом из которых заменяется лишь один символ. В частности, правило вида $AB \to BA$ может быть реализовано на основе правил НС-грамматик вида $\gamma A\delta \to \gamma\beta\delta$ с добавлением четырех новых нетерминальных

символов A_1, A_2, B_1, B_2 и с введением правил подстановки: $AB \to AB_1$, $AB_1 \to A_1B_1$, $A_1B_1 \to BB_1$, $BB_1 \to BA$.

Таким образом, введение ограничений на правила вывода не обязательно приводит к сужению класса порождаемых языков, но с большей вероятностью делает описание языка более громоздкими. Для теоретического анализа свойств грамматик обычно удобнее сделать правила максимально простыми и однотипными, если это не приводит к сужению класса порождаемых языков.

Классы языков, порождаемых грамматиками типов 0-3, не совпадают. Наиболее широкими, естественно, являются языки класса 0. Для любого языка класса 0 существует допускающая его машина Тьюринга (язык допускается некоторой машиной Тьюринга, если она останавливается за конечное число шагов, получив на вход любую из цепочек этого языка). И наоборот, если язык допускается некоторой машиной Тьюринга, то он является языком типа 0. Поскольку проблема останова произвольной машины Тьюринга неразрешима, многие проблемы, связанные с неограниченными грамматиками, также являются алгоритмически неразрешимыми. В частности, неразрешима проблема определения того, порождается ли некоторая цепочка некоторой неограниченной грамматикой в произвольном случае.

Языки типа 3 совпадают с множеством языков, допустимых автоматами гораздо более частного, чем машина Тьюринга, вида – конечными автоматами. Для автоматных грамматик многие проблемы имеют весьма простые решения, но сами грамматики обладают чрезмерно меньшей выразительной силой. Некий компромисс предоставляют НС- и КС-грамматики, которым и было посвящено большинство исследований. Для любой грамматики типа 1 или 2, как и для грамматики типа 3, может быть построен алгоритм, позволяющий для любой цепочки определить, порождается ли она данной грамматикой. Это принципиально отличает данные грамматики от неограниченных грамматик. Наибольший интерес представляют КС-грамматики, поскольку они достаточно выразительны для того, чтобы с их помощью исследовать языки программирования и, в определенной степени, – естественные языки; при этом для КС-грамматик многие проблемы имеют более простые решения, чем для НС-грамматик.

Существуют различные типы грамматик, состоящие из правил вывода специфического вида, но совпадающие с одним из четырех упомянутых выше типов по множеству порождаемых языков. Были предложены и типы грамматик, порождающих промежуточные классы языков. Например, линейные грамматики, являющиеся такими КС-грамматиками, у которых все правила вывода в правой части имеют не более чем один нетерминальный символ, порождают множество языков, более узкое, чем множество, порождаемое всеми КС-грамматиками, но более широкое, чем множество, порождаемое только автоматными грамматиками. Выделено большое количество (несколько десятков) типов грамматик.

Несколько в стороне находятся стохастические грамматики, которые, однако, весьма полезны в задачах машинного обучения.

Стохастические грамматики отличаются от обычных формальных грамматик лишь в том, что правила подстановки в них имеют вид $\alpha \xrightarrow{P(\beta|\alpha)} \beta$, где $P(\beta \mid \alpha)$ – вероятность замены цепочки α на цепочку β (а не на другую цепочку). Имеются следующие ограничения на вероятности: $0 < P(\beta \mid \alpha) \le 1$ (равенство вероятности нулю равносильно отсутствию соответствующего правила подстановки в грамматике) и $\sum_{\beta} P(\beta \mid \alpha) = 1$. В зависимости от формы правил, стохастические грамматики также могут быть типов 0-3, и различия между типами обычных формальных грамматик сохраняются и для стохастических грамматик.

Полагая применение правил независимым, можно определить вероятность осуществления некоторого процесса вывода через произведение вероятностей использованных правил. Поскольку используются условные вероятности $P(\beta \mid \alpha)$, полагается, что на каждом шаге вывода выбор осуществляется только между правилами, в левой части которых стоит цепочка α. В то же время, может быть применено некоторое правило и к какой-то другой подцепочке текущей цепочки. Таким образом, вероятность вывода как произведение вероятностей его отдельных правил имеет смысл безусловной вероятности, только если вывод не зависит от последовательности применения правил.

Вероятность некоторого вывода (вернее, структурного дерева) может быть принята за вероятность цепочки, являющейся результатом этого вывода. Это будет верно тогда, когда грамматика является однозначной, то есть данной цепочке может соответствовать только одно структурное дерево. В противном случае определение вероятностей предложений порождаемого грамматикой языка $\alpha \in \Gamma(G)$ оказывается гораздо более проблематичным. Это одна из причин, по которой часто ограничиваются рассмотрением однозначных грамматик.

Стохастические грамматики задают представления цепочек символов с распределением вероятностей по цепочкам, что отличает такие грамматики от обычных порождающих грамматик, задающих лишь представления с жесткими ограничениями на множества описываемых в их рамках цепочек. Посмотрим на конкретных примерах, как стохастические грамматики задают распределение априорных вероятностей.

Пример. Пусть дана грамматика G: $V_T = \{a, b\}$, $V_N = \{S\}$, $P = \{S \to 0S, S \to 1S, S \to \Lambda\}$. Эта грамматика порождает все битовые строки произвольной длины. Зададим распределение вероятностей для цепочек языка $\Gamma(G)$, определив вероятности для правил вывода.

1. $P(0\,|\,S) = P(1\,|\,S) = P(\Lambda\,|\,S) = 1/3$. Все цепочки длины n являются равновероятными и порождаются с вероятностью $3^{-(n+1)}$, так как вывод цепочки длины n осуществляется посредством $n+1$ правил подстановки.

2. $P(0\,|\,S) = P(1\,|\,S) = 0.49, P(\Lambda\,|\,S) = 0.02$. При таких вероятностях для правил вывода вероятность для каждой цепочки длины n будет равен $0.49^n \cdot 0.02$. Вероятность также зависит только от длины цепочки, но не от ее содержания, однако эта зависимость отличается от зависимости в предыдущем примере.

3. $P(0\,|\,S) = 0.125, P(1\,|\,S) = 0.375, P(\Lambda\,|\,S) = 0.5$. Несложно убедиться, что в такой стохастической грамматике наиболее вероятными будут цепочки, содержащие три четверти единиц и одну четверть нулей. Таким образом, стохастические грамматики позволяют устанавливать вероятности предложений языка в зависимости от входящих в эти предложения символов.

4. Рассмотрим другую стохастическую грамматику G: $V_T = \{a, b\}$, $V_N = \{A, S\}$, $P = \{S \to AS, S \to \Lambda, A \to 011111, A \to 0, A \to 1\}$, где вероятности $P(AS\,|\,S) = P(\Lambda\,|\,S) = 0.5$, $P(011111\,|\,A) = P(0\,|\,A) = P(1\,|\,A) = 1/3$. Этой грамматикой могут порождаться любые цепочки, но с большей вероятностью будут порождаться предложения, содержащие три четверти единиц и одну четверть нулей. То есть по содержанию символов в предложениях эта грамматика не отличается от рассмотренной в предыдущем примере. Однако она будет с большей вероятностью порождать цепочки, содержащие шаблон “011111”.

Установление вероятностей для правил вывода позволяет описывать грамматическую структуру языков более тонко, однако так же, как и обычные порождающие грамматики, не описывает семантику.

Проблема описания семантики языка является одним из уточнений проблемы смысла или проблемы понимания текстов на естественном языке. Проблема смысла является одной из центральных проблем когнитивных наук. Она исследуется в философии, психологии, лингвистике. И хотя существует множество попыток решения этой проблемы, ни одна из них не выглядит вполне удовлетворительной.

В области ИИ исследование проблемы смысла ЕЯ высказываний тесно связано с системами представления знаний. Мы здесь рассмотрим лишь один важный тип представления знаний, который возник в результате попытки решения данной проблемы.

Очень давно было замечено, что человек имеет склонность к ассоциированию. Это послужило отправной точкой для создания ассоционистских теорий, в которых смысл некоторого понятия определяется через его ассоциативные связи с другими понятиями, которые в совокупности образуют своего рода сеть. Понятия являются основой нашего знания о мире, поэтому такую ассоциативную сеть можно рассматривать в качестве представления знаний. Ассоциативные связи

возникают на основе опыта и выражают эмпирические отношения между признаками или поведением объектов.

Например, на основании опыта мы ассоциируем понятие «снег» с другими понятиями, такими как «зима», «холод», «белый», «лед» и т.д. Наши знания о снеге и истинность утверждений типа «снег белый» выражаются в виде сети ассоциаций. Если такая сеть реально используется человеком, можно ли узнать, какова она?

Для ответа на этот вопрос психологами проводился следующий эксперимент. Людям задавали вопросы из некоторой области, например, об особенностях и поведении птиц. Эти вопросы были очень простые: «Может ли голубь летать?», «Может ли канарейка петь?», «Является ли ворона птицей?».

При этом проверялась, конечно же, не правильность ответов на вопросы. Вместо этого оценивалось время, требуемое для ответа. Все вопросы были составлены так, что в них фигурировало два понятия. Наличие непосредственной ассоциации между этими понятиями должно было сказываться на времени ответа.

Как оказалось, времена ответов действительно различались. Так, например, для ответа на вопрос «Может ли канарейка летать?» требовалось больше времени, чем для ответа на вопрос: «Может ли канарейка петь?».

Коллинс и Квиллиан, ставившие этот эксперимент, объясняли большее время ответа на некоторый вопрос тем, что участвующие в вопросе понятия, хотя и расположены достаточно близко в ассоциативной сети, не являются непосредственно связанными. При этом оказывалось, что свойства объектов запоминаются людьми на наиболее абстрактном уровне.

Вместо того чтобы запоминать каждое свойство для каждой птицы (канарейки летают, вороны летают, голуби летают), люди хранят информацию о том, что канарейки – птицы, а птицы, как правило, способны летать. Поскольку поют не все птицы, способность к пению – более частное свойство, которое запоминается на менее абстрактном уровне, чем способность летать, поэтому и ответ на вопрос «Может ли канарейка петь?» требует меньше времени, так же, как и ответ на вопрос «Является ли канарейка желтой?». Таким образом, быстрее всего вспоминаются самые конкретные свойства объектов. Оказалось, что и исключения из правил также запоминаются на самом нижнем, детальном уровне. Примером может служить вопрос «Может ли страус летать?», который для ответа требует меньше времени, чем тот же вопрос о канарейке или другой «нормальной» птице.

На рис. 16 представлен пример фрагмента ассоциативной сети, которая могла бы быть построена в результате такого эксперимента. Конечно, для корректной постановки эксперимента нужно учесть много тонкостей: как получить достоверные измерения времени ответа (можно ли усреднять по большому числу людей, то есть обладают ли разные люди

сходными ассоциативными сетями; можно ли усреднять время ответа одного человека на один и тот же вопрос, то есть нет ли здесь эффекта привыкания), какие именно понятия включить в рассмотрение и т.д. Все эти неоднозначности могут сильно сказываться на достоверности результатов построения ассоциативной сети. Однако главное, что демонстрирует этот эксперимент, – это сам факт того, что понятие ассоциативной сети имеет право на существование, и это понятие отражает некоторые аспекты хранения знаний в памяти человека.

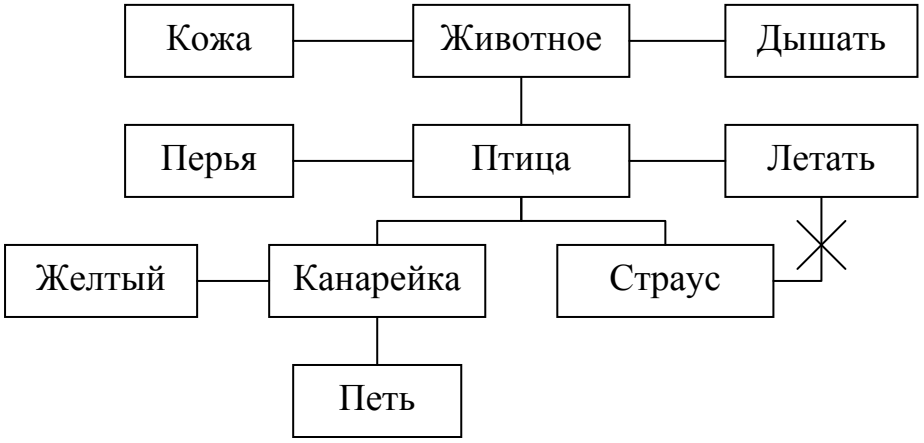


Рис. 16. Пример фрагмента ассоциативной сети

Однако простая ассоциативная связь оказалась недостаточно выразительной для фиксации всевозможных отношений, что видно даже на рассмотренном простом примере. Как оказалось, полезно было бы маркировать не только узлы в ассоциативной сети, но и дуги, поскольку отношения между узлами могут быть самыми разнообразными. Как результат, возникло представление знаний, названное семантическими сетями.

Семантическая сеть представляет собой (ориентированный) граф, узлы которого соответствуют понятиям, а дуги – отношениям между ними. При этом, как узлы, так и дуги имеют метки, указывающие, какое именно понятие помещено в узле, или какое именно отношение обозначает дуга (см. рис. 17).

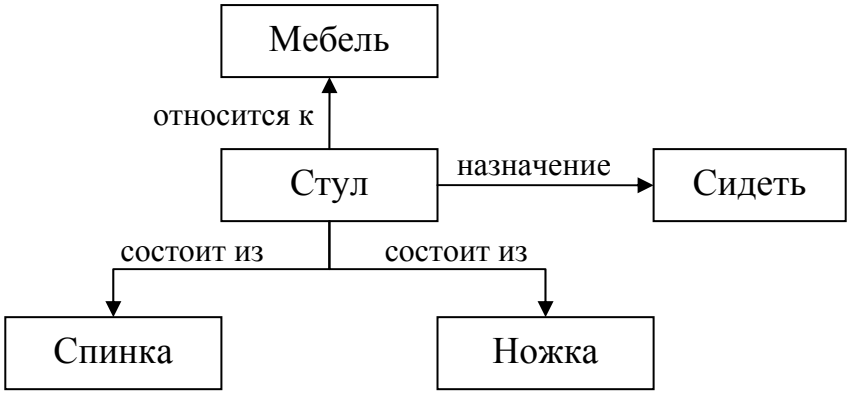


Рис. 17. Пример фрагмента семантической сети

Семантические сети могут использоваться не только для описания общих знаний о мире (вернее, некоторой предметной области), но и для описания структуры единичных высказываний. Такая структура также называется *падежным фреймом* (см. рис. 18). Эта структура имеет то преимущество, что она более однозначна, чем предложение на естественном языке. Одинаковая структура будет характерна и для предложений: «Молотком Егор гвоздь забьет» или «Гвоздь будет забит Егором с помощью молотка». Падежные фреймы не зависят от грамматических падежей конкретных естественных языков, а отражают глубинные (ролевые) взаимосвязи между элементами некоторой ситуации. Падежные фреймы также гораздо проще поддаются автоматическому анализу и, в частности, переводу на другой язык. Они имеют тесную связь с глубинными грамматиками, исследуемыми когнитивными лингвистами, и с трансформационными грамматиками Н. Хомского, являющимися разновидностью формальных грамматик.

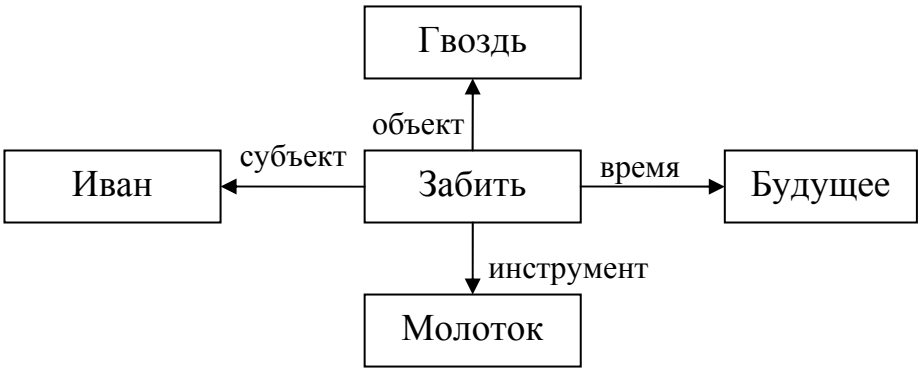


Рис. 18. Семантическая сеть, описывающая падежный фрейм предложения «Иван забьет гвоздь молотком»

Развитие семантических сетей шло путем типизации сетевых отношений. Такого рода типизация была необходима для введения примитивов – таких меток над дугами, для понимания которых интерпретатор программируется заранее, и которые не описываются в самой семантической сети. Мощность представления в виде семантических сетей и определяется типизированными отношениями, поскольку они необходимы, чтобы осуществлять манипуляцию знаниями. Примером общеупотребительного отношения является отношение «относится к», указывающее, что какое-то понятие выступает в качестве конкретизации другого понятия, например, «канарейка относится к птицам», «стул относится к мебели». Этот тип отношений как раз и позволяет распространять свойства более абстрактных понятий на более конкретные понятия (канарейка умеет летать, поскольку канарейка относится к птицам, а птицы умеют летать). Другие типизированные отношения – это, например, «состоит из», «умеет», «обладает свойством» и другие. Для описания падежных фреймов используется собственный

набор типизированных (падежных) отношений, учитывающих грамматику языка.

Вопросы и упражнения

1. Какое отличие между терминальными и нетерминальными символами в формальных грамматиках?
2. Какой тип формальных грамматик является наиболее широким?
3. Какие типы формальных грамматик допускают представление предложений в виде структурного дерева?
4. Для каких типов формальных грамматик является алгоритмически разрешимой проблема синтаксического разбора (определения того, что некоторое предложение может порождаться в рамках данной грамматики)?
5. Какое отличие семантических и ассоциативных сетей?
6. Перечислить основные типизированные отношения в семантических сетях.

9. Фреймы и объектно-ориентированный подход в представлении знаний

Исчисления предикатов и семантические сети являются мощными средствами для представления знаний, однако они плохо описывают сложно структурированные объекты или явления. Так, набор правил – это линейный список не связанных явным образом между собой записей. Логические системы также оперируют формулами, не объединенными в какую-либо структуру, а входящие в эти формулы высказывания, предикаты или объекты просто представляются в виде атомарных символов. В семантических сетях положение узлов не фиксировано в пространстве, то есть при изоморфном отображении (с сохранением всех связей) мы получим совершенно другую по расположению элементов сеть, но представляющую те же знания, поскольку узлы и дуги в исходной и преображенной сети одинаковы.

При этом все узлы в сетевом представлении формально находятся на одном уровне детализации или абстракции (хотя отношение «относится к» подразумевает некоторую иерархию, в явном виде эта иерархия не фиксируется). Это означает, что расположение узлов никак не упорядочено на основе таких отношений, как, например, объект-класс. То есть, к примеру, понятия «животные» и «канарейка» располагаются в общей совокупности узлов. При этом, взяв некий узел, мы не может сказать, насколько абстрактное понятие он представляет (для этого нам придется тщательно исследоваться все его связи).

То же самое относится и к другим «упорядоченным» отношениям, например, объект-часть. Можно было бы представить себе трехмерную семантическую сеть, в которой в зависимости от высоты менялась бы

степень абстрактности понятий, либо же такую сеть, в которой узлы объединялись бы в группы иерархическим образом. Например, популярное сейчас представление знаний в виде *онтологий* зачастую является иерархической семантической сетью с определенным набором отношений. Однако в оригинальном сетевом представлении положение узлов не фиксируется и не используется при манипулировании со знаниями. Подобному расположению мешают также ситуации, представленные на рис. 19. Устранение таких ситуаций всецело накладывается на разработчика, но в самом сетевом представлении отсутствует какой-либо внутренний контроль его согласованности.

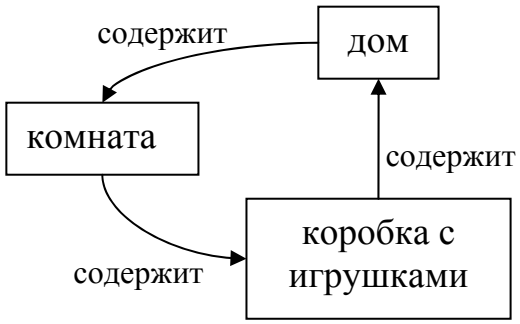


Рис. 19. Пример не вполне корректно составленной семантической сети, возможная ошибочность которой не обнаруживается автоматически в рамках самого представления

Особенно неэффективно с помощью логических и сетевых представлений описывать знания о пространственном расположении и структуре объектов. Предикат «слева» или аналогичным образом помеченная дуга в семантической сети в действительности не дают вполне ясного представления о понятии «левый» и «правый». Хотя существуют различного рода пространственные логики, они также являются достаточно абстрактными. Представлению объекта как единой целостности уделяется основное внимание во *фреймовых представлениях*.

Описание объекта или некоторой ситуации во фреймовых представлениях сконцентрировано в структуре вида:

Имя фрейма
Имя первого слота, значение первого слота
…
Имя *n*-го слота, значение *n*-го слота

Например:

Студент
Фамилия, Иванов
Пол, мужской
Курс, 4
Специальность, математика
Успеваемость, хорошо

В описании фрейма четко выделяется сама структура фрейма, или его оболочка, которая называется *протофрейм* – фрейм с незаполненными значениями слотов, и собственно данные, описывающие конкретную

реализацию или экземпляр фрейма – *экзофрейм*. В семантических сетях такое разделение на прототип и реализацию исходно не было предусмотрено, что вызывало некоторые трудности: с помощью семантических сетей можно описывать общие знания о мире, содержание некоторого рассказа или падежный фрейм, но нет четких правил, как использовать эти уровни описания одновременно.

Протофрейм, как правило, представляет собой не просто перечень слотов, но также и некие значения по умолчанию для них. Таким образом, протофрейм описывает некое стереотипное представление об объекте или ситуации. Когда наблюдается конкретный объект или ситуация, то создается экземпляр фрейма, в котором значения слотов исходно установлены по умолчанию. По мере накопления данных эти значения могут уточняться. Некоторые слоты могут не иметь значений по умолчанию. К примеру, у автомобиля или одежды нет типичного цвета.

Помимо значений по умолчанию для каждого из слотов могут указываться ограничения на его значение. Эти требования к фрейму позволяют определить, может ли некий объект по своим параметрам соответствовать данному фрейму, что позволяет выполнять своего рода распознавание.

В качестве содержимого слота может выступать и другой фрейм, что обеспечивает возможность описывать структурированные объекты, состоящие из других объектов. Таким образом, отношение «состоит из», которое было в семантических сетях просто меткой на дуге, во фреймах реализуется фактическим помещением одного фрейма в другой.

Например:

Группа
Номер, 4552
Кафедра, Вычислительная техника
Число студентов, 23
Студент, (Фрейм первого студента)
Студент, (Фрейм второго студента)
…

Содержимое слотов-фреймов в протофрейме будет также пусто (если нет значений по умолчанию), однако при этом между протофреймами будет установлена связь. Таким образом, во фреймовых представлениях протофреймы являются не изолированными единицами знаний, а входят в общую систему. Однако одного типа связей между фреймами явно недостаточно.

Помимо отношения «состоит из», между фреймами могут устанавливаться и отношения других типов. Чаще всего, это отношение типа «частное-общее». В таком отношении могут находиться фреймы «стул» и «мебель» или «студент» и «человек». Другие возможные типы отношений – пространственные, временные, причинно-следственные, в результате чего фреймы образуют структуру типа семантических сетей (см. рис 20). Однако в отличие от семантических сетей, здесь возникает вопрос, как подобные отношения учитывать при создании экземпляров

фреймов из протофреймов, что существенно увеличивает требования к четкости спецификации сетевых отношений.

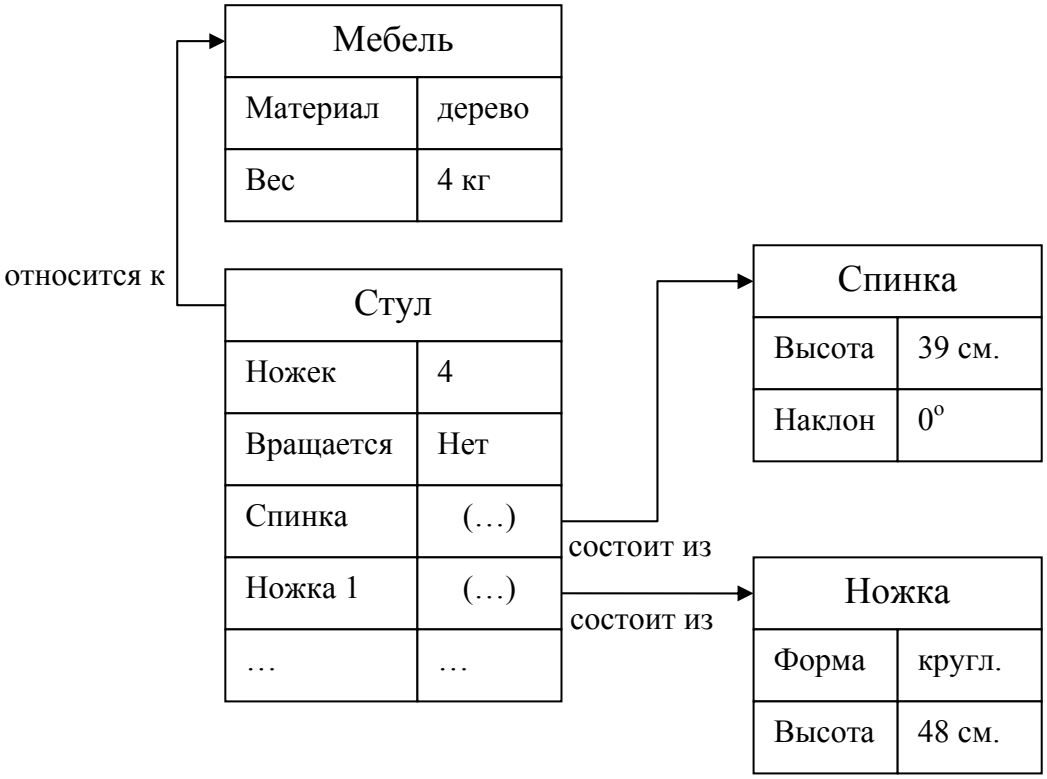


Рис. 20. Пример фрагмента сети фреймов

Отличительной особенностью фреймов является возможность присоединения к ним процедур, осуществляющих внутреннюю интерпретацию данных фрейма и описывающих их использование. В частности, процедурная информация помогает использовать произвольные типы отношений между фреймами.

Таким образом, во фреймовых представлениях объекты и ситуации описываются в виде единых структур, в которых выделяется протофрейм (сама структура) и экзофрейм (ее наполнение); протофрейм представляет собой стереотипное представление об объекте, включая информацию о типичных значениях своих слотов и присоединенные процедуры. Фреймы могут входить между собой в различные отношения, образуя структуру, подобную семантической сети.

Достаточно близким к фреймовому является объектно-ориентированный подход (ООП) к представлению знаний, который возник примерно в то же время, что и фреймовый подход, и развивался независимо от него. Если в логических представлениях основным компонентом являлось высказывание, а в семантических сетях – понятие, то в ООП, как и во фреймах, в центре рассмотрения находится объект, как целостное, но обладающее внутренней структурой, образование, наделенное некими свойствами и поведением.

Идеи данного подхода наиболее ярко выражены в объектно-ориентированном программировании, которое, однако, не следует

отождествлять с данным подходом в целом, поскольку оно имеет в значительной степени прикладную направленность. Тем не менее, объектно-ориентированное программирование также удобно использовать для иллюстрации основных идей ООП в области представления знаний.

Объектно-ориентированное программирование (далее также обозначаемое как ООП) базируется на трех главных принципах:

- инкапсуляция,
- наследование,
- полиморфизм.

При этом в ООП различается класс и его экземпляр (объект). Таким образом, понятие класса во многом похоже на понятие протофрейма, а объекта – на понятие экзофрейма.

Инкапсуляция означает, что в единую структуру объединена вся информация об объекте. Сама эта структура и является описанием класса. В этой структуре находятся все параметры (поля), характеризующие объект, а также процедуры (методы), которые этот объект умеет «выполнять», то есть описывают его поведение и взаимодействие с другими объектами. В этом смысле описание объекта во многом похоже на фрейм, состоящий из совокупности слотов и обладающий присоединенными процедурами. В ООП, однако, описания классов являются элементами текстов программ, на синтаксис которых накладываются жесткие ограничения. В частности, каждое поле в описании класса – это типизированная переменная (хотя в некоторых интерпретируемых языка типизация может быть нежесткой). При этом в качестве типа может выступать произвольный (но определенный в тексте программы) класс. То есть объект может содержать в себе другие объекты (отношение «состоит из»), что, опять же, близко к фреймам.

Наследование в ООП реализует отношение «частное-общее», однако здесь оно гораздо более четко определено. Так, если один класс является потомком другого, то он полностью наследует все поля и методы класса-родителя. При создании экземпляра класса-потомка создается объект, содержащий в себе и экземпляры всех родительских классов, прослеживаемых по иерархии наследования.

При наследовании, однако, зачастую бывает необходимо изменить поведение класса-потомка по сравнению с родительским классом (поскольку реализация некоторого действия, имеющего одно и то же название, может изменяться, конкретизируясь и наполняясь дополнительным содержанием для классов-потомков). Это реализуется в рамках концепции полиморфизма. Эта концепция становится мощным средством благодаря поддержке со стороны объектно-ориентированных языков программирования ряда технических приемов, таких как, например, таблицы виртуальных функций.

Несмотря на большое сходство ООП и фреймовых представлений, между ними есть ряд отличий. В ООП значения по умолчанию не выделяются в специфический элемент описания класса. Хотя

инициализация всех полей при создании объекта обычно присутствует, она не осуществляется типичными значениями соответствующих признаков. При этом, как правило, не хранится информация о том, записано ли в данное поле типичное значение признака или его конкретное значение для данного объекта. Также в ООП не регламентировано использование ограничений на значения полей, как это сделано во фреймовых представлениях для слотов. Разумеется, все это может быть запрограммировано, однако в ООП как представление знаний эти элементы не входят. Иными словами, класс не описывает стереотипное представление об объекте или ситуации (в отличие от протофрейма), а дает лишь каркас для создания объектов.

При этом, однако, в ООП гораздо прозрачнее смысл методов (присоединенных процедур), для которых также определена возможность полиморфизма (если, конечно, не говорить о реализации фреймов в языках программирования в стиле ООП). В ООП введены только два стандартных типизированных отношения: «состоит из» и «относится к», но они определены максимально четко по сравнению с тем, как это сделано в других представлениях знаний.

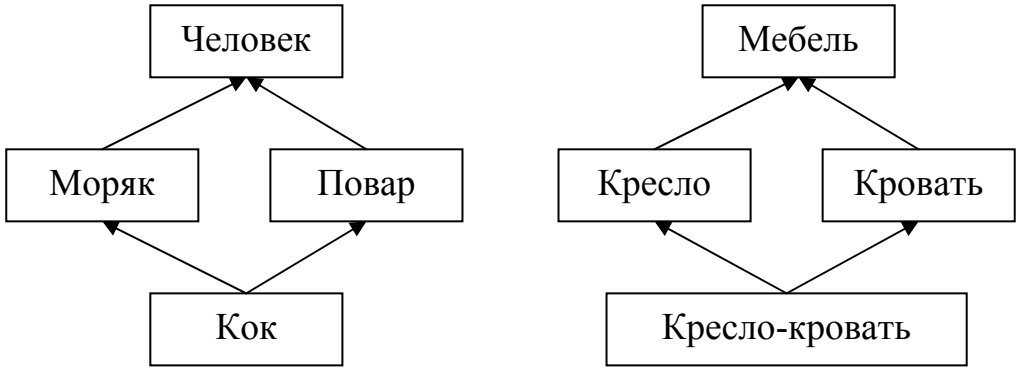


Рис. 21. Примеры иерархии классов, в которых возникает проблема множественного наследования

В то же время, и в ООП существуют несколько различные реализации основных принципов, а также существуют и некоторые их расширения. К примеру, в разных языках программирования несколько по-разному осуществляется решение проблемы множественного наследования. Суть этой проблемы отражена на рис. 21. При множественном наследовании может возникнуть ситуация, при которой свойства родительского класса наследуются несколько раз через какое-то количество его потомков. Возникает вопрос: следует ли дублировать поля родительского класса в классе-потомке, наследующем его несколько раз, или следует использовать лишь одну копию класса-родителя? В разных языках программирования проблема множественного наследования решается по-разному: оно может быть запрещено вообще или же может быть запрещено повторное наследование одного и того же класс (так что иерархия классов должно образовывать дерево, а не граф), либо же в

случая возникновения конфликтов наследования программист должен явно указывать компилятору, как их разрешать.

Проблема множественного наследования – это не чисто техническая проблема. Она возникает в объектно-ориентированном подходе как таковом. Объекты обладают большим числом признаков, наследование по которым неизбежно приводит к конфликтам в иерархии. Графически сущность этой проблемы представлена на рис. 22, где кружками отмечены множества объектов. В рамках ООП нельзя непосредственно представить понятия, образованные на основе произвольных множество объектов.

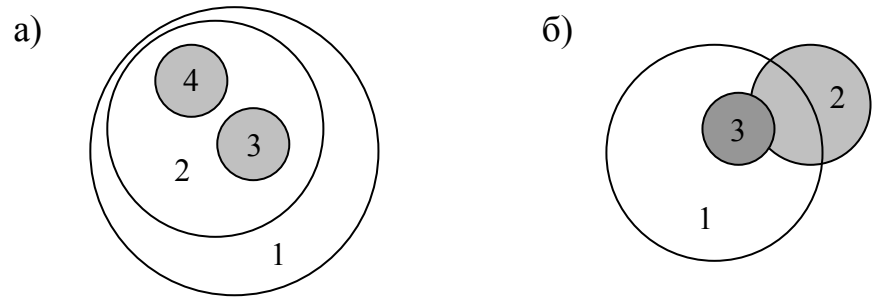


Рис. 22. Сущность проблемы декомпозиции системы знаний в иерархию понятий: а) множества объектов, допускающих корректное представление в виде иерархии классов; б) множества объектов, не описываемых через иерархию классов

Заметим также следующее. Если объектно-ориентированное программирование рассматривать как модель представления знаний, то следует заключить, что это представление процедурное (это не относится к объектно-ориентированному подходу вообще), в отличие от фреймов. Фреймы, как декларативное представление, дают полный доступ к своему содержимому для программ-интерпретаторов, осуществляющих манипулирование знаниями на основе фреймов. В ООП же исходные описания классов в процессе исполнения недоступны, что сильно снижает их ценность как представления знаний. Этот недостаток частично преодолевается с введением возможности интроспекции: языки ООП расширяются так, что классу в процессе исполнения становится доступна информация о его структуре и структуре иерархии классов, в которую он входит.

Еще одним интересным (с точки зрения ИИ) расширением стандартной концепции ООП является понятие метаклассов, реализованных в некоторых языках программирования. Метакласс отличается от класса так же, как класс отличается от объекта. Иными словами, класс является экземпляром метакласса. В некоторых языках (преимущественно интерпретируемых, а не компилируемых), например Ruby, есть возможность изменять состояния экземпляров метаклассов во время исполнения, так же как есть возможность изменять состояние экземпляров обычных классов в классическом ООП. Это приводит к тому, что программа во время исполнения сама может формироваться новые

классы или изменять существующие, что, например, может быть необходимо для обучающихся программ, для которых предустановленная иерархия классов является слишком жесткой и неприемлемой. В обычных языках (таких как C++) эти же возможности приходится реализовывать искусственно, например, в виде фабрик классов.

В разных языках ООП есть большое число собственных тонкостей, таких как передача сообщений, квалификаторы наследования, шаблоны и т.д. Все эти тонкости являются определяющими в удобстве использования того или иного языка, однако не относятся напрямую к проблеме представления знаний, поэтому мы их не рассматривали.

В заключение данного раздела скажем несколько слов об экспертных системах, которые сейчас часто обозначаются более общим термином – *интеллектуальные системы* (ИС). Экспертная система (ЭС) – это программный комплекс, который оперирует со знаниями в определенной предметной области в целях решения проблем или выработки рекомендаций. Экспертные системы имеют многочисленные применения: диагностика неисправностей (в технических и биологических системах), планирование, проектирование, анализ сложных объектов, а также анализ наблюдательных данных (хотя и весьма ограниченный). При этом от ЭС требуется не только получить какое-то решение, но и объяснить его пользователю.

Таким образом, ЭС включает следующие основные функции: приобретение знаний (то есть передача полезного опыта решения проблемы от экспертов или некоторого другого источника знаний и преобразование его в вид, позволяющий использовать эти знания в программе), представление знаний, управление процессом поиска решений, разъяснение принятого решения. Для функционирования экспертной системе требуется наличие следующих компонентов:

- базы знаний;
- машины логического вывода (или подсистемы манипуляции знаниями);
- блока общения.

База знаний подразумевает использование одного или нескольких представлений знаний для описания экспертных сведений о предметной области. Подсистема манипулирования знаниями необходима для поиска решения сформулированной пользователем задачи. При этом используются различные методы поиска в пространстве решений. Во многих случаях этот поиск представляется как логический вывод, который, однако, также требует привлечения методов эвристического программирования. И, наконец, блок общения осуществляет взаимодействие с пользователем, что необходимо не только для постановки задач, но также и для получения новых сведений самой ЭС. Такое взаимодействие часто осуществляется на ограниченном естественном языке и требует от ЭС неких лингвистических способностей.

Таким образом, для создания ЭС требуется использование основных результатов, которые были достигнуты в рамках двух главных парадигм в ИИ: поиске в пространстве решений и представлении знаний. К этому также добавляются результаты, полученные в области систем общения на естественном языке. Такая концентрация основных вопросов ИИ в одном месте на некоторое время сделала проблему экспертных систем одной из центральных в области искусственного интеллекта. При этом понятие ИИ даже подменялось понятием ЭС, что иногда случается и до сих пор.

Стоит при этом отметить, что работы в области ЭС имели преимущественно прикладную направленность, а проблемы представления и манипулирования знаний рассматривались в них в достаточно ограниченном контексте. ЭС применялись в узких предметных областях с четкой структурой, имеющей выражение в рамках символьных представлений. В то же время, работы в области ЭС поставили в области ИИ ряд новых проблем.

Желание преодолеть существенную специализированность ЭС привело к попыткам создания «пустых» ЭС, которые бы превращались в полноценные ЭС в результате лишь наполнения базы знаний. Однако вскоре выяснилось, что при переходе от одной предметной области к другой необходимо менять не только наполнение базы знаний, но и ее структуру (типы используемых представлений), а также методы манипулирования знаниями, используемые для решения задач.

Другой центральной проблемой в области ЭС стала проблема приобретения знаний. Формализация знаний экспертов оказалась крайне трудоемкой операцией, которая выполнялась т.н. инженерами знаний – людьми, которые общались с экспертами, осваивая на поверхностном уровне их предметную область для того, чтобы выделить в ней основные понятия и взаимосвязи между ними, то есть построить модель этой области.

В результате, стала очевидна необходимость автоматизации процесса приобретения знаний, что привело к выдвижению на первый план проблемы машинного обучения, которое также вызывает необходимость рассмотрения вопросов представления нечетких, неполных и недостоверных знаний и рассуждений в условиях неопределенности.

Вопросы и упражнения

1. Какую информацию содержит протофрейм?
2. Как во фреймовых представлениях реализуется отношение между понятиями «состоит из» или «содержит»?
3. Какие три концепции лежат в основе объектно-ориентированного программирования?
4. Чем отличаются объектно-ориентированные представления от фреймовых?
5. В чем заключается проблема множественного наследования?

6. Что такое метаклассы?
7. Какие основные компоненты экспертных систем?

10. Основные понятия распознавания образов

Как отмечалось ранее, развитие экспертных систем показало, что одной из наиболее трудоемких задач является приобретение знаний, то есть заполнение базы знаний для описания конкретной предметной области. Если даже для одной области объем знаний оказывается большим, то объем всех знаний одного человека об окружающем мире оказывается просто огромным, а формирование соответствующей базы знаний вручную – практически невозможным. В связи с этим, на первый план выходит задача автоматического приобретения знаний или, говоря шире, задача машинного обучения.

Дать общее определение обучению весьма проблематично, поскольку обучение многогранно. Однако для нас важно в обучении то, что оно помогает решать новые задачи на основе прошлого опыта. Таким образом, в обучении одним из основных компонентов является та исходная информация, на основе которой оно осуществляется. Без такой информации никакое обучение невозможно.

Рассмотрим следующий пример. Пусть школьник изучает квадратные уравнения. И пусть для некоторого уравнения $x^2+4x+3=0$ он получил ответ: $x_1=-3$, $x_2=-1$. Просто запомнив этот ответ, в следующий раз он сможет его быстро получить для этого же уравнения. Однако такое запоминание любого количества решений конкретных уравнений ни в коей мере не поможет ученику решить новое ранее не встречавшееся уравнение, даже если его форма не изменится. Очевидно, подобный результат обучения не является удовлетворительным. Вместо этого нам бы хотелось, чтобы обучающаяся система в результате решения нескольких задач, составляющих *обучающую выборку*, была в состоянии решать любые задачи данного типа.

Таким образом, обучение заключается не только в накоплении информации, но и в ее переработке, т.е. в переводе в некоторое представление, в рамках которого происходит обобщение. Условно это можно изобразить в виде схемы, представленной на рис. 23.

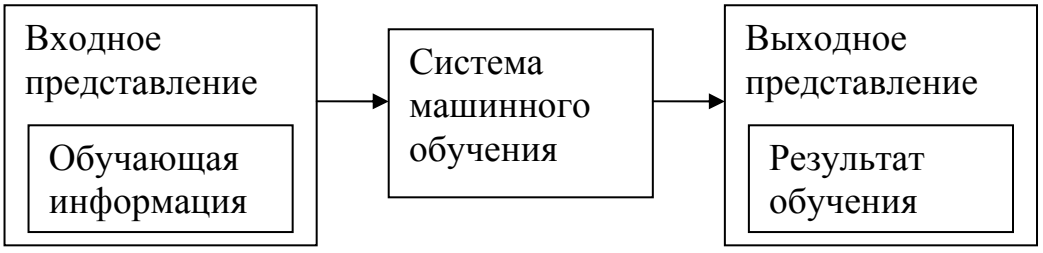


Рис. 23. Машинное обучение как преобразование информации

В системе обучения помимо обучающей информации основными компонентами являются выходное представление информации и собственно алгоритм обучения. Здесь мы не будем говорить об универсальных методах обучения, поскольку они, хотя и представляют теоретический интерес, на настоящий момент не имеют значимого прикладного применения. На практике же оказывается эффективным разделять проблемы обучения в зависимости от свойств обучающей выборки.

Во-первых, методы обучения различаются в зависимости от того, имеет ли обучающая информация количественный, логический или символьный характер. От этого меняется как привлекаемое выходное представление (в качестве которого может использоваться одно из ранее рассмотренных представлений знаний или некоторые другие представления информации), так и алгоритм обучения, в основе которого лежит поиск. В зависимости от входного представления поиск может реализовываться либо в непрерывном пространстве, либо в виде методов эвристического программирования.

Во-вторых, методы обучения могут быть разделены на *обучение с учителем*, *обучение с подкреплением* и *обучение без учителя* (также называемое самообучением). Если обучающая информация представляет собой совокупность задач, то при обучении с учителем системе машинного обучения помимо условия для каждой из этих задач также сообщается и правильное решение. При обучении с подкреплением системе не сообщается правильный ответ, но после решения каждой из задач ей сообщается, правильно ли была решена эта задача. При обучении без учителя системе сообщаются лишь условия задач. Сущуствуют и промежуточные ситуации, при которых, например, правильные ответы известны не для всех примеров обучающей выборки.

Обучение с учителем широко используется в практических приложениях, когда от системы требуется вполне определенное функционирование. При этом обучение с учителем играет роль своего рода настройки системы.

Обучение с подкреплением моделирует поведение организма в среде: например, когда организм ищет пищу, ему не сообщается правильный маршрут (как в обучении с учителем), но ему сообщается, нашел ли он в итоге пищу или нет, что сопровождается поощрением (удовольствием от еды) или наказанием (голодом). В связи с этим, обучение с подкреплением широко используется в моделях искусственной жизни и при исследовании адаптивного поведения с помощью аниматов.

Обучение без учителя реже используется в прикладных задачах, поскольку поведение системы после такого обучения в некотором смысле непредсказуемо, однако обучения без учителя предоставляет наиболее широкие возможности, в частности, оно свойственно научно-исследовательским задачам, в которых решение неизвестно и человеку.

В-третьих, обучение делится на инкрементное и неинкрементное. В неинкрементном обучении вся обучающая информация предоставляется системе одновременно. В инкрементном же обучении обучающие примеры предоставляются системе последовательно, и система должна корректировать результаты обучения, выполненного по предыдущим примерам, на основе новой информации. По сути, система дообучается в процессе функционирования.

Неинкрементное обучение чаще используется в прикладных задачах, поскольку соответствует одноразовой настройке системы. В случае инкрементного обучения поведение системы меняется в процессе функционирования, что уменьшает ее предсказуемость и осложняет адаптацию человека к этой системе, если она используется в качестве инструмента. В то же время, инкрементное обучение более жизненно и имеет более широкие возможности, если речь идет о построении сравнительно автономной системы, поскольку инкрементность обучения тесно связана с адаптационными способностями системы в изменяющейся среде.

Более подробное рассмотрение вопросов обучения без формальной постановки конкретных задач обучения. В качестве одной из таких задач мы рассмотрим задачу распознавания образов.

Формирование распознавания образов как научной дисциплины началось в 50-х годах XX века (почти одновременно с зарождением науки об искусственном интеллекте). Многие первоначальные работы в этой области были посвящены построению автоматов, предназначенных для чтения печатных и рукописных знаков, с чем и было связано введение в употребление термина «образ». Существенное влияние на развитие этой области также оказал перцептронный подход, предложенный Ф. Розенблаттом в конце 50-х годов. Привлекаемый для решения задач распознавания образов математический аппарат постепенно расширялся, включая теорию статистических решений, математическую логику, теорию информации, теорию формальных грамматик и т.д. Вместе с тем росла и сфера применения этой области.

Задачи распознавания как прикладные задачи возникают во многих отраслях науки и техники, в связи с чем методы распознавания часто выделяют в самостоятельный раздел прикладной математики. Однако наибольший прогресс методов распознавания связан с областью ИИ. Некоторое время назад существовало мнение (во многом справедливое и сейчас), согласно которому создание искусственного интеллекта – это, прежде всего, построение распознающих систем, приближающихся по своим возможностям к возможностям человека. Когда в 1980-х гг. в качестве отдельного вопроса ИИ выделилась область машинного обучения, она включила в свое рассмотрение под определенным ракурсом значительную часть проблематики распознавания образов. При этом в области машинного обучения до сих пор наиболее развитыми остаются именно методы распознавания образов.

Когда человек узнает своего знакомого в толпе людей, определяет, является ли данное число простым или составным, или устанавливает класс звезды по ее спектру, он решает, казалось бы, совершенно разные задачи. Однако можно заметить и общие черты. В каждой из задач присутствует некоторый объект (человек, число, звезда), представленный значениями своих признаков, а также некоторое количество классов, к одному из которых необходимо отнести данный объект. Таким образом, распознавание – это отнесение некоторого неизвестного объекта по его описанию к одному из классов.

Возможна и другая задача: разделить заданное множество объектов на классы (задача таксономии). В теории распознавания образов ставится следующий основной вопрос: могут ли столь разные задачи иметь одно и то же математическое описание и сходные алгоритм решения?

Сложность этого вопроса заключается в том, что исходные описания очень сильно отличаются в различных приложениях, так же как и способы объединения объектов в классы. Тем не менее, существует несколько общих подходов к распознаванию образов, которые позволяют решать широкие классы задач. Как и в машинном обучении вообще, в распознавании образов можно выделить логические (работающие с логическими представлениями), синтаксические (работающие с символьными представлениями) и дискриминантные (работающие с непрерывными представлениями) методы. Здесь мы рассмотрим дискриминантные методы. Для начала, введем основные понятия.

Пусть $\mathbf{x} \in \mathrm{X}$ – *описание объекта* (или *образ*), а X – *пространство описаний* (множество всех возможных образов). В дискриминантном подходе к распознаванию простанство $\mathrm{X}=R^N$ – пространство признаков, а образ $\mathbf{x}$ – N-компонентный вектор признаков: $\mathbf{x} = (x_1,...,x_N)$.

Через $\mathrm{A} = \{a_1, a_2, ..., a_d\}$ обозначим некоторое множество, состоящее из d элементов, $1 < d < +\infty$, где a_i – i-й *класс образов*, а A – *множество классов* (также называемое алфавитом классов).

Решающим правилом назовем отображение $\varphi : \mathrm{X} \to \mathrm{A}$, которое ставит в соответствие элементу пространства описаний класс из заданного множества.

Решающее правило может также задаваться неявно через целевую функцию $\rho : \mathrm{X} \times \mathrm{A} \to R$, определяющую степень соответствия (например, в форме вероятности) между описанием объекта и каждым классом. Решающее правило можно определить через целевую функцию как

$$\varphi(\vec{x}) = \arg\max_{a \in \mathrm{A}} \rho(\vec{x}, a). \qquad (10)$$

Во многих практических задачах вводится матрица потерь L_{ij}, определяющая стоимость ошибочного отнесения объекта класса i к классу j, а задача формулируется, как минимизация ожидаемых потерь в ходе классификации. Однако учет потерь при классификации нужен лишь при принятии решения, к какому классу отнести данный объект, но не на

процедуру вывода вероятностей принадлежности объекта тому или иному классу. Конечно, в подходах, не опирающихся на теорию вероятностей, матрица потерь непосредственно влияет на решающее правило, но суть подходов не меняется и в том случае, если эта матрица не используется. В связи с этим, для простоты изложения матрица потерь будет опускаться.

Теперь сформулируем задачи распознавания образов как задачи машинного обучения. В зависимости от имеющейся информации и характера обучающей выборки можно выделить следующие задачи.

Задача *классификации* (распознавания без обучения) заключается в определении по описанию объекта того класса, к которому он принадлежит. При этом решающие правила считаются известными. Иными словами, распознавание единичного образа (далее будет использоваться термин «классификация» для избежания путаницы) сводится к применению решающего правила φ к данному образу $\mathbf{x}$.

Собственно задача *распознавания образов* (обучения с учителем) заключается в построении решающего правила, которое считалось известным в задаче классификации. В качестве исходной информации здесь выступает обучающая выборка, каждый элемент которой представляет собой описание объекта (задача) и соответствующий ему класс (ответ).

В задаче *группирования* количество исходной информации меньше, а именно, в ней не определено пространство классов A, которое и требуется сформировать, опираясь на заданный набор образов, не разбитых на классы в отличие от задачи распознавания с учителем. Иными словами, здесь в обучающей выборке имеется только набор задач, предоставленных без правильных ответов.

Задача классификации в рамках дискриминантного подхода является очень простой, коль скоро известно решающее правило. Обычно, однако, решающее правило здесь представляется в специфической форме. Введем следующее определение.

Решающей функцией $\kappa(\mathbf{x})$ для двух классов $a_1, a_2 \in \mathrm{A}$ назовем такую функцию $\kappa : X \to R$, что $\kappa(\mathbf{x}) > 0$, если образ $\mathbf{x}$ принадлежит классу a_1, и $\kappa(\mathbf{x}) < 0$, если образ $\mathbf{x}$ принадлежит классу a_2.

На основе подобной решающей функции не составляет проблем сформировать решающее правило. Преимущество решающих функций в том, что у них не только аргументы, но и принимаемые значения имеют непрерывный характер, что позволяет применять к ним обширный аппарат математического анализа.

Уравнение $\kappa(\mathbf{x}) = 0$ задает поверхность, разделяющую два класса и называемую *дискриминантной поверхностью*. Поскольку при принятии решения об отнесении образа к тому или иному классу абсолютные значения функции $\kappa(\mathbf{x})$ внутри классов роли не играют (то есть не имеет значения, как именно распределены образы внутри классов), всю необходимую информацию о том, как следует разделять классы, несет

именно эта поверхность, описывающая границу между классами в пространстве признаков (см. рис. 24). Это и объясняет название дискриминантного подхода.

Разделяющая поверхность удобна для принятия решений при классификации образов. Однако класс может быть задан не через его границы с другими классами, а как отдельная область в пространстве признаков. Если области, соответствующие двум классам, не пересекаются, то эти классы называются *разделимыми* в данном пространстве признаков. Иными словами, разделимость классов означает, что для них существует решающая функция, корректно классифицирующая любой образ. Напротив, если области пересекаются, то такой функции не существует, и классы называются *неразделимыми*. Например, классы, представленные на рис. 24 неразделимы, если пространство признаков ограничить лишь признаком x_2. В то же время они разделимы в исходном пространстве признаков.

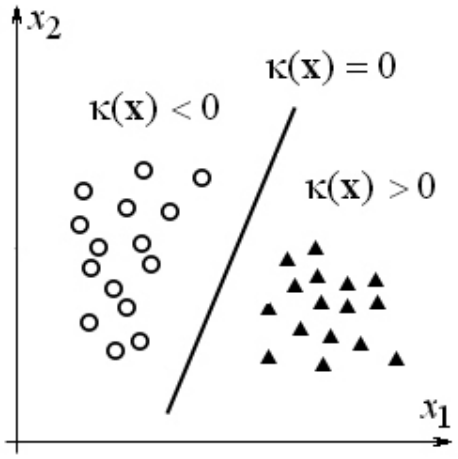


Рис. 24. Пример разделения образов на два класса решающей функцией $\kappa(\mathbf{x})$ в двумерном пространстве признаков

В случае нескольких классов возможны различные определения решающей функции. Один из способов заключается в том, чтобы непосредственно воспользоваться определением для случая двух классов и ввести d^2 решающих функций $\kappa_{ij}(\mathbf{x})$ (где d – это количество классов), каждая из которых разделяет два разных класса $a_i, a_j \in \mathrm{A}$. Для таких решающих функций верно $\kappa_{ij}(\mathbf{x}) > 0$, если образ $\mathbf{x}$ не может принадлежать классу a_j, и $\kappa_{ij}(\mathbf{x}) < 0$, если образ не может принадлежать классу a_i. Решающее правило примет форму

$$\varphi(\mathbf{x}) = a_i \Leftrightarrow (\forall j)\kappa_{ij}(\mathbf{x}) > 0. \qquad (11)$$

Поскольку верно равенство $\kappa_{ij}(\mathbf{x}) = -\kappa_{ji}(\mathbf{x})$, а функции $\kappa_{ii}(\mathbf{x})$ лишены смысла, всего требуется построить $d(d-1)/2$ решающих функций.

Другой способ заключается в отделении данного класса одновременно ото всех остальных. Для этого необходимо d дискриминантных функций $\kappa_i(\mathbf{x})$, а решающее правило примет форму

$$\varphi(\mathbf{x}) = a_i \Leftrightarrow \kappa_i(\mathbf{x}) > 0. \qquad (12)$$

Естественно, должно выполняться условие

$$\kappa_i(\mathbf{x}) > 0 \Rightarrow (\forall j : j \neq i)\kappa_j(\mathbf{x}) < 0. \qquad (14)$$

Второй вариант кажется предпочтительнее, так как требует меньшего количества решающих функций. Однако построение таких решающих функций сложнее, особенно если они выбираются из простых семейств.

К примеру, на рис. 25 представлены четыре класса, не каждый из которых может быть отделен ото всех остальных линейной решающей функцией. Попарно все классы являются линейно разделимыми, поэтому достаточно шести решающих функций, по которым можно было бы определить принадлежность образа соответствующему классу. Однако благодаря тому, что одни и те же решающие функции могут быть использованы для разделения различных пар классов, оказывается достаточно лишь двух решающих функций, что даже лучше четырех решающих функций, которые бы потребовались в случае отделения каждого класса ото всех остальных.

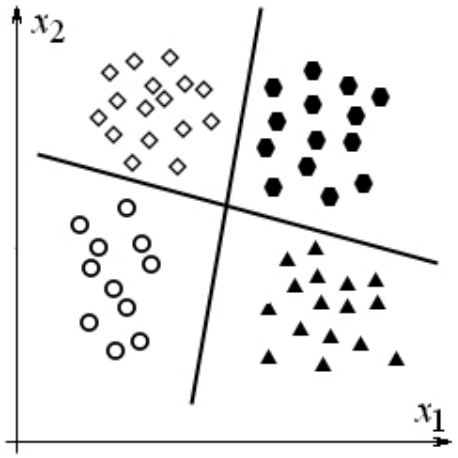


Рис. 25. Пример четырех классов, для которых отделение каждого класса ото всех остальных оказывается проблематичным, хотя для правильной классификации достаточно лишь двух линейных решающих функций

В общем случае решающую функцию можно определить как функцию, разделяющую два подмножества множества классов. Видно, что это определение обобщает оба подхода: в первом производится разделение таких множеств, как $\{a_i\}$ и $\{a_j\}$, а во втором случае – $\{a_i\}$ и $\mathrm{A} \setminus \{a_i\}$.

Разделение подмножеств классов может позволить еще уменьшить количество решающих функций (см. рис. 25), но их число не может быть меньше $[\log_2 d]$.

Задача минимизации количества решающих функций, достаточных для классификации образов, может быть очень важна, особенно если количество классов *d* велико. Если представить себе, с каким количеством классов объектов (или понятий) имеет дело человек, становится ясно, что решение этой проблемы в том или ином виде потребуется при разработке универсальной системы машинного обучения. Мы, однако, этот вопрос здесь рассматривать не будем, а перейдем к проблеме распознавания образов.

Для простоты будем рассматривать случай двух классов. В задаче распознавания образов в качестве исходных данных выступает обучающая выборка: $(\mathbf{x}_1, A_1),(\mathbf{x}_2, A_2),...,(\mathbf{x}_M, A_M)$, где $\mathbf{x}_i \in R^N$ и $A_i \in \{a_1, a_2\}$, состоящая из *M* элементов. На основе этих данных требуется построить решающее правило $\varphi : \mathrm{X} \to \mathrm{A}$ или решающую функцию $\kappa(\mathbf{x})$.

Однако уже при постановке этой задачи возникает вопрос, какие требования следует предъявлять к решающему правилу? Казалось бы, от этого правила требуется лишь, чтобы оно правильно классифицировало все образы обучающей выборки. Но сразу ясно, что это требование не является достаточным, даже если считать, что классы разделимы.

Действительно, можно представить такое решающее правило, которое «помнит» примеры обучающей выборки и возвращает для них правильные номера классов, а для всех других образов возвращает некоторое случайное значение. Как было сказано ранее, такой результат обучения совершенно неудовлетворителен. Стоит подчеркнуть, что такого рода проблемы одинаковы для всех задач обучения, и все они сводятся к заданию адекватного критерия качества результата обучения (в данном случае – критерия качества решающей функции).

Прямого ответа на этот вопрос можно избежать, если конструировать методы распознавания на основе неких эвристических соображений. Два наиболее широко распространенных эвристических метода – это метод эталонных образов и метод ближайшего соседа.

Метод эталонных образов

Метод эталонных образов – это один из эвристических методов построения решающих правил. В основу этого метода положена идея, которая заключается в том, что некоторая совокупность объектов, объединенных в отдельный класс, может быть представлена одним или несколькими эталонными объектами. Эти эталонные объекты являются наиболее типичными представителями класса. Типичность эталонного объекта означает, что он в среднем максимально похож на все объекты класса.

Поскольку сходство двух объектов может трактоваться как величина, противоположная расстоянию между ними в пространстве описаний (образов), то эталон – это объект, для которого минимально среднее расстояние до других объектов.

Пусть в обучающей выборке первому классу соответствует M_1 элементов $\mathbf{x}_{1,i}$, а второму классу – M_2 элементов $\mathbf{x}_{2,i}$. Тогда эталонные образы для каждого из классов могут быть определены как

$$\mathbf{x}_{0,1} = \frac{1}{M_1}\sum_{i=1}^{M_1}\mathbf{x}_{1,i}\,,\ \mathbf{x}_{0,2} = \frac{1}{M_2}\sum_{i=1}^{M_2}\mathbf{x}_{2,i}\,. \qquad (15)$$

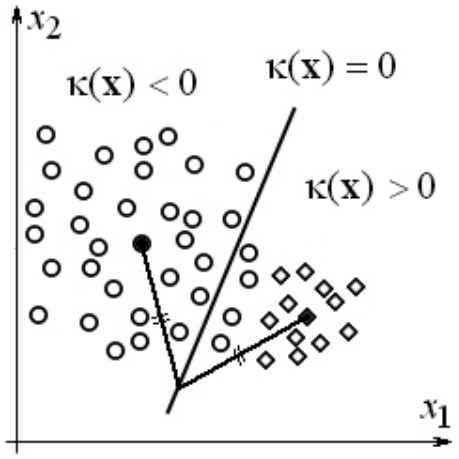


Рис. 26. Пример двух классов разных размеров, для которых критерий евклидового расстояния до эталонных образов не является корректным

Классы, однако, могут обладать разными свойствами. Простейшим свойством является характерный размер класса (см. рис. 26), который может быть оценен как

$$r_1 = \sqrt{\frac{1}{M_1}\sum_{i=1}^{M_1}\left|\mathbf{x}_{0,1} - \mathbf{x}_{1,i}\right|^2}\,,\ r_2 = \sqrt{\frac{1}{M_2}\sum_{i=1}^{M_2}\left|\mathbf{x}_{0,2} - \mathbf{x}_{2,i}\right|^2}\,. \qquad (16)$$

Тогда для классификации нового образа $\mathbf{x}$ используется следующая решающая функция:

$$\kappa(\mathbf{x}) = \frac{\left|\mathbf{x} - \mathbf{x}_{0,1}\right|}{r_1} - \frac{\left|\mathbf{x} - \mathbf{x}_{0,2}\right|}{r_2}. \qquad (17)$$

Также может использоваться функция

$$\kappa(\mathbf{x}) = \frac{\left|\mathbf{x} - \mathbf{x}_{0,1}\right|^2}{r_1^2} - \frac{\left|\mathbf{x} - \mathbf{x}_{0,2}\right|^2}{r_2^2}, \qquad (18)$$

которая задает ту же разделяющую поверхность.

Если значение этой функции отрицательное, то образ относится к первому классу, в противном случае – ко второму. Разделяющая поверхность для двух классов задается уравнением $\kappa(\mathbf{x}) = 0$.

Метод ближайшего соседа

Другой широко распространенный эвристический метод распознавания – метод ближайшего соседа (или его обобщение – метод *k*-ближайших соседей).

Идея этого метода крайне проста: новый образ относится к тому классу, к которому он ближе. При этом расстояние от образа до класса определяется как расстояние от образа до ближайшего элемента класса.

Тогда на основе обучающей выборки $\mathbf{x}_i, \alpha_i$, i=1,…,M, может быть построено следующее решающее правило:

$$\varphi(\mathbf{x}) = \alpha_{\arg\min_i |\mathbf{x}-\mathbf{x}_i|}. \qquad (19)$$

В соответствии с данным решающим правилом просматривается вся обучающая выборка, в ней находится образ, расположенный наиболее близко к данному и устанавливается, к какому классу он принадлежит (это известно, поскольку он находится в обучающей выборке). Этот класс и приписывается новому образу.

Метод ближайшего соседа весьма чувствителен к *выбросам*, то есть тем образам обучающей выборки, для которых указаны ошибочные классы. В методе *k*-ближайших соседей выбирается *k* образов обучающей выборки, наиболее близко расположенных к классифицируемому образу, и определяется, к какому классу относится больше всего из них. Поскольку выбросов, как правило, значительно меньше, чем правильных примеров, можно надеяться, что среди *k* ближайших соседей выбросов будет мало, и они не окажут влияния на результат классификации.

У метода эталонных образов также имеются модификации, в частности, в одной из них каждый класс может описываться несколькими эталонами, а классификация осуществляется так же, как и в методе ближайшего соседа, но вместо образов выборки ищется ближайший эталон. В предельном случае каждый образ выборки может выступать в роли эталона и метод эталонных образов превратится в обычный метод ближайшего соседа. Таким образом, эти два метода – два крайних случая классификации с использованием функций расстояния.

Оба метода могут быть расширены за счет использования неевклидовой метрики. В общем случае используется произвольная функция расстояния $s(\mathbf{x},\mathbf{y})$. В методе ближайшего соседа решающее правило просто преобразуется к виду:

$$\varphi(\mathbf{x}) = \alpha_{\arg\min_i s(\mathbf{x},\mathbf{x}_i)}. \qquad (20)$$

Реализация же метода эталонных образов усложняется. Поскольку эталон – это объект, для которого минимально среднее (или среднеквадратичное) расстояние до других объектов, то для поиска эталонов следует минимизировать величины:

$$S(\mathbf{x}_{0,1}) = \frac{1}{M_1}\sum_{i=1}^{M_1} s^2(\mathbf{x}_{0,1} - \mathbf{x}_{1,i}),\ S(\mathbf{x}_{0,2}) = \frac{1}{M_2}\sum_{i=1}^{M_1} s^2(\mathbf{x}_{0,2} - \mathbf{x}_{2,i}). \quad (21)$$

Запишем необходимое условие экстремума для эталонного образа первого класса:

$$0 = \frac{\partial S(\mathbf{x}_{0,1})}{\partial \mathbf{x}_{0,1}} = \frac{1}{M_1}\sum_{i=1}^{M_1} s(\mathbf{x}_{0,1} - \mathbf{x}_{1,i})\frac{\partial s(\mathbf{x}_{0,1} - \mathbf{x}_{1,i})}{\partial \mathbf{x}_{0,1}}. \quad (22)$$

Несложно убедиться, что для евклидового расстояния эталонный образ, полученный из этого условия, будет соответствовать вектору средних.

Помимо евклидового расстояния могут использоваться и другие функции расстояния, например,

$$s(\mathbf{x},\mathbf{y}) = \frac{\mathbf{x}^T\mathbf{y}}{\|\mathbf{x}\|\cdot\|\mathbf{y}\|} \text{ или } s(\mathbf{x},\mathbf{y}) = \frac{\mathbf{x}^T\mathbf{y}}{\mathbf{x}^T\mathbf{x} + \mathbf{y}^T\mathbf{y} - \mathbf{x}^T\mathbf{y}}. \quad (23)$$

Выбор меры сходства основывается на знании природы входных данных. Например, если объектами являются прямые линии, а векторы признаков – это векторы, описывающих их направление, то использовать в качестве меры сходства угол между ними будет предпочтительнее, чем евклидово расстояние. Если же рассматриваются отрезки прямых линий, то необходимо также учитывать и различие их длин, что приведет к специфической для данных объектов мере сходства. Выбор меры сходства обычно задается человеком, а не осуществляется автоматически.

Однако мера сходства может задаваться в параметрическом виде (с автоматическим выбором значений параметров). Наиболее типичный пример – расстояние Махаланобиса, которое имеет вид

$$s(\mathbf{x},\mathbf{y}) = (\mathbf{x}-\mathbf{y})^T\mathbf{C}^{-1}(\mathbf{x}-\mathbf{y}), \quad (24)$$

где $\mathbf{C}$ – некоторая матрица, элементы которой и являются параметрами данной функции расстояния. На практике в качестве этой матрицы берется ковариационная матрица распределения векторов данного класса:

$$\mathbf{C} = \frac{1}{M_1}\sum_{i=1}^{M_1}\left[(\mathbf{x}_{i,1} - \mathbf{x}_{0,1})(\mathbf{x}_{i,1} - \mathbf{x}_{0,1})^T\right],\ \mathbf{x}_{0,1} = \frac{1}{M_1}\sum_{i=1}^{M_1}\mathbf{x}_{1,i}, \quad (25)$$

где $\mathbf{x}_{0,1}$ – вектор средних для данного (первого класса). Для второго класса функция расстояний вычисляется аналогичным образом, но значения ее параметров будут другими.

Стоит отметить следующий частный случай диагональной ковариационной матрицы с одинаковой дисперсией по каждой координате:

$$\mathbf{C} = \begin{pmatrix} r_1^2 & 0 & \dots & 0 \\ 0 & r_1^2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & r_1^2 \end{pmatrix}. \quad (26)$$

В этом случае расстояние Махаланобиса примет вид:

$$s_1(\mathbf{x},\mathbf{y}) = \frac{|\mathbf{x}-\mathbf{y}|^2}{r_1^2} \text{ и } s_2(\mathbf{x},\mathbf{y}) = \frac{|\mathbf{x}-\mathbf{y}|^2}{r_2^2} \quad (27)$$

для первого и второго классов соответственно, что приведет к решающей функции (18). Таким образом, нормирование расстояния до эталонного образа на «радиус» класса является лишь частным случаем расстояния Махаланобиса, когда распределение векторов внутри классов является сферическим. В общем случае расстоянию Махаланобиса соответствует некоторый эллипс, описанный вокруг класса образов и учитывающий вытянутость и ориентацию этого класса.

Таким образом, расстояние Махаланобиса является более общим, чем евклидово расстояния, даже если в последнем учитываются размеры классов. Однако обоснование выбора в качестве матрицы **C** ковариационной матрицы не может быть осуществлено в рамках метода эталонных образов, что показывает ограниченность эвристических методов и необходимость более строго математического анализа проблемы.

Эвристичность описанных методов заключается в предположении, согласно которому близко расположенные образы, вероятнее всего принадлежат одному и тому же классу. Даже использование неевклидовой метрики не нарушает этого предположения. Стоит отметить, что в методе ближайшего соседа, по сути, производится просто запоминание частных примеров без какого-либо обобщения. Как мы видели раньше, такой способ обучения в обычной ситуации не должен был бы работать вообще. Работоспособность метода ближайшего соседа обеспечивается только предположением о совпадении классов у близко расположенных образов.

Таким образом, это очень мощная эвристика (если указанное предположение не нарушается). Эта эвристика присутствует и в других дискриминантных методах распознавания: предположение о существовании разделяющей поверхности означает, что решающее правило непрерывно везде, кроме границы классов, имеющей размерность на 1 меньше, чем размерность пространства признаков. Рассмотрим следующий утрированный пример, в котором это предположение нарушается.

Пусть есть два класса, заданных перечнем образов: {(18, 7), (6, 3), (11, 17), (2, 14)} и {(13.334, 16.825), (21.12, 6.1), (5.85, 10.13), (7.31, 3.7)}. Каждый образ описывается двумя признаками, а в качестве пространства признаков выступает R^2, что вполне согласуется с дискриминантным подходом. Требуется построить решающее правило и определить класс, к которому относится образ (14, 20). На основе, например, правила ближайшего соседа результат будет выглядеть, как показано на рис. 27. Но человек, не особо задумываясь и не имея дополнительных априорных сведений, построит совершенно другое решающее правило: векторы с

целочисленными компонентами относятся к первому классу, а с дробными – ко второму.

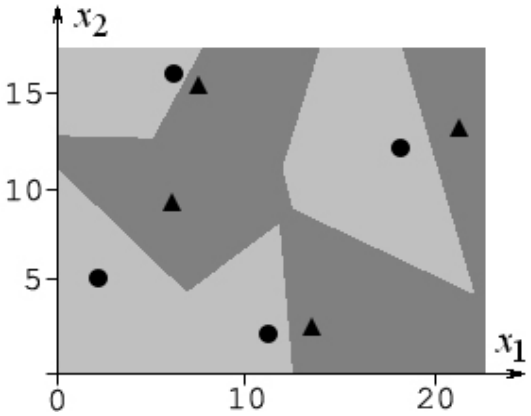


Рис. 27. Попытка разделения двух классов: целых (круги) и дробных (треугольники) чисел, – при использовании правила ближайшего соседа

В рамках дискриминантного подхода такая решающая функция построена быть не может ни одним из существующих подходов, так как нарушается предположение о непрерывности. Таким образом, неявное предположение о непрерывности является мощнейшей эвристикой, но в то же время оно накладывает и определенные ограничения, которые нужно иметь в виду при проектировании систем распознавания.

Вопросы и упражнения

1. Перечислить основные типы машинного обучения.
2. Чем решающее правило отличается от решающей функции?
3. Какое минимальное количество решающих функций может обеспечить классификацию образов из n классов?
4. Какие эвристические методы распознавания образов существуют?
5. Какой из эвристических методов распознавания наиболее чувствителен к выбросам?
6. Какими формами обладают разделяющие поверхности, строящиеся в методе эталонных образов и методе ближайшего соседа?

11. Метод решающих функций и метод опорных векторов

Метод решающих функций

В эвристических методах распознавания проблему критерия качества решающей функции удается избежать за счет использования некоторых неявных предположений. Более строгим подходом является явное построение решающих функций. Обычно рассматриваются не произвольные решающие функции, а лишь функции, относящиеся к некоторому параметрическому семейству, элементы которого $\kappa(\mathbf{x}, \mathbf{w})$ определяются вектором параметров $\mathbf{w} = (w_1, ..., w_n)$, где n – количество

параметров. Выбор конкретных значений параметров w_i соответствует выбору решающей функции. Тогда решение задачи распознавания образов сводится к определению оптимальных значений параметров по обучающей выборке.

Мы рассматриваем задачу построения решающих функций для случая двух классов a_1 и a_2. Как и раньше, в задаче распознавания имеются исходные данные: $D = \left((\mathbf{x}_1, A_1), (\mathbf{x}_2, A_2), ..., (\mathbf{x}_M, A_M)\right)$ – обучающая выборка из M элементов, в которой $\mathbf{x}_i \in R^N$ – образы, представленные N-компонентными векторами признаков, а $A_i \in \{a_1, a_2\}$ – соответствующие им классы.

Важное параметрическое семейство составляют *линейные решающие функции*. Поиск линейных решающих функций проще как с теоретической, так и с практической точки зрения, а классификаторы, построенные на их основе, являются также и наиболее эффективными в смысле требуемых вычислительных ресурсов. Линейные решающие функции задаются следующим образом:

$$\kappa(\mathbf{x}, \mathbf{w}) = w_1 x_1 + w_2 x_2 + ... + w_N x_N + w_{N+1} = \mathbf{w}\mathbf{x}', \qquad (28)$$

где $\mathbf{x}' = (x_1, x_2, ..., x_N, 1)^T$ – пополненный вектор признаков, а $\mathbf{w} = (w_1, w_2, ..., w_{N+1})$ – вектор весов, который требуется определить по обучающей выборке, исходя из условий: $\mathbf{w}\mathbf{x}'_i > 0$, если $A_i = a_1$, и $\mathbf{w}\mathbf{x}'_i < 0$, если $A_i = a_2$. Разделяющая поверхность, задаваемая уравнением $\mathbf{w}\mathbf{x}'_i = 0$, в данном случае будет гиперплоскостью.

Чтобы представить эти условия единообразно, обычно пользуются следующим приемом. Пусть $z_i = 1$, если $A_i = a_1$, и $z_i = -1$, если $A_i = a_2$, тогда ограничения на вектор параметров будут следующие:

$$(\forall i) z_i \mathbf{w}\mathbf{x}'_i > 0. \qquad (29)$$

Если хотя бы одно решение существует, то их бесконечно много. Это очевидно из тех соображений, что неравенства в (29) являются строгими, то есть ограничения на каждый компонент вектора $\mathbf{w}$ при фиксированных других компонентах задаются в виде пересечения интервалов (открытых множеств), которые либо содержат бесконечно много точек, либо являются пустыми. А поскольку решений может быть бесконечно много, необходим некоторый дополнительный критерий, однозначно определяющий, какая из разделяющих гиперплоскостей лучше. Здесь мы опять сталкиваемся с проблемой критерия качества решающей функции.

В зависимости от критерия качества и метода поиска параметров, максимизирующих этот критерий, могут быть построены различные процедуры нахождения линейных решающих функций.

Одной из идей здесь является применение классического метода наименьших квадратов (МНК). Необходимо, чтобы решающая функция правильно классифицировала образы обучающей выборки. Это можно выразить в виде следующего условия: $\kappa(\mathbf{x}_i, \mathbf{w}) = z_i$. Тогда задача

распознавания сводится к задаче аппроксимации, для которой в рамках МНК можно записать следующую целевую функцию:

$$L = \frac{1}{M}\sum_{i=1}^{M}\left(\kappa(\mathbf{x}_i, \mathbf{w}) - z_i\right)^2 . \qquad (30)$$

Удобство линейных решающих функций в том, что для их нахождения используется линейный МНК, имеющий эффективное решение. Для определения значений параметров, при которых достигается минимум критерия (30), продифференцируем его и получим систему линейных уравнений:

$$\frac{\partial L}{\partial w_k} = \frac{2}{M}\sum_{i=1}^{M}\left(\sum_{j=1}^{N+1} w_j x'_{i,j} - z_i\right)x'_k = 0, \qquad (31)$$

которую не представляет сложности решить.

Данный метод вполне можно использовать, однако вызывает сомнения применение суммы квадратов разностей (30) в качестве критерия оптимальности решающей функции. Действительно, условие $\kappa(\mathbf{x}_i, \mathbf{w}) = z_i$ вовсе не является необходимым для правильной классификации образов обучающей выборки. Более того, поскольку z_i принимают только -1 и 1 в качестве значений, строгое равенство невозможно, по крайней мере, для линейных решающих функций. В связи с этим, осуществлялись попытки введения более осмысленного критерия качества, наибольшее внимание из которых заслуживает метод опорных векторов.

Метод опорных векторов

Основы метода опорных векторов были заложены в начале 70-х годов прошлого века. Популярность же метода, которой он начал пользоваться в 1990-х годах после выхода ряда работ, вызвана открытием интересной возможности его расширения на случай нелинейных разделяющих границ (этот способ обобщения метода несколько отличен от тех, которые применялись для других линейных методов). Долгое время этот метод использовался только для обучения с учителем, но сейчас открыта возможность его применения и при обучении без учителя, которые мы, однако, рассматривать не будем.

Смысл критерия качества, привлекающегося в методе опорных векторов, заключается в следующем. Пусть есть некоторая разделяющая гиперплоскость. Ее можно перемещать параллельным переносом (меняя параметр w_{N+1}) в некоторых пределах так, что она все еще будет разделять классы. Совокупность таких параллельных гиперплоскостей образует полосу определенной ширины. В зависимости от ориентации, определяемой параметрами $w_1, w_2, ..., w_N$, ширина полосы будет различной.

К примеру, на рис. 28 представлены два класса, которые могут быть разделены различными линейными решающими функциями. При фиксированной ориентации разделяющие поверхности заметают полосу некоторой ширины. На рисунке представлены две такие полосы, обладающие разными ширинами: s_1 и s_2. Полагается, что чем шире полоса, тем лучше соответствующая поверхность разделяет классы. С каждой полосой соприкасается, по крайней мере, по одному вектору из каждого класса. Эти векторы называются *опорными* (на рисунке представлены заполненными кружками и треугольниками). Опорные векторы могут меняться в зависимости от ориентации полосы.

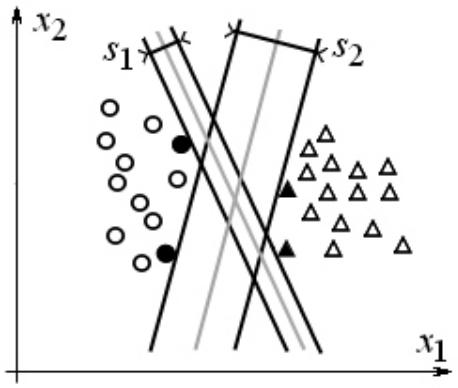


Рис. 28. Два линейно разделимых класса образов, для которых представлены две разделяющие полосы с разными ширинами: s_1 и s_2

Ширина полосы и является критерием качества, в соответствии с которым из множества возможных разделяющих поверхностей выбирается лучшая. Идея ширины полосы как критерия качества является наглядной и интуитивно привлекательной.

Среди всех гиперплоскостей, принадлежащих наиболее широкой полосе, выбирается та, расстояние до которой от опорных векторов одинаково (то есть расположенная посередине полосы). Заметим, что положение лучшей гиперплоскости определяется лишь опорными векторами; расположение других векторов обучающей выборки никак не учитывается. Эти условия позволяют определить лучшую гиперплоскость единственным образом (за исключением некоторых вырожденных случаев расположения векторов обучающей выборки).

Чтобы вычислить ширину полосы, прибегают к следующему приему. Если неравенства (29) справедливы, то всегда можно найти такую положительную константу, после умножения вектора **w** на которую будут верны следующие неравенства:

$$(\forall i) z_i \mathbf{w} \mathbf{x}_i' \geq 1, \qquad (32)$$

причем хотя бы для одного вектора $\mathbf{x}_i$ равенство будет точным. Стоит отметить, что умножение всех компонентов вектора w на одно и то же число не меняет положения разделяющей гиперплоскости.

Гиперплоскость находится в центре соответствующей ей полосы, если расстояния от опорных векторов до нее равны. Расстояние от точки до плоскости вычисляется по формуле:

$$s = \frac{\left|\mathbf{wx}'\right|}{\sqrt{w_1^2 + ... + w_N^2}}. \qquad (33)$$

Наименьшее расстояние достигается на опорных векторах, для которых $\left|\mathbf{wx}'\right| = 1$ (знак выражения $\mathbf{wx}'$ будет разным для опорных векторов разных классов). Поскольку искомая плоскость отделена таким расстоянием от обоих классов, ширина соответствующей полосы составит:

$$s_0 = 2 \Big/ \sqrt{w_1^2 + ... + w_N^2}\ . \qquad (34)$$

Вместо максимизации расстояния s_0 удобнее минимизировать обратную величину при системе ограничений (32). Эта задача решается методом неопределенных множителей Лагранжа, для чего составляется лагранжиан:

$$L(\mathbf{w}, \boldsymbol{\lambda}) = \frac{1}{2}\left(w_1^2 + ... + w_N^2\right) - \sum_{i=1}^{M} \lambda_i \left(z_i \mathbf{w}\mathbf{x}_i' - 1\right), \qquad (35)$$

который минимизируется при следующих ограничениях: $\lambda_i \geq 0, i = 1,..., M$ и $z_i \mathbf{w}\mathbf{x}_i' = 1$ для каких-либо опорных векторов. Здесь λ_i – неопределенные множители Лагранжа.

Условный экстремум лагранжиана ищется путем его дифференцирования по w_i и λ_i и приравнивания всех производных нулю. Полное решение этой задачи не слишком сложное, но достаточно громоздкое, и оно может быть найдено в справочной литературе, поэтому здесь мы его разбирать не будем. Приведем лишь результат:

$$w_k = \sum_{i=1}^{M} \lambda_i z_i x_{i,k}\ ,\ k = 1,..., N\ , \qquad (36)$$

а w_{N+1} просто получается из соотношения $z_s \mathbf{w}\mathbf{x}_s' = 1$, записанного для произвольного опорного вектора. Следует также заметить, что множители λ_i отличны от нуля, только для опорных векторов. Сами же величины λ_i находится путем минимизации квадратичной функции, получаемой путем подстановки (36) в (35), при указанных выше ограничениях на значения λ_i (задача поиска экстремума квадратичной функции при ограничениях на значения параметров относится к области квадратичного программирования).

Интересной особенностью данного метода является то, что коэффициенты оптимизируемой функции зависят только от произведений $\mathbf{x}_i \mathbf{x}_j^T$, а не от самих векторов. Более того, подставив найденные параметры w_i в уравнение решающей функции (28), можно убедиться, что и для вычисления решающей функции достаточно знать только произведения

$\mathbf{x}_i\mathbf{x}_j^T$, а не сами векторы. Это будет иметь значение при обсуждении обобщенных решающих функций, к рассмотрению которых мы и перейдем.

Обобщенные решающие функции и ядра

Не любые два набора точек в R^N разделяются гиперплоскостью, а значит, могут быть корректно классифицированы с помощью линейной решающей функции. Это является платой за простоту и вычислительную эффективность линейных методов. Нелинейные методы сложны и вычислительно трудоемки. К счастью, существует стандартный прием, позволяющий расширять процедуры построения линейных решающих функций на нелинейные функции. Этот прием заключается в том, что вводятся *обобщенные решающие функции* вида

$$\kappa(\mathbf{x}, \mathbf{w}) = w_1 f_1(\mathbf{x}) + w_2 f_2(\mathbf{x}) + ... + w_n f_n(\mathbf{x}), f_i : R^N \to R. \qquad (37)$$

В частности, несложно получить линейные решающие функции, используя $n = N + 1$ и $f_i(\mathbf{x}) = x_i'$.

Функции $f_i(\mathbf{x})$ предполагаются известными заранее, то есть имеется возможность однозначно получить их значения для любого вектора $\mathbf{x}$. Тогда решающие функции вида (37) оказываются линейными по неизвестным параметрам w_i, и несложно убедиться, что любой метод, предназначенный для нахождения параметров линейных решающих функций, также будет работать и для обобщенных решающих функций.

Один из стандартных способов задания обобщенных решающих функций – это представление их в виде многочленов (обычно используются ортонормированные системы функций, например, многочлены Лежандра или Эрмита). По сути, это означает, что для любой *непрерывной* (в некотором замкнутом интервале) решающей функции существует последовательность обобщенных решающих функций, равномерно сходящихся (на этом интервале) к ней. Это справедливо согласно теореме Вейерштрасса о приближении функций. Итак, если не ограничивать количество n слагаемых в разложении (37), то можно описать любую (непрерывную) решающую функцию. Также несложно убедиться, что, если в состав различных классов не входят идентичные векторы образов, то всегда можно найти разделяющие границы.

Интересно также отметить, что использование функций вида (37) равносильно использованию нового пространства описаний R^n с признаками $\chi_i = f_i(\mathbf{x})$, $i = 1,...,n$. Во-первых, это говорит нам, что выбор подходящих признаков может сделать классы линейно разделимыми, а задачу распознавания достаточно простой. Во-вторых, привлечение обобщенных решающих функций задачу выбора признаков не решает, так как функции $f_i(\mathbf{x})$ задаются априорно, а не строятся автоматически.

В то же время, использование в качестве новых признаков некоторой полной системы функций позволяет разделять любые классы образов, но при этом может быть порождено потенциально бесконечное число новых признаков. С точки зрения машинного обучения, этот результат вызывает определенные сомнения. Действительно, человек стремится использовать минимально необходимое число признаков, с помощью которых он описывает те или иные классы объектов. Казалось бы, аппроксимация истинной решающей функции многочленами (или другой полной системой функций) математически корректна. В чем же тут проблема? Почему возникает впечатление, что такой подход может дать плохой результат?

Чтобы ответить на этот вопрос, обратимся к рис. 29. Как видно из рис. 29а, использование лишь одного дополнительного признака $f_4(\mathbf{x}) = x_1 x_2$ приводит к неверной классификации двух признаков обучающей выборки. При использовании расширенного вектора из двадцати пяти признаков (рис. 29б) также неверно, но с меньшей ошибкой, классифицируются два образа обучающей выборки. При этом видно, что решающее правило таково, что образы имеют тенденцию располагаться вблизи границ областей.

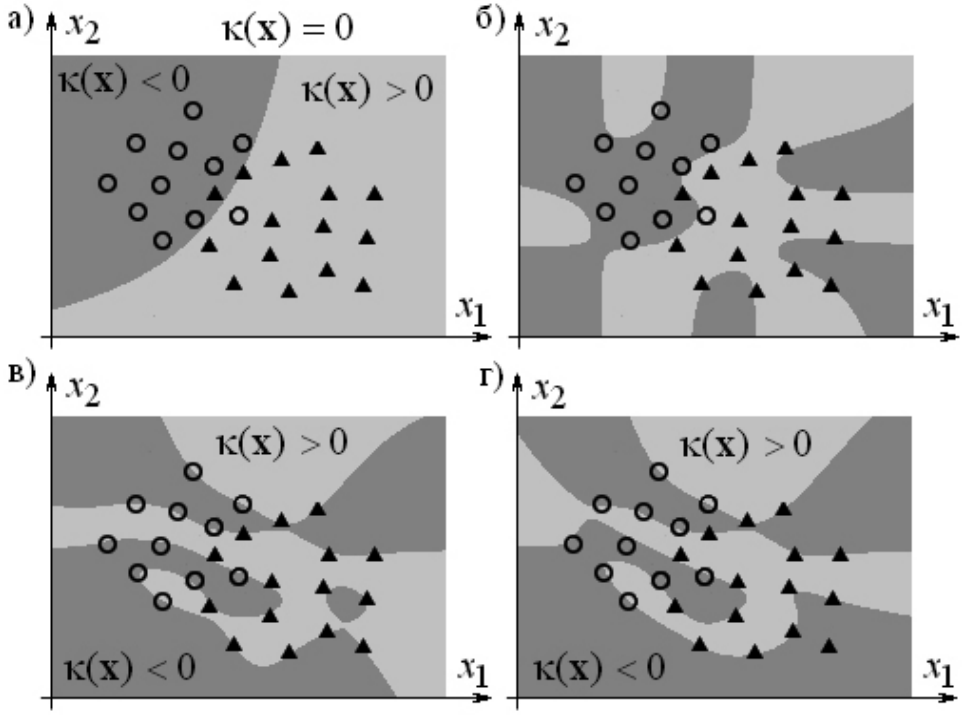


Рис. 29. Использование обобщенных решающих функций для разделения двух классов: а) на основе признаков: *1*, x_1, x_2, x_1x_2; б) на основе 25 новых признаков; в) на основе 36 новых признаков; г) на основе 36 новых признаков, но один из векторов обучающей выборки при построении решающего правила не использован

Использование тридцати шести новых признаков дает верную классификацию всех образов обучающей выборки (рис. 29в), однако построенные области интуитивно воспринимаются плохими. Чтобы

понять, в чем здесь дело, достаточно построить решающее правило по обучающей выборке, из которой исключен один из векторов. Результат такой операции представлен на рис. 29г. Исключенный образ (правый нижний треугольник на рисунке) полученным решающим правилом классифицируется неверно.

Итак, построение обобщенного вектора признаков большой размерности действительно позволяет классифицировать правильно все образы обучающей выборки, однако полученное в результате решающее правило очень плохо предсказывает классы тех образов, которые в обучающую выборку не вошли. Это типичный пример эффекта, получившего название *переобучение*, чрезмерно близкая подгонка или потеря обобщающей способности. Наша обучающая система очень хорошо «вызубрила» обучающие примеры, но не осуществила их обобщения, которое и составляет суть обучения.

Существуют различные приемы борьбы с переобучением. Во-первых, если интерпретировать функции $f_i(\mathbf{x})$ как новые признаки, то можно обратиться к методам уменьшения размерности, которые мы рассмотрим позднее. Во-вторых, можно снять ограничение на то, что все векторы обучающей выборки должны разделяться, то есть разрешить системе ошибаться на обучающих примерах. Как правило, при этом вводится эвристическая система штрафов как за подобные ошибки, так и за сложность решающей функции (за количество параметров в ней). Для рассмотренного выше примера штрафы должны быть подобраны таким образом, чтобы использование решающей функции от четырех признаков, приводящей к двум ошибкам классификации, было выгоднее, чем использование решающей функции от тридцати шести признаков, дающей верную классификацию всех образов обучающей выборки.

Однако эвристическое введение штрафов снижает привлекательность математических методов и возвращает нас к проблеме обоснованного критерия качества для выбора решающей функции. Таким образом, хотя обобщенные решающие функции существенно расширяют возможности линейных решающих функций, они вовсе не дают окончательного решения проблемы распознавания.

Поскольку в качестве примера критерия выбора разделяющей гиперплоскости был приведен метод опорных векторов, нельзя не сказать про другую возможность расширения линейных методов. Если в методе опорных векторов использовать тот же прием, что и в методе обобщенных решающих функций, и поменять все векторы $\mathbf{x}_i$ на обобщенные векторы признаков $\boldsymbol{\chi}_i = (\chi_{i,1}, \chi_{i,2}, \ldots \chi_{i,n})$, где $\chi_{i,k} = f_k(\mathbf{x}_i)$, то решающая функция будет задаваться через скалярные произведения $\boldsymbol{\chi}_i \boldsymbol{\chi}_j^T$ вместо $\mathbf{x}_i \mathbf{x}_j^T$. Пусть $F(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x}))$, тогда $\boldsymbol{\chi}_i \boldsymbol{\chi}_j^T = \langle F(\mathbf{x}_i), F(\mathbf{x}_j) \rangle$, где через $\langle \cdot, \cdot \rangle$ обозначено скалярное произведение функций с векторными значениями.

Далее введем функцию $\mathrm{K}(\mathbf{x}_i,\mathbf{x}_j)=\langle F(\mathbf{x}_i),F(\mathbf{x}_j)\rangle$, называемую *ядром*. Таким образом, в методе опорных векторов при использовании обобщенных признаков решающая функция зависит не от самих признаков $f_i(\mathbf{x})$, а только от их попарных скалярных произведений – ядер $\mathrm{K}(\mathbf{x}_i,\mathbf{x}_j)$.

Интересны они тем, что позволяют вместо признаков $F:R^N\to R^n$ с числовыми значениями использовать признаки $F:R^N\to L_2(R^N)$, значениями которых являются функции (пространство L_2 используется для того, чтобы можно было вычислить скалярное произведение двух функций $\langle F(\mathbf{x}_i),F(\mathbf{x}_j)\rangle$). Это позволяет легко добиваться линейной разделимости любых обучающих совокупностей при достаточно простых ядрах. Например, таким свойством обладает популярное гауссово ядро $\mathrm{K}(\mathbf{x}_i,\mathbf{x}_j)=e^{-|\mathbf{x}_i-\mathbf{x}_j|^2}$, которому соответствует функция-признак

$$[F(\mathbf{x})](\mathbf{y})=c\exp\left[-0.5\cdot|\mathbf{y}-2\mathbf{x}|^2\right]. \qquad (38)$$

Линейная разделимость любой (конечной) совокупности попарно различных векторов $\mathbf{x}_1,...,\mathbf{x}_M$ является следствием линейной независимости функций $\exp\left[-0.5\cdot|\mathbf{y}-2\mathbf{x}_i|^2\right], i=1,...,M$. Это можно интерпретировать так, что под каждый вектор обучающей выборки отводится собственная размерность в новом пространстве признаков (в результате количество опорных векторов может сильно возрасти).

Сама линейная разделимость в случае гауссовых ядер не очень интересна (так как имеет тривиальную интерпретацию в рамках методов, основанных на функциях расстояния), но интересной является возможность одновременного использования различные ядра.

К сожалению, хотя использование ядер имеет ряд преимуществ перед явным заданием признаков $f_i(\mathbf{x})$, здесь остается как необходимость задавать эти ядра априори, так и проблема переобучения. Причем последняя проблема становится даже более острой, так как использование ядер не позволяют применять стандартные методы уменьшения размерности. Именно поэтому здесь применяются системы штрафов за ошибки классификации на обучающих примерах и за количество опорных векторов. Однако из идеи опорных векторов корректная и обоснованная система штрафов никак не следует.

Вопросы и упражнения

1. Является ли метод обобщенных решающих функций линейным по своим параметрам, по исходным признакам?

2. Из какого соображения выбирается разделяющая поверхность в методе опорных векторов?
3. Какими методами находятся оптимальные параметры в методах обобщенных решающих функций и опорных векторов?
4. Какими преимуществами обладают методы обобщенных решающих функций и опорных векторов по сравнению с эвристическими методами распознавания образов?
5. Чем удобно использование ядер в методе опорных векторов?

12. Байесовский подход к распознаванию образов

Классификация образов в рамках статистического подхода

Выше были описаны детерминистские подходы, для которых характерно однозначное отнесение образа к одному из классов. В частности, это затрудняет прямое оценивание качества классификации. В статистическом подходе, напротив, полагается, что объект может принадлежать любому из классов, но с некоторой вероятностью. Сами же образы рассматриваются как отсчеты некоторого случайного вектора, принимающего значения из множества X и характеризующегося плотностью распределения вероятностей $p(\mathbf{x})$. Статистический подход также позволяет определять вероятность ошибочной классификации, с помощью которой и оценивается качество классификации.

Со статистической точки зрения оптимальному качеству классификации соответствует байесовский классификатор. Запишем правило Байеса для данной задачи:

$$P(a_i \mid \mathbf{x}) = \frac{P(a_i)\,p(\mathbf{x} \mid a_i)}{p(\mathbf{x})},\ a_i \in \mathrm{A}. \qquad (39)$$

Здесь через $P(a_i \mid \mathbf{x})$ обозначается апостериорная вероятность класса a_i, то есть вероятность того, что наблюдаемый образ $\mathbf{x}$ принадлежит классу a_i. Вероятность $P(a_i)$ – это априорная вероятность получения образа, принадлежащего классу a_i, а $p(\mathbf{x} \mid a_i)$ – плотность распределения вероятностей образов класса a_i или правдоподобие того, что из класса a_i будет выбран образ $\mathbf{x}$.

Рассмотрим простой пример байесовской классификации в одномерном пространстве признаков (см. рис. 30). Пусть для каждого из двух классов a_1 и a_2 есть возможность вычислить их апостериорные вероятности для каждого образа $P(a_i \mid x)$. И пусть все образы, для которых значение признака меньше некоторого порога $x < x_0$, относятся к классу a_1, а образы со значением признака $x > x_0$ – к классу a_2. Тогда величина P_1 (площадь под соответствующим графиком) характеризует вероятность ошибочного отнесения объекта второго класса к первому классу (или вероятность *ложной тревоги*, если класс a_1 отвечает понятию «цель»).

Вероятность P_2 – это вероятность ошибочного отнесения объекта первого класса ко второму классу (вероятность *пропуска цели*).

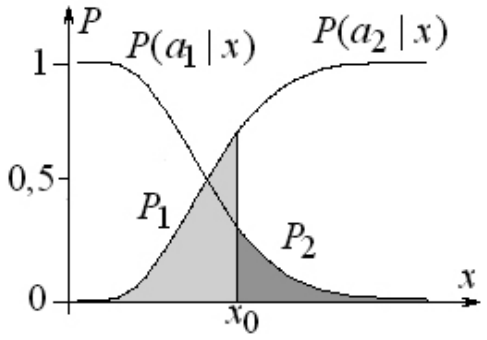


Рис. 30. Пример байесовской классификации на основе одного признака

Естественно, эти два события (пропуск цели и ложная тревога) могут быть неравнозначными. В связи с этим построение байесовского классификатора в общем случае означает минимизацию среднего риска, для чего привлекается матрица потерь **R**. Элементы матрицы потерь $r_{i,j}$ означают потери, которые возникают при классификации объекта класса a_i как объект класса a_j (обычно $a_{i,i} = 0$). Если задана матрица потерь, то в байесовском классификаторе выбирается не класс, для которого максимальна апостериорная вероятность $P(a_i \mid \mathbf{x})$, а класс, для которого минимизируются ожидаемые потери. Для a_j класса потери примут форму

$$c_j = \sum_{i=1}^{d} r_{j,i} P(a_i \mid \mathbf{x}). \qquad (40)$$

Поясним смысл матрицы потерь на следующем примере. Пусть на автоматизированном производстве выполняется обзор производственного помещения в целях обнаружения посторонних объектов в сфере действия робота. При этом требуется определить, может ли появившийся посторонний предмет повредить роботу или нет. Если может, то робот должен быть остановлен, в противном случае работу необходимо продолжить. Поскольку этот анализ должен выполняться автоматически, и невозможно заранее описать все возможные посторонние объекты, отнесение объекта к классу опасных или безопасных выполняется на основе таких общих характеристик, как, например, размеры s, скорость перемещения v, высота над уровнем пола h, извлеченных из стереоизображений помещения. Значения этих характеристик и будут вектором признаков $\mathbf{x} = (s, v, h)$, а множество классов будет A={*“опасный”, “безопасный”*}.

Величины *P*(*“опасный”*) и *P*(*“безопасный”*) – это априорные вероятности соответствующих гипотез, то есть частота опасных и безопасных посторонних объектов, появляющихся внутри данного помещения. Вероятность *P*(*s,v,h*|*“опасный”*) – это доля всех опасных объектов, имеющих размеры s, скорость перемещения v и высоту h,

аналогично, $P(s,v,h|\text{“безопасный”})$ – это доля всех безопасных объектов с такими характеристиками. Эти условные вероятности определяют правдоподобие того, что объект с такой скоростью перемещения, размерами и положением над полом может быть причиной поломки робота в случае столкновения.

Оценки как априорных вероятностей, так и величин правдоподобия могут быть получены из обучающей выборки. Пусть, например, за время испытания системы обзора производственного помещения наблюдалось сто посторонних объектов, для которых были измерены характеристики s, v, h, а человеком была оценена степень опасности этих объектов. Тогда $P(\text{“опасный”})$ – это общее число наблюдавшихся опасных объектов, деленное на сто, а $P(s,v,h|\text{“опасный”})$ – это число повстречавшихся опасных объектов со скоростью v, размером s и высотой h, отнесенное к общему числу повстречавшихся опасных объектов.

Теперь в процессе эксплуатации системы после обнаружения постороннего объекта и измерения его характеристик необходимо сравнить произведения вероятностей $P(s,v,h|\text{“опасный”})P(\text{“опасный”})$ и $P(s,v,h|\text{“безопасный”})P(\text{“безопасный”})$, которые уже не представляет сложности вычислить. Чем больше появляется опасных объектов по сравнению с безопасными, тем более вероятно, что новый объект будет являться опасным, и наоборот. Чем более характерны величины скорости перемещения, размера и положения над полом для данного класса, тем более вероятно, что обнаруженный объект относится именно к нему. Далеко не всегда можно будет однозначно заключить, к какому классу относится объект, поскольку их возможные характеристики перекрываются (к примеру, объекты с одинаковыми размерами могут представлять разную угрозу в зависимости от своей массы, которую, однако, на основе изображения объекта оценить проблематично), но правило Байеса позволяет минимизировать среднее количество ошибок.

Определение того, является ли объект опасным или нет, влияет на то, следует ли останавливать робота для избежания его поломки вследствие столкновения с опасным объектом. Остановка робота приводит к кратковременной остановке производства. Поломка же робота может привести к значительно более длительной остановке и, соответственно, к большим потерям. Видно, что разные типы ошибок имеют разную цену. В данном случае матрица потерь может иметь денежное выражение. При несимметричной матрице потерь может выбираться менее вероятное решение в целях минимизации рисков, хотя это и приводит к общему росту количества ошибок. Другой характерный случай несимметричной матрицы потерь – диагностика заболевания.

Имея матрицу потерь, не составляет труда решить проблему выбора класса, коль скоро известны апостериорные вероятности. В связи с этим дальше матрица потерь будет опускаться, а задача классификации будет рассматриваться как поиск такого класса $a_i \in \mathrm{A}$, для которого

максимальна апостериорная вероятность $P(a_i \mid \mathbf{x})$. При этом разделяющие поверхности задаются уравнениями $P(a_i \mid \mathbf{x}) = P(a_j \mid \mathbf{x})$.

Поскольку величина $p(\mathbf{x})$ в правиле Байеса не влияет на выбор класса, ее обычно не рассматривают. При классификации считается, что априорные вероятности $P(a_i)$ известны, а значения правдоподобий $p(\mathbf{x} \mid a_i)$ могут быть вычислены. Для этого они должны быть представлены в некотором удобном для вычисления виде. Одним из простейших способов, использующихся для построения байесовских классификаторов, является представление плотностей распределений условных вероятностей в виде нормального закона:

$$p(\mathbf{x} \mid a_i) = \frac{1}{(2\pi)^{N/2} \left|\mathbf{C}_i\right|^{1/2}} \exp\left[-\frac{1}{2} (\mathbf{x} - \mathbf{x}_{0,i})^T \mathbf{C}_i^{-1} (\mathbf{x} - \mathbf{x}_{0,i}) \right], \qquad (41)$$

где $\mathbf{x}_{0,i} = E_i[\mathbf{x}]$ – среднее, а $\mathbf{C}_i = E_i[(\mathbf{x} - \mathbf{x}_{0,i})(\mathbf{x} - \mathbf{x}_{0,i})^T]$ – ковариационная матрица распределения $p(\mathbf{x} \mid a_i)$, которые в задаче классификации считаются известными. Величина $\left|\mathbf{C}_i\right|$ обозначает определитель соответствующей матрицы.

Рассмотрим случай двух классов. Равенство апостериорных вероятностей даст уравнение разделяющей поверхности $p(a_1) p(\mathbf{x} \mid a_1) = p(a_2) p(\mathbf{x} \mid a_2)$. После логарифмирования получим:

$$\begin{aligned} &2 \ln p(a_1) - \ln\left|\mathbf{C}_1\right| - (\mathbf{x} - \mathbf{x}_{0,1})^T \mathbf{C}_1^{-1} (\mathbf{x} - \mathbf{x}_{0,1}) = \\ &= 2 \ln p(a_2) - \ln\left|\mathbf{C}_2\right| - (\mathbf{x} - \mathbf{x}_{0,2})^T \mathbf{C}_2^{-1} (\mathbf{x} - \mathbf{x}_{0,2}) \end{aligned} . \qquad (42)$$

Таким образом, в случае нормального распределения классы разделяются поверхностью второго порядка. Если плотности распределений действительно распределены по нормальному закону, то никакие поверхности другого вида не будут в среднем давать лучшего качества классификации. На самом деле, этот результат верен и для более общего случая: если плотности распределения $p(\mathbf{x} \mid a_1)$ и $p(\mathbf{x} \mid a_2)$ являются симметричными и монотонно убывающими от центра симметрии, то байесовская граница, разделяющая классы a_1 и a_2, является поверхностью не более чем второго порядка. Это, в свою очередь, говорит, что предположения о виде плотности распределения вероятностей являются гораздо более сильными, чем о виде разделяющих поверхностей.

Рассмотрим частный случай, когда $\mathbf{C}_1 = \mathbf{C}_2 = \sigma^2 \mathbf{E}$, где $\mathbf{E}$ – единичная матрица. Несложно убедиться, что разделяющая поверхность при этом является гиперплоскостью, а априорные вероятности $P(a_{1,2})$ определяют смещение этой гиперплоскости относительно того положения, которое было бы выбрано согласно критерию минимума расстояния. Если трактовать математические ожидания $\mathbf{x}_{0,i}$ как эталонные образцы, то в случае произвольных ковариационных матриц максимизация

апостериорной вероятности будет соответствовать минимизации расстояния Махаланобиса:

$$s(\mathbf{x}, \mathbf{y}_i) = (\mathbf{x} - \mathbf{x}_{0,i})^T \mathbf{C}_i^{-1} (\mathbf{x} - \mathbf{x}_{0,i}). \quad (43)$$

Таким образом, видна тесная связь между методами, применяющими решающие функции, основанными на минимизации расстояния и использующими правило Байеса. При этом в рамках байесовского подхода удается обосновать использование конкретных функций расстояния, что было затруднительно в рамках эвристических подходов.

Коль скоро есть возможность вычислить плотности распределения вероятностей $p(\mathbf{x} \mid a_i)$, и известны априорные вероятности $P(a_i)$, решить задачу классификации нового образа в рамках байесовского подхода не представляет сложности. Гораздо труднее является задача распознавания образов. В статистическом подходе эта задача сводится к оценке плотностей распределения условных вероятностей $p(\mathbf{x} \mid a_i)$. Существуют параметрические и непараметрические методы оценивания плотностей распределения.

Параметрические методы оценивания плотности вероятностей

В рамках статистического подхода задача распознавания образов сводится к оцениванию плотности распределения вероятностей по конечному набору испытаний. Проблема оценивания плотностей вероятностей возникает далеко не только в распознавании образов, и она относится к одной из основных проблем статистического обучения. В связи с этим ей посвящена очень обширная литература, и разработано множество методов для ее решения. Мы кратко рассмотрим лишь некоторые наиболее базовые из этих методов, тесно связанные с задачей распознавания.

Методы оценивания плотности распределения можно разделить на параметрические и непараметрические. Это деление несколько условно, поскольку часто непараметрическое оценивание тем или иным способом сводится к некоторому параметрическому методу. Разница между ними обычно заключается в том, что при сведении непараметрических методов к параметрическим количество параметров оказывается нефиксированным.

Общий метод оценивания параметров основывается на правиле Байеса (или его упрощении – методе максимального правдоподобия), но примененном на этот раз к самим плотностям распределения вероятностей. В общем случае пришлось бы рассматривать плотность вероятностей как случайную функцию и искать наиболее вероятную ее реализацию. В задаче распознавания, однако, очень часто пользуются предположением о том, что векторы обучающей выборки, принадлежащие одному классу, *статистически независимы и одинаково распределены* (iid, independently and identically distributed). Тогда им соответствует единственная плотность распределения вероятностей $p(\mathbf{x} \mid \mathbf{w})$ заданного вида, но с неизвестными параметрами $\mathbf{w}$, которые требуется оценить. Естественно также считать,

что плотности вероятностей, описывающие разные классы, независимы, и можно оценивать их параметры отдельно.

Пусть $\mathbf{x}_1,...,\mathbf{x}_M$ – векторы обучающей выборки, принадлежащие одному классу, тогда согласно теореме Байеса величина

$$p(\mathbf{w} \mid \mathbf{x}_1,...,\mathbf{x}_M) = \frac{p(\mathbf{w})\, p(\mathbf{x}_1,...,\mathbf{x}_M \mid \mathbf{w})}{p(\mathbf{x}_1,...,\mathbf{x}_M)} \tag{44}$$

является апостериорной вероятностью для вектора $\mathbf{w}$. Статистическая независимость $\mathbf{x}_i$ влечет

$$p(\mathbf{x}_1,...,\mathbf{x}_M \mid \mathbf{w}) = \prod_{i=1}^{M} p(\mathbf{x}_i \mid \mathbf{w}), \tag{45}$$

а оптимальное значение вектора параметров будет определяться как

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \left[p(\mathbf{w}) \prod_{i=1}^{M} p(\mathbf{x}_i \mid \mathbf{w}) \right]. \tag{46}$$

Удобнее работать не с самой вероятностью, а с ее логарифмом (являющимся оценкой количества информации):

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \left[-\ln p(\mathbf{w}) - \sum_{i=1}^{M} \ln p(\mathbf{x}_i \mid \mathbf{w}) \right]. \tag{47}$$

В качестве примера рассмотрим случай нормального распределения:

$$p(\mathbf{x} \mid \mathbf{C}, \mathbf{x}_0) = \frac{1}{(2\pi)^{N/2} |\mathbf{C}|^{1/2}} \exp\left[-\frac{1}{2} (\mathbf{x} - \mathbf{x}_0)^T \mathbf{C}^{-1} (\mathbf{x} - \mathbf{x}_0) \right] \tag{48}$$

и максимизацию правдоподобия (вместо апостериорной вероятности):

$$\begin{aligned} p(\mathbf{x}_1,...,\mathbf{x}_M \mid \mathbf{C}, \mathbf{x}_0) &= \prod_{i=1}^{M} p(\mathbf{x}_i \mid \mathbf{C}, \mathbf{x}_0) = \\ &= \frac{1}{(2\pi)^{MN/2} |\mathbf{C}|^{M/2}} \prod_{i=1}^{M} \exp\left[-\frac{1}{2} (\mathbf{x}_i - \mathbf{x}_0)^T \mathbf{C}^{-1} (\mathbf{x}_i - \mathbf{x}_0) \right]. \end{aligned} \tag{49}$$

Количество информации, которое нужно минимизировать, в этом случае примет вид

$$\begin{aligned} L &= -\ln p(\mathbf{x}_1,...,\mathbf{x}_M \mid \mathbf{C}, \mathbf{x}_0) = \\ &= \frac{MN}{2} \ln(2\pi) + \frac{M}{2} \ln|\mathbf{C}| + \frac{1}{2} \sum_{i=1}^{M} \left[(\mathbf{x}_i - \mathbf{x}_0)^T \mathbf{C}^{-1} (\mathbf{x}_i - \mathbf{x}_0) \right]. \end{aligned} \tag{50}$$

Приравняв нулю частные производные количества информации, взятые по параметрам плотности распределения вероятностей, можно получить систему линейных уравнений. Не так сложно убедиться, что вероятность максимальна (а величина L минимальна) при значениях параметров, задаваемых формулами (25), которые приводились для расстояния Махаланобиса. Это также интуитивно очевидно, исходя из

того, что $\mathbf{x}_0$ – это математическое ожидание, а $\mathbf{C}$ – ковариационная матрица нормального распределения.

При использовании нормальных плотностей границы между классами описываются поверхностями второго порядка. К сожалению, не для любых классов, разделимых такими сравнительно простыми поверхностями, использование моделей нормальных распределений приводит к построению правильных решающих функций (см. рис. 31), что показывает ограниченность данных моделей.

Интересно отметить следующее. В рамках статистического подхода интеграл по пространству признаков от вероятности (48) должен быть равен единице. Это накладывает определенные ограничения на матрицу $\mathbf{C}$, которые было бы сложно получить и обосновать в рамках подхода, основанного на функциях расстояния. Видно, что привлекательность статистического подхода в его строгости.

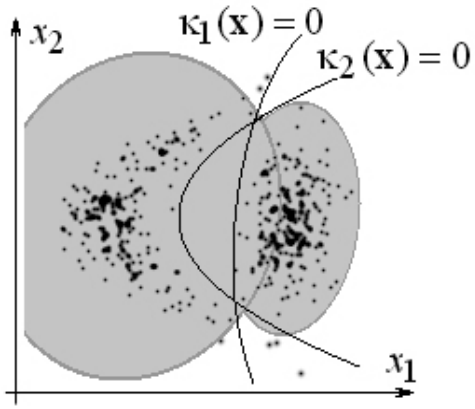


Рис. 31. Пример решения задачи распознавания для двух классов, разделимых кривой второго порядка $\kappa_2(\mathbf{x}) = 0$; использование метода максимального правдоподобия и описание распределения образов каждого из классов с помощью нормального закона приводит к построению разделяющей границы, которая также описывается кривой второго порядка $\kappa_1(\mathbf{x}) = 0$, но не является оптимальной

Здесь мы продемонстрировали лишь общую схему вывода формул для параметров распределений. Аналогичным образом можно получить уравнения и в случае байесовского подхода, при котором привлекается плотность распределения априорных вероятностей $p(\mathbf{w})$, а также для случаев использования других видов плотностей $p(\mathbf{x} \mid \mathbf{w})$.

К сожалению, в явном виде получить уравнения, по которым можно было бы вычислять параметры плотностей вероятностей, удается для весьма немногих типов плотностей. Для приближенного (например, с помощью градиентного спуска) нахождения параметров распространенным является т.н. метод *стохастической аппроксимации*. В этом методе вместо максимизации значения самой плотности вероятностей $p(\mathbf{w} \mid \mathbf{x}_1, ..., \mathbf{x}_M)$ максимизируется регрессионная функция вида

$$\rho(\mathbf{w},\mathbf{w}^*)=\int_X \xi(\mathbf{x},\mathbf{w})\,p(\mathbf{x}\,|\,\mathbf{w}^*)d\mathbf{x}, \qquad (51)$$

где $\mathbf{w}^*$ – фиксированное, но неизвестное истинное значение вектора параметров, а $\xi(\mathbf{x},\mathbf{w})$ – некоторая функция, которая должна удовлетворять определенным условиям регулярности. Хотя вектор $\mathbf{w}^*$ неизвестен, значение $p(\mathbf{x}\,|\,\mathbf{w}^*)$ может быть оценено в точках обучающей выборки. В простейшем случае эти значения принимаются равными единице в точках обучающей выборки, соответствующих данному классу, и равными нулю в остальных точках выборки. Тогда интеграл в уравнении (51) оценивается через сумму

$$\rho'(\mathbf{w})=\frac{1}{M}\sum_{i=1}^{M}\xi(\mathbf{x}_i,\mathbf{w}), \qquad (52)$$

максимум которой и будет достигаться в точке $\mathbf{w}=\mathbf{w}^*$. То есть ищется максимум величины $\rho'(\mathbf{w})$ вместо максимума $p(\mathbf{w}\,|\,\mathbf{x}_1,...,\mathbf{x}_M)$.

В качестве функции $\xi(\mathbf{x},\mathbf{w})$ может, например, использоваться $\xi(\mathbf{x},\mathbf{w})=\ln p(\mathbf{x}\,|\,\mathbf{w})$. Тогда оценка функции регрессии $\rho'(\mathbf{w})$ будет совпадать с логарифмом правдоподобия $\ln p(\mathbf{w}\,|\,\mathbf{x}_1,...,\mathbf{x}_M)$ (см. формулу (47)), то есть стохастическая аппроксимация будет являться частным случаем байесовского оценивания. Некоторое расширение байесовских методов здесь видится лишь в том, что вместо просмотра всего пространства параметров (когда решение не удается получить в явном виде, как это было в случае с гауссовым распределением), применяется один из приближенных методов нахождения экстремума регрессионной функции $\rho'(\mathbf{w})$. При этом вид функции $\xi(\mathbf{x},\mathbf{w})$ может быть подобран таким образом, чтобы эту процедуру было проще осуществлять.

Приведем еще один возможный вариант функции $\xi(\mathbf{x},\mathbf{w})$, лучше поясняющий название метода стохастической аппроксимации. Пусть $\xi(\mathbf{x},\mathbf{w})=\left|p(\mathbf{x},\mathbf{w})-p(\mathbf{x},\mathbf{w}^*)\right|^2$, а $\rho'(\mathbf{w})$ требуется минимизировать. Пусть $p_i=p(\mathbf{x}_i,\mathbf{w}^*)$ – оценки истинной плотности распределения в точках обучающей выборки (как и ранее, эти значения равны единице для образов, принадлежащих данному классу, и нулю для образов, принадлежащих другим классам). Тогда задача сводится к нахождению функции $p(\mathbf{x}\,|\,\mathbf{w})$, которая проходит ближе всего (в среднеквадратичном смысле) к точкам $(\mathbf{x}_i,p_i)$, что является классической задачей аппроксимации.

Непараметрические методы оценивания плотности вероятностей

Часто возможна такая ситуация, что никаких предположений о виде плотности распределения сделать нельзя. В этом случае используют

непараметрические методы оценивания. Однако и в этих методах все же необходимо делать некоторые априорные допущения, такие, как, например, непрерывность или симметрия плотности распределения вероятностей. Один из широко распространенных подходов к непараметрическому оцениванию заключается в представлении неизвестной плотности в виде линейной комбинации плотностей известного (параметрического) вида. Это т.н. *смеси*. Мы рассмотрим *конечные смеси*, которые представляются в виде:

$$p(\mathbf{x}) = \sum_{i=1}^{m} p(\mathbf{x} \mid \mathbf{w}_i) P(\mathbf{w}_i), \qquad (53)$$

где m – число компонентов смеси. Чтобы подчеркнуть, что величины $P(\mathbf{w}_i)$ являются численными коэффициентами, мы будем использовать обозначение $P_i = P(\mathbf{w}_i)$. Поскольку они имеют смысл вероятностей, то для них должны выполняться ограничения $0 \le P_i \le 1$ и $P_1 + ... + P_m = 1$.

Таким образом, смесь представляет собой взвешенную сумму некоторого количества различных распределений. Обычно (но вовсе не обязательно) распределения принадлежат одному и тому же параметрическому семейству и различаются лишь значениями параметров.

Поскольку нас интересует оценивание плотностей распределения вероятностей, как векторы параметров $\mathbf{w}_1,...,\mathbf{w}_m$, так и коэффициенты $P_1,...,P_m$ являются неизвестными. В связи с этим необходимо писать:

$$p(\mathbf{x} \mid \mathbf{w}_1,...,\mathbf{w}_m, P_1,...,P_m) = \sum_{i=1}^{m} P_i \cdot p(\mathbf{x} \mid \mathbf{w}_i). \qquad (54)$$

В общем случае неизвестным может быть и число компонентов смеси m.

В распознавании образов смеси актуальны еще и по следующей причине. Каждый класс a_i имеет свою модель, выражающуюся через плотность вероятностей $p(\mathbf{x} \mid a_i)$, а также вероятность $P(a_i)$ того, что произвольно взятый вектор будет принадлежать этому классу. Тогда верно

$$p(\mathbf{x}) = \sum_{i=1}^{d} p(\mathbf{x} \mid a_i) P(a_i). \qquad (55)$$

Несложно сопоставить эту формулу с формулой (53). Таким образом, наличие нескольких классов, каждый из которых имеет собственную плотность вероятностей, естественным образом порождает смесь. В отличие от оценивания параметров распределения для каждого класса в отдельности, привлечение модели смеси для работы со всеми классами одновременно позволяет получать также величины P_i, которые в данном случае являются ни чем иным, как априорными вероятностями классов, используемых в байесовском классификаторе.

Смеси полезны и как средство непараметрического оценивания плотностей распределения вероятности в отдельных классах. При этом работа со смесью может вестись абсолютно так же, как и с обычной

параметрической плотностью. В частности, здесь оказывается применимым метод стохастической аппроксимации.

Один из способов использования смеси для оценивания плотностей заключается в разложении последних по базисным функциям. Если в уравнении (53) в качестве набора функций $\{p(\mathbf{x}\,|\,\mathbf{w}_i)\}_{i=1}^{m}$ использовать полную систему функций (задаваемую, как правило, априори), то с помощью смеси можно будет аппроксимировать произвольную (непрерывную) плотность вероятностей. Это является замечательным свойством, когда априорные сведенья о виде плотности вероятностей отсутствуют. Выбор набора базисных функций, казалось бы, не накладывает никаких ограничений на то, какие плотности могут быть восстановлены, коль скоро либо этот набор является полным, либо пространство, натянутое на него, гарантированно содержит искомую плотность.

Однако хорошо прослеживается аналогия между этим подходом и методом обобщенных решающих функций. Как и в том подходе, здесь также возникает проблема переобучения, связанная с отсутствием надлежащего критерия выбора количества компонентов смеси. Одним из решений является привлечение теоретико-информационного подхода, который мы рассмотрим несколько позднее.

Одной из наиболее популярных смесей является смесь нормальных плотностей, получающаяся подстановкой нормального распределения (48) в смесь (53) вместо плотностей $p(\mathbf{x}\,|\,\mathbf{w}_i)$ с различными ковариационными матрицами и векторами средних. Эта смесь имеет тесную связь с методами, базирующимися на функциях расстояния. Действительно, как мы убедились ранее, максимизация плотности вероятностей в случае нормального распределения соответствует минимизации расстояния Махаланобиса. Значит, каждый компонент смеси задает положение некоторого эталонного образа с локально оцененной метрикой. Однако проблема выбора количества эталонных образов (или компонентов смеси) остается и в чистом статистическом подходе.

В частном случае для аппроксимации плотности распределения элементов одного класса можно жестко задать параметры смеси следующим образом. Количество m компонентов смеси равно числу эталонных образов M. Ковариационные матрицы всех компонентов являются единичными матрицами. Вектор средних $\mathbf{x}_{0,i}$ i-го компонента смеси равен i-му образу обучающей выборки $\mathbf{x}_{0.i} = \mathbf{x}_i$, а коэффициенты смеси $P_i = 1/M$. Иными словами, в каждую точку обучающей выборки «помещается» нормальное распределение с единичной ковариационной матрицей. Если таким образом описать плотность вероятностей каждого класса, то получим 1-БС правило. Но в отличие от этого правила, в статистическом подходе можно отказаться от использования единичных

ковариационных матриц, тогда появится возможность производить локальное оценивание метрики пространства признаков.

Из локального оценивания плотности вероятностей можно также получить и *k*-БС правило. Основная идея здесь заключается в том, что для некоторой точки $\mathbf{x}$, в которой производится оценивание плотности, находится некоторая область, например, шар $B_r(\mathbf{x})$ радиуса r с центром в данной точке, содержащая помимо точки $\mathbf{x}$ также k элементов обучающей выборки. Откуда можно определить локальную плотность распределения точек, а, произведя соответствующую нормировку, получить и оценку плотности вероятностей. Строгое развитие этой простой идеи приводит как к получению формальных ограничений на вид плотностей вероятностей, которые могут быть оценены таким способом (эти ограничения можно перенести на метод *k*-БС, в рамках которого их получить затруднительно), так и к построению более эффективных методов. В качестве примера такого метода можно назвать метод окон Парзена и другие методы сглаживания, использующие вместо выделения области с четкими границами сглаживание с некоторым окном или весовой функцией (также называемой ядром).

Непараметрические методы оценивания плотностей вероятностей на основе конечных смесей находят применение в задаче распознавания без учителя, поэтому более детальный разбор этих методов будет сделан после постановки этой задачи.

Вопросы и упражнения

1. Какому методу соответствует метод распознавания на основе правила Байеса при описании плотностей распределения образов внутри классов гауссианами?
2. В каких целях используются конечные смеси?
3. Какому методу соответствует Байесовский метод распознавания на основе смесей нормальных плотностей распределения?
4. Для каких целей используется стохастическая аппроксимация?

13. Методы кластеризации

В задаче распознавания без учителя машинной системе предоставляется лишь совокупность образов $\mathbf{x}_i \in \mathrm{X}$, i=1,…,M. На основе этих образов система должна не только построить решающее правило $\varphi : \mathrm{X} \to \mathrm{A}$, но и сформировать само множество классов $\mathrm{A} = \{a_1, a_2, ..., a_N\}$.

Поскольку решающее правило относит каждый образ обучающей выборки к одному из классов, задача, по сути, сводится к тому, чтобы объединить образы обучающей выборки в группы (на основе которых и формируются классы). Такое объединение называется *группированием*. Коль скоро группирование осуществлено, для построения решающего

правила могут быть применены ранее рассмотренные методы распознавания с учителем. Здесь, однако, возникает вопрос: на каком основании какие-то образы следует относить к одной группе, а какие-то – к другой?

Один из интуитивно очевидных ответов на этот вопрос заключается в том, что объединяться должны похожие друг на друга образы. Степень сходства определяется расстоянием в пространстве признаков. Выбор метрики, однако, во многом произволен, хотя чаще всего используют евклидово расстояние. Если в классы объединяются наиболее близко расположенные друг к другу образы, то задача группирования превращается в задачу *кластеризации*, то есть в задачу поиска кластеров (областей, содержащих компактно расположенные группы образов).

С использованием расстояния как степени сходства образов мы уже встречались при обсуждении эвристических методов распознавания. В задаче кластеризации эта эвристика также является одной из базовых.

Мы кратко рассмотрим несколько эвристических алгоритмов кластеризации. Один такой алгоритм основан на вычислении k внутригрупповых средних, или кратко *алгоритм k средних*. Этот алгоритм требует задания числа кластеров, исторически обозначаемых через k. Мы, однако, как и выше, будем использовать переменную d для обозначения числа классов и $\mathrm{A} = \{a_1,...,a_d\}$ для обозначения множества классов. Алгоритм состоит из следующих шагов:

1. Каждому из d кластеров произвольным образом назначаются их центры (или эталонные образы) $\mathbf{x}_{0,i}$. Часто в качестве этих центров выступают первые (или случайно выбранные) d образов обучающей выборки $\mathbf{x}_{0,i} = \mathbf{x}_i, i = 1,...,d$.
2. Каждый образ выборки относится к тому классу, расстояние до центра которого минимально:
$$A_i = \arg\min_{a_j \in \mathrm{A}} \left(s(\mathbf{x}_i, \mathbf{x}_{0,j})\right), i = 1,...,M . \qquad (56)$$
3. Центры кластеров пересчитываются, исходя из того, какие образы к каждому из них были отнесены: $\mathbf{x}_{0,j} = \frac{1}{M_j} \sum_{(\forall i) A_i = a_j} \mathbf{x}_i$, где M_j – количество образов, попавших в класс a_j.
4. Шаги 2 и 3 повторяются, пока не будет достигнута сходимость, то есть пока классы не перестанут изменяться.

Существуют различные модификации алгоритма k средних, например, может быть использовано алгоритм, в котором евклидово расстояние является взвешенным по каждой координате. Аналогично можно использовать и расстояние Махаланобиса, вычисляемое для всех классов на каждой итерации (поскольку принадлежность образов, составляющих

классы, меняется на каждой итерации, меняются и ковариационные матрицы, описывающие форму каждого из классов).

Хотя алгоритм k средних основывается на функции расстояния, вычисляющейся для пар образов, несложно заметить, что он минимизирует глобальную меру – суммарное среднеквадратичное отклонение образов от центров своих классов.

Одним из ограничений алгоритма k средних является необходимость задания числа классов. Чтобы решить эту проблему, можно выполнить алгоритм для различного числа классов и выбрать среди решений некоторое лучшее решение. При выборе по критерию минимальной суммы расстояний от образов до центров классов предпочтение будет отдаваться решениям с большим числом классов, поэтому требуется применять более сложные критерии.

Существуют и алгоритмы, не требующие задания числа классов. Однако во многих из них присутствует какой-либо иной параметр. Простейший такой алгоритм требует задания порога s^* на размер кластера и состоит из шагов:

1. Сформировать один кластер ($d = 1$) из первого образа обучающей выборки и положить $\mathbf{x}_{0,1} = \mathbf{x}_1$.
2. Выбрать следующий еще нерассмотренный вектор $\mathbf{x}_i$ и определить минимальное расстояние $s' = \min\limits_{j=1,\ldots,d} s(\mathbf{x}_i, \mathbf{x}_{0,j})$. Если это расстояние меньше порога $s' < s^*$, то отнести образ к соответствующему классу, в противном случае увеличить число классов d на единицу и сформировать новый класс с центров $\mathbf{x}_{0,d} = \mathbf{x}_i$.
3. Повторить шаг 2 последовательно для всех образов обучающей выборки.

Этот алгоритм требует лишь однократного отнесения каждого образа к некоторому классу и является очень эффективным в вычислительном плане, но при этом его результат, помимо всего прочего, сильно зависит от порядка предъявления образов.

Хотя для работы этого алгоритма не требуется знание числа классов, ему необходим размер кластеров s^*, причем этот размер одинаков для всех классов. Эти два параметра (число классов и размер кластеров) несут примерно одинаковое количество, хотя и немного разной, априорной информации. В связи с этим нельзя сказать, что этот подход предпочтительнее предыдущего. Если он применяется на практике, и порог s^* неизвестен, то алгоритм желательно выполнять для различных значений порога и использовать некоторый критерий для выбора лучшего решения.

Этот алгоритм, правда, представляет некоторый интерес как простейший пример инкрементного обучения. Действительно, образы в нем предъявляются последовательно, и его работу можно

интерпретировать следующим образом: если новый встреченный объект не похож ни на один уже известный класс объектов, то это объект нового, ранее не встречавшегося класса. Если же объект может быть надежно классифицирован, то он пополняет информацию о выбранном классе: на его основе уточняется модель класса (в данном случае смещается вектор средних).

Существуют алгоритмы, использующие другие стратегии построения кластеров. Например, в алгоритме *максиминного* (максимально-минимального) *расстояния* сначала выделяются наиболее удаленные (наиболее различные) образы, служащие прототипами классов. Таким образом, помимо максимизации сходства объектов, относящихся к одним и тем же классам, максимизируется и различие объектов, принадлежащих разным классам.

Алгоритм *ISODATA* (Iterative Self-Organizing Data Analysis Techniques) основывается на алгоритме *k* средних, но включает набор оказавшихся полезными на практике эвристик и параметры по их настройке. Одним из задаваемых априори параметров является желаемое число кластеров *K*. Это число выступает в качестве рекомендации: в результате работы алгоритма может быть построено как меньшее, так и большее число кластеров, но оно будет не сильно отличаться от значения *K*. Сам алгоритм здесь детально описываться не будет (в целом, в нем используются те же шаги, что и в алгоритме *k* средних); приведем лишь основные эвристики.

1. Ликвидируются кластеры, в состав которых входит менее чем заданное число элементов.
2. Для каждого текущего кластера определяется направление максимальной вытянутости. Наиболее вытянутый кластер может быть расщеплен на два. Решение о расщеплении принимается с учетом размера кластера в направлении вытянутости (этот размер может сравниваться с фиксированным порогом и отклонением от среднего размера всех кластеров, а также общего числа кластеров, которое должно быть мало (с учетом параметра *K*).
3. Попарно сливаются кластеры, расстояние между центрами которых меньше заданного порога, если число кластеров велико (с учетом параметра *K*).

Хотя эти эвристики поиска опираются на функцию расстояния, они могут быть применены для произвольных алгоритмов группирования. Однако для этого потребуется введение более обоснованных критериев, по которым можно было бы осуществлять удаление и создание кластеров, а также их слияние и расщепление.

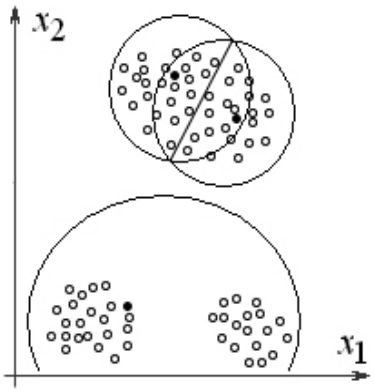


Рис. 32. Пример негативного влияния на результат кластеризации неудачного выбора начальных центров (они отмечены заполненными кружками) в методе *k* средних; слияние двух верхних кластеров и расщепление нижнего кластера, осуществляющиеся алгоритмом *ISODATA* при удачно настроенных параметрах, позволяют получить верное решение

Перечисленные методы кластеризации преимущественно являются итерационными, что позволяет не перебирать все возможные варианты группирования, но делает эти методы чувствительными к начальной гипотезе о положении центров кластеров либо к порядку предъявления векторов обучающей выборки. Использующиеся в алгоритме *ISODATA* эвристики помогают не только подбирать более подходящее число классов, но и находить более приемлемое решение (см. рис. 32), несколько ослабляя (но не убирая полностью) зависимость от начальной гипотезы.

Другой проблемой алгоритмов кластеризации является то, что априорное задание меры расстояния сильно ограничивает их возможности. Например, при использовании евклидового расстояния строящиеся классы всегда будут линейно разделимыми, а понятия, соответствующие этим классам, то есть те понятия, которые может обнаружить система машинного обучения, достаточно простыми.

Проблема кластеризации может решаться и в рамках байесовского подхода. Если для всех векторов обучающей выборки плотность распределения вероятностей оценивается в форме смеси, то каждый компонент смеси можно трактовать как кластер в пространстве признаков:

$$p(\mathbf{x} \mid \mathbf{w}_1,...,\mathbf{w}_m, P_1,...,P_m) = \sum_{i=1}^{m} P_i\, p(\mathbf{x} \mid \mathbf{w}_i)\,. \qquad (57)$$

Если плотности $p(\mathbf{x} \mid \mathbf{w}_i)$ являются локализованными в пространстве, то уравнения $p(\mathbf{x} \mid \mathbf{w}_i) = p(\mathbf{x} \mid \mathbf{w}_j)$ будут описывать границы между классами. Удобными в качестве компонентов смеси являются гауссовы распределения. В этом случае каждый класс будет описываться эллипсоидом. Сама же процедура кластеризации будет похожа на модифицированный метод *k* средних, но использование байесовского подхода позволяет математически более строго описать проблему кластеризации.

Для упрощения записи введем обозначения для гипотезы кластеризации $h = (\mathbf{w}_1,...,\mathbf{w}_m, P_1,...,P_m)$ и для исходных данных $D = (\mathbf{x}_1,...,\mathbf{x}_M)$. Запишем правило Байеса:

$$P(h \mid D) = \frac{P(h)P(D \mid h)}{P(D)}. \qquad (58)$$

Следует выбрать такую гипотезу h, апостериорная вероятность которой $P(h \mid D)$ максимальна. Величина $P(D)$ не зависит от гипотезы h, поэтому ее можно не учитывать. Далее имеем

$$P(h)P(D \mid h) = P(h)P(\mathbf{x}_1,...,\mathbf{x}_M \mid h) = P(h)\prod_{i=1}^{M} P(\mathbf{x}_i \mid h). \qquad (59)$$

Последняя часть равенства возникает в рамках предположения, что образы обучающей выборки получены независимо (хотя на практике это предположение может нарушаться). Тогда имеем

$$h_{\max} = \arg\max_h \left[P(h)\prod_{i=1}^{M} p(\mathbf{x}_i \mid h) \right]. \qquad (60)$$

Вместо максимизации данного произведения можно минимизировать минус логарифм от него:

$$h_{\max} = \arg\min_h \left[-\log_2 P(h) + L(D \mid h) \right], \qquad (61)$$

где $L(D \mid h) = -\sum_{i=1}^{M} \log_2 p(\mathbf{x}_i \mid h)$ – минус логарифм правдоподобия.

Оценивание величины априорных вероятностей в задаче кластеризации для байесовского подхода является проблематичным, поэтому часто ограничиваются методом максимального правдоподобия, в котором рассматривается только слагаемое $L(D \mid h)$.

Алгоритм ожидания-максимизации

Для поиска гипотезы, обладающей максимальной апостериорной вероятностью, может использоваться *алгоритм ожидания-максимизации* (ОМ).

Алгоритм ОМ широко применяется для одновременного нахождения параметров модели и недостающих данных. *Задачу с недостающими данными* можно сформулировать следующим образом. Пусть $p(D, D' \mid \mathbf{w})$ – некоторая стохастическая модель, причем D – известные данные, D' – недостающие данные, $\mathbf{w}$ – параметры модели, которые необходимо оценить. Если бы данные D' были также известны, то задача свелась к поиску таких значений параметров $\mathbf{w}$, которые бы максимизировали правдоподобие $p(D, D' \mid \mathbf{w})$ или, в общем случае, апостериорную

вероятность $p(\mathbf{w} \mid D, D')$. Поскольку эти данные неизвестны, их также необходимо определить. Алгоритм ОМ решает эту задачу, используя некоторое исходное предположение о значении вектора параметров и итерационно выполняя два шага.

1. Шаг ожидания. Руководствуясь стохастической моделью при текущем значении параметров, оценить недостающие данные.
2. Шаг максимизации. Используя исходные данные и текущую оценку недостающих данных, получить новую оценку параметров модели, максимизирующих правдоподобие всей совокупности данных.

Подробнее описывать алгоритм ОМ в общем виде мы не будем, а поясним его работу на задаче группирования (которая является лишь одним из частных случаев задачи с недостающими данными).

Рассмотрим смесь нормальных плотностей для простого одномерного случая.

$$p(x \mid \mathbf{w}) = \sum_{i=1}^{d} P_i p(x \mid x_{0,i}, \sigma_i) = \sum_{i=1}^{d} \frac{P_i}{\sqrt{2\pi}\sigma_i} \exp\left[-\frac{(x - x_{0,i})^2}{2\sigma_i^2} \right]. \qquad (62)$$

Пусть величина

$$P_{i,j} = \frac{P_j p(x_i \mid x_{0,j}, \sigma_j)}{\sum_{l=1}^{d} P_l p(x_i \mid x_{0,l}, \sigma_l)}, i = 1,..., M \qquad (63)$$

является вероятностью того, что i-й образ обучающей выборки принадлежит j-му классу. Эти вероятности являются недостающими данными в задаче группирования, и их можно вычислить, зная параметры смеси. Если же вероятности $P_{i,j}$ известны, то несложно найти параметры смеси, максимизирующие правдоподобие:

$$P_j = \frac{1}{M} \sum_{i=1}^{M} P_{i,j}, \qquad (64)$$

$$x_{0,j} = \frac{1}{MP_j} \sum_{i=1}^{M} P_{i,j} x_i, \qquad (65)$$

$$\sigma_j^2 = \frac{1}{MP_j} \sum_{i=1}^{M} P_{i,j} \left(x_i - x_{0,j} \right)^2. \qquad (66)$$

Шаг ожидания алгоритма ОМ состоит в вычислении вероятностей (63), а шаг максимизации – в вычислении параметров смеси (64–66). Итеративное выполнение этих шагов позволяет добиться максимизации правдоподобия. Обычно алгоритм останавливается, когда правдоподобие (или его логарифм) начинает мало меняться на шаге максимизации, или выполнено большое число итераций. Алгоритм несложно обобщить на

многомерный случай; одномерный случай приведен лишь для большей наглядности.

Алгоритм ОМ требует задания начальной гипотезы. Если первым выполняется шаг ожидания, то требуется задать какие-либо значения параметров смеси. Если же сначала выполняется шаг максимизации, то необходимы значения вероятностей $P_{i,j}$. Эти значения обычно задаются случайным образом. В зависимости от начальной гипотезы алгоритм может сойтись к различным решениям, поэтому иногда он выполняется несколько раз и из полученных решений выбирается лучшее.

Несложно заметить, что действие алгоритма ОМ для случая смеси нормальных плотностей очень похоже на работу метода k средних. Действительно, выбираются некоторые начальные положения и размеры кластеров, а затем они итерационно подправляются, исходя из того, какой вектор попал в какой кластер. Здесь, однако, векторы не жестко относятся к единственному классу, а принадлежат всем классам с некоторой вероятностью, определяемой на основе нормальной смеси. При оценивании нормального распределения также оценивается и размер кластера (а в многомерном случае ковариационная матрица, то есть расстояние Махаланобиса). Этот алгоритм является более строгим со статистической точки зрения, но его суть та же, что и у метода k средних.

Хотя статистический подход кажется математически корректным, однако и он не полностью решает проблему критерия качества решения задачи кластеризации. В этом несложно убедиться, если предположить, что в смеси количество компонентов неизвестно. Очевидно, смесь с большим числом компонентов будет обладать большим правдоподобием, в частности, правдоподобие смеси с числом компонентов, равным числу образов выборки, будет равно единице.

Принцип минимальной длины описания

Проблема выбора числа кластеров в задаче группирования очень важна, поскольку предположение о том, что число кластеров известно, является очень сильным ограничением на класс решаемых задач. В то же время во многих методах эта проблема решается чисто эвристически и, зачастую, не вполне корректно, так что методы оказываются склонными либо формировать чрезмерно большое, либо слишком малое число кластеров по сравнению с оптимальным их количеством. Мы уже неоднократно сталкивались с проблемой критерия качества в задачах обучения с учителем, но при обучении без учителя эта проблема встает наиболее остро.

Наиболее многообещающим инструментом решения этой проблемы является *принцип минимальной длины описания* (*МДО*). Этот принцип имеет глубокое теоретическое обоснование в рамках алгоритмической теории информации А.Н. Колмогорова, а также имеет ряд расширений и уточнений. Мы же ограничимся рассмотрением его не вполне

формального практического применения. В своей нестрогой, словесной, формулировке этот принцип гласит: среди всех моделей, объясняющей имеющиеся данные, нужно выбрать ту, которая позволяет минимизировать сумму:

- длины описания (в битах) данных в рамках модели;
- длины описания (в битах) самой модели.

В связи с этой формулировкой принцип МДО часто называют теоретико-информационной формализацией принципа «бритва Оккама», который гласит: «То, что можно объяснить посредством меньшего, не следует выражать посредством большего», – или более кратко: «сущностей не следует умножать без необходимости». Бритва Оккама часто использовалась как один из принципов научной методологии, позволявший исключать безосновательные объяснения природных феноменов. Принцип же МДО говорит, что следует искать компромисс между сложностью модели и ее точностью, при этом и то, и другое может быть выражено в терминах количества информации.

Посмотрим, как функционирует принцип МДО, на примере того, как астрономами строились модели движения планет.

- При очень грубых и нерегулярных наблюдениях, вероятно, полагалось, что планеты (практически неотличимые невооруженным глазом от звезд – лишь по отсутствию мерцания) движутся по окружности вокруг Земли. Здесь модель движения некоторой планеты является очень простой – это окружность фиксированного радиуса.
- При появлении способа определения координат на небесной сфере было замечено, что для некоторых объектов движение не является круговым, и Птолемей ввел систему эпициклов для описания некругового движения. Каждая планета теперь вращалась вокруг некоторого центра, который в свою очередь вращался вокруг Земли. Все отклонения от этой модели относились на счет неточности наблюдений (в рамках принципа МДО эти отклонения входили в описание данных в рамках модели). Сложность модели возросла, так как уже было нужно описывать параметры нескольких окружностей-эпициклов.
- При развитии техники наблюдения были обнаружены регулярные (не случайные) отклонения от этой модели, что привело к введению дополнительных эпициклов. То есть движение планеты рассматривалось как движение по окружности, центр которой движется по другой окружности, центр которой движется по третьей окружности и т.д. По сути, это разложение в ряд гармонических функций. Если взять достаточное количество членов, то можно описать любое периодичное движение сколь угодно точно. При этом, однако, сложность модели существенно возрастает, так как приходится описывать параметры большого числа окружностей.

- При дальнейшем развитии техники (и точности наблюдений) число эпициклов продолжило расти, что послужило одним из стимулов развития представления, согласно которому все небесные тела (и Земля в том числе) вращаются вокруг Солнца по эллиптическим орбитам. Это привело сначала к созданию гелиоцентрической системы мира Коперника, а потом и к выводу уравнений Кеплера. Движение планеты описывается лишь одним эллипсом, то есть эта модель движения обладает значительно меньшей сложностью, чем большая совокупность эпициклов, поэтому с точки зрения принципа МДО является гораздо более предпочтительной.

Иногда утверждается, что выбор между геоцентрической системой мира Птолемея и гелиоцентрической системой Коперника – это не более чем вопрос удобства, а движение планет с их помощью описывается одинаково хорошо. В действительности же это не так: они одинаково хорошо описывают *имеющиеся* данные, но точность предсказания у первой существенно хуже, чем у второй: при усовершенствовании точности наблюдений оказывается необходимым вводить все новые эпициклы, параметры которых абсолютно неизвестны заранее; параметры же эллипсов при этом меняются весьма мало. Несложно увидеть, что большое число эпициклов соответствует неоднократно упоминавшейся проблеме переобучения.

Посмотрим теперь, как принцип МДО может использоваться для решения проблемы выбора числа классов в задаче кластеризации. Исходными данными в этой задаче является обучающая выборка, состоящая из совокупности образов $\mathbf{x}_i \in \mathrm{X}$, i=1,…,M.

Рассмотрим сначала, как выбор числа кластеров может быть осуществлен на основе принципа МДО в методе k средних. Пусть на некотором шаге этого алгоритма получены центры кластеров: $\mathbf{x}_{0,i}, i = 1,...,d$. Эти центры кластеров, по сути, являются параметрами модели данных. Общее число параметров в такой модели равно Nd.

Кодирование параметров модели обычно выполняется путем дискретизации пространства параметров с некоторой точностью. Известно, что для регулярных параметрических семейств оптимальной является точность $1/\sqrt{M}$, где M – количество элементов в выборке. Объяснение этой точности на интуитивном уровне сводится к тому, что величина $1/\sqrt{M}$ представляет собой величину ожидаемой ошибки в оценке каждого из элементов вектора параметров, поэтому кодировать эти параметры с большей точностью не имеет смысла. Помимо этого в описание модели также должно входить значение числа кластеров d, для чего потребуется примерно $\log_2 d$ бит. Таким образом, общая длина описания модели может быть оценена как

$$L_m = Nd\left(-\log_2(1/\sqrt{M})\right) + \log_2 d = \frac{Nd}{2}\log_2 M + \log_2 d\,. \qquad (67)$$

Описанием каждого образа $\mathbf{x}_i$ обучающей выборки в рамках данной модели будет:

- номер n_i ближайшего эталона (центра какого-либо класса): $n_i = \arg\min_n \left|\mathbf{x}_i - \mathbf{x}_{0,n}\right|$, для указания которого требуется $\log_2 d$ бит.
- закодированный в некотором виде вектор разности $\mathbf{x}_i - \mathbf{x}_{0,n_i}$. Длина описания этой разности может быть оценена как $\log_2 \left|\mathbf{x}_i - \mathbf{x}_{0,n_i}\right| + C$, где C – некоторая константа, которая может не учитываться в конечном критерии. Смысл этой константы в том, что она указывает точность кодирования векторов. Если разность $\mathbf{x}_i - \mathbf{x}_{0,n_i}$ меньше точности кодирования, то длина описания этой разности полагается равной нулю (если этот момент не учитывать, то при стремлении разности к нулю, длина описания будет стремиться к минус бесконечности, что лишено смысла).

Таким образом, длина описания данных в рамках модели может быть оценена как

$$L_d = \sum_{i=1}^{M} \left(\log_2 \left|\mathbf{x}_i - \mathbf{x}_{0,n_i}\right| + \log_2 d\right). \qquad (68)$$

Итоговый критерий на основе принципа МДО будет

$$L = L_m + L_d = \frac{Nd}{2}\log_2 M + (M+1)\log_2 d + \sum_{i=1}^{M} \log_2 \left|\mathbf{x}_i - \mathbf{x}_{0,n_i}\right|. \qquad (69)$$

При фиксированном значении d поиск центров кластеров может производиться обычным методом k средних. Если же этот метод выполнить при разных значениях d, то выбор между полученными результатами можно осуществить на основе критерия МДО (69).

Благодаря использованию критерия МДО для выбора числа кластеров решается проблема переобучения. Действительно, если увеличивать число кластеров, то не только будет уменьшаться длина описания данных L_d, определяемая средним расстоянием от образов выборки до центров кластеров, но также будет увеличиваться и длина описания модели L_m.

Сходным образом решается проблема выбора числа кластеров и для Байесовского подхода на основе нормальных смесей. В этом методе моделью является смесь

$$p(\mathbf{x} \mid \mathbf{w}_1,...,\mathbf{w}_d, P_1,...,P_d) = \sum_{i=1}^{d} P_i\, p(\mathbf{x} \mid \mathbf{w}_i), \qquad (70)$$

где $\mathbf{w}_i$ – вектор параметров i-го компонента смеси. Если $p(\mathbf{x} \mid \mathbf{w}_i)$ – нормальное распределение, то в качестве его параметров выступает N-мерный вектор средних и ковариационная матрица размера NxN. Ковариационная матрица является симметричной, поэтому в ней $\frac{N(N+1)}{2}$

свободных параметров. Таким образом, один компонент смести описывается $\frac{N(N+1)}{2}+N$ параметрами. Помимо этого, необходимо описать значения P_i, среди которых d–1 свободных параметров (так как их сумма равна единице). Таким образом, длина описания модели составит

$$L_m = \frac{1}{2}\left(\frac{N(N+1)}{2}d + Nd + d - 1\right)\log_2 M . \qquad (71)$$

Описание некоторого вектора в рамках этой модели будет состоять из двух компонентов:

- номера класса n_i, к которому относится этот вектор; длина описания этой составляющей может быть оценена так же, как и ранее;
- и описания самого вектора в рамках стохастической модели $p(\mathbf{x} \mid \mathbf{w}_{n_i})$; длина этого описания оценивается как $-\log_2 p(\mathbf{x}_i \mid \mathbf{w}_{n_i})$.

Отсюда получаем длину описания данных в рамках модели:

$$L_d = -\sum_{i=1}^{M} \log_2 p(\mathbf{x}_i \mid \mathbf{w}_{n_i}) + M \log_2 d . \qquad (72)$$

Как и в предыдущем случае, при фиксированном числе кластеров параметры смеси могут быть оценены на основе базового метода, в данном случае, метода нормальных смесей. Далее на основе критерия МДО может быть произведено сравнение решений, полученных для разных значений d, и из них выбрано лучшее.

Следует отметить, что приведенные оценки длин описания могут быть уточнены. К примеру, номера классов, к которым относятся разные образы, могут кодироваться с учетом частоты их появления.

Несложно убедиться, что принцип МДО позволяет решать проблему построения критерия качества обучения не только в задаче кластеризации, но также и в задаче распознавания. К примеру, в методе обобщенных решающих функций использование дополнительных признаков будет очевидным образом увеличивать длину описания модели, что не всегда будет компенсироваться уменьшением длины описания данных.

Таким образом, принцип МДО позволяет вводить корректные критерии качества обучения, которые позволяют избегать переобучения или, наоборот, чрезмерного обобщения.

Вопросы и упражнения

1. Классы какой формы строятся методом k внутригрупповых средних?
2. Какие основные эвристики используются в алгоритме ISODATA?
3. Какие задачи решаются с помощью алгоритма ожидания-максимизации?
4. В чем заключается принцип минимальной длины описания?

5. Какая основная проблема, возникающая в других методах кластеризации, решается с помощью принципа МДО?

14. Выбор признаков

В задачах распознавания образов и кластеризации пространство признаков $\mathrm{X} = R^N$ предполагалось заданным. В этом исходном пространстве и осуществлялось построение разделяющих поверхностей, или очерчивались границы классов. В области распознавания образов существует еще одна часто встречающаяся задача – выбор признаков.

В задаче выбора признаков также известно исходное пространство X, соответствующее тому, какая информация об объекте доступна. Новые признаки могут быть получены только на основе имеющейся об объекте информации. Но если эти признаки не несут новой информации, то в чем же заключается задача выбора признаков?

Рассмотрим в качестве примера проблему распознавания изображений. Изображение приходит в компьютер в виде совокупности яркостей пикселей. Этот массив яркостей можно трактовать как вектор признаков. Однако одно и то же изображение, лишь слегка сдвинутое, будет характеризоваться абсолютно новым вектором признаков, каждый компонент которого будет отличаться от соответствующего признака несмещенного изображения. Хотя описания изображений в виде массива яркостей и содержать всю имеющуюся информацию об изображении, но классы образов, соответствующие изображениям одного и того же объекта, образуют в этом пространстве области слишком сложной формы и не могут быть распознаны рассмотренными выше методами.

Таким образом, задача выбора признаков заключается в том, чтобы найти такие признаки, значения которых мало меняются для объектов одного класса, но сильно меняются для объектов разных классов. Часто оказывается, что для надежного распознавания достаточно лишь небольшого числа таких признаков, так что выбор признаков может также приводить и к снижению размерности признакового пространства.

Задача выбора признаков может возникать как при обучении с учителем, так и при обучении без учителя.

Преобразование кластеризации при обучении с учителем

Рассмотрим проблему выбора признаков в случае обучения с учителем. Как и в задаче распознавания, здесь в качестве исходных данных выступает набор векторов $\{\mathbf{x}_1,...,\mathbf{x}_M\}, \mathbf{x}_i \in \mathrm{X}$, для которых известна принадлежность классам: i-й вектор принадлежит $A_i \in \mathrm{A}$ классу. Как и ранее, обозначим через $\mathbf{x}_1^{(k)},...,\mathbf{x}_{M_k}^{(k)}$ образы, принадлежащие k-му классу, где M_i – число образов в i-м классе, причем $M_1 + ... + M_d = M$. Прежде чем

переходить к формальным методам выбора признаков, попробуем сначала понять, в чем заключается смысл самой задачи.

В интуитивном понимании образы, принадлежащие одному классу, должны быть похожи между собой или должны иметь близкие значения некоторых признаков. Однако могут присутствовать и признаки, которые сильно различаются для объектов данного класса. Например, разный цвет двух автомобилей не мешает им быть одной модели или, тем более, являться автомобилями. Таким образом, различные признаки могут в разной степени характеризовать тот или иной класс образов. Наилучшими будут инвариантные признаки, то есть признаки, одинаковые для всех образов класса. Из этих соображений задачу выбора признаков можно трактовать как задачу выбора инвариантных признаков. Однако для признаков с вещественными значениями характерна погрешность их измерения, поэтому задачу следует ставить как задачу присвоения весов $q_1,...,q_N$, с которыми должны учитываться признаки в процессе классификации.

Мерой сходства двух образов является расстояние между ними. В случае евклидового расстояния с учетом весов получаем:

$$s_{\mathbf{q}}^2(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{N} q_i (x_i - y_i)^2 . \qquad (73)$$

Следует найти такие веса, которые бы максимизировали среднее расстояние между образами различных классов и минимизировали среднее расстояние между образами, принадлежащими одному и тому же классу:

$$L(\mathbf{q}) = \frac{1}{d^2}\sum_{k=1}^{d}\sum_{l=1}^{d} S_{\mathbf{q}}^2\left(\left\{\mathbf{x}_i^{(k)}\right\}_{i=1}^{M_k}, \left\{\mathbf{x}_j^{(l)}\right\}_{j=1}^{M_l}\right) - \frac{1}{d}\sum_{k=1}^{d} S_{\mathbf{q}}^2\left(\left\{\mathbf{x}_i^{(k)}\right\}_{i=1}^{M_k}\right), \qquad (74)$$

где $S_{\mathbf{q}}(\cdot)$ и $S_{\mathbf{q}}(\cdot,\cdot)$ – средние расстояния внутри множества и между двумя множествами соответственно (с учетом весов признаков). Для их вычисления необходимо усреднить расстояния между всеми парами точек, однако в случае евклидовой метрики вместо этого можно считать расстояния до векторов средних.

Пусть $\mathbf{y}^{(k)}$ – эталонный образ (вектор средних) *k*-го класса, тогда внутриклассовое расстояние может быть вычислено по формуле

$$S_{\mathbf{q}}^2\left(\left\{\mathbf{x}_i^{(k)}\right\}_{i=1}^{M_k}\right) = \frac{1}{M_k}\sum_{j=1}^{M_k}\sum_{i=1}^{N} q_i \left(x_{j,i}^{(k)} - y_i^{(k)}\right)^2 . \qquad (75)$$

Преобразуем среднее внутриклассовое расстояние:

$$\frac{1}{d}\sum_{k=1}^{M_k} S_{\mathbf{q}}^2\left(\left\{\mathbf{x}_i^{(k)}\right\}_{i=1}^{M_k}\right) = \frac{1}{d}\sum_{k=1}^{M_k}\frac{1}{M_k}\sum_{j=1}^{M_k}\sum_{i=1}^{N} q_i \left(x_{j,i}^{(k)} - y_i^{(k)}\right)^2 =$$

$$= \sum_{i=1}^{N} q_i \frac{1}{d} \sum_{k=1}^{M_k} \frac{1}{M_k} \sum_{j=1}^{M_k} \left(x_{j,i}^{(k)} - y_i^{(k)} \right)^2 = \sum_{i=1}^{N} q_i \sigma_i^2 ,$$

где σ_i^2 – средняя по всем классам дисперсия *i*-го признака.

Расстояние между классами образов в простейшем случае может быть вычислено как расстояние между их эталонными образами:

$$S_{\mathbf{q}}^2 \left(\left\{ \mathbf{x}_i^{(k)} \right\}_{i=1}^{M_k}, \left\{ \mathbf{x}_j^{(l)} \right\}_{j=1}^{M_l} \right) = \sum_{i=1}^{N} q_i \left(y_i^{(k)} - y_i^{(l)} \right)^2 . \quad (76)$$

Среднее расстояние между классами составит

$$\frac{1}{d^2} \sum_{k=1}^{d} \sum_{l=1}^{d} \sum_{i=1}^{N} q_i \left(y_i^{(k)} - y_i^{(l)} \right)^2 = \sum_{i=1}^{N} q_i \frac{1}{d^2} \sum_{k=1}^{d} \sum_{l=1}^{d} \left(y_i^{(k)} - y_i^{(l)} \right)^2 = \sum_{i=1}^{N} q_i \Sigma_i^2 , \quad (77)$$

где Σ_i^2 – среднее расстояние вдоль *i*-го признака между классами образов.

Тогда целевую функцию можно записать в виде

$$L(\mathbf{q}) = \sum_{i=1}^{N} q_i \Sigma_i^2 - \sum_{i=1}^{N} q_i \sigma_i^2 . \quad (78)$$

Здесь дисперсии σ_i^2 и Σ_i^2 зависят только от распределения образов внутри классов и от взаимного расположения классов, но не зависят от вектора параметров $\mathbf{q}$.

Очевидно, для нахождения единственного минимума целевой функции необходимо ввести некоторое ограничение на вектор весов. В качестве такого ограничения можно использовать, например, условие

$$\sum_{i=1}^{N} q_i^2 = 1 . \quad (79)$$

Тогда получаем задачу нахождения условного экстремума, для которой необходимо составить функцию Лагранжа:

$$L(\mathbf{q}, \lambda) = L(\mathbf{q}) - \lambda \left(\sum_{i=1}^{N} q_i^2 - 1 \right). \quad (80)$$

Используя условие экстремума

$$\frac{\partial L(\mathbf{q}, \lambda)}{\partial q_i} = \left(\Sigma_i^2 - \sigma_i^2 \right) - 2\lambda q_i = 0 , \quad (81)$$

несложно получить значения весов по формуле

$$q_i = \frac{1}{2\lambda} \left(\Sigma_i^2 - \sigma_i^2 \right), \text{ где } 2\lambda = \left[\sum_{j=1}^{N} \left(\Sigma_j^2 - \sigma_j^2 \right)^2 \right]^{0,5} . \quad (82)$$

Однако если вычислять межклассовые расстояния как расстояния между эталонными образами, то веса могут получиться отрицательными. По этой и ряду других причин подобный способ вычисления расстояний

между классами является не вполне корректным: правильнее считать среднее расстояние от образов одного класса до эталонного образа другого класса. В этом случае межклассовые расстояния всегда будут превосходить внутриклассовые.

Руководствуясь несколько другой эвристически сконструированной функцией потерь и условием $\prod_{i=1}^{N} q_i = 1$, описывающим сохранение элемента объема, можно получить более удобные в использовании веса:

$$q_i = \frac{1}{\lambda}\Sigma_i^2 \Big/ \sigma_i^2 \text{ , где } \lambda = \left(\prod_{i=1}^{N} \frac{\sigma_i^2}{\Sigma_i^2} \right)^{1/N} . \qquad (83)$$

Такие веса вполне могут использоваться на практике (рис. 33).

Использование евклидового расстояния в качестве меры сходства имеет очевидные ограничения. Более универсальным критерием является количество информации. В классическом (шенноновском) подходе среднее количество информации выражается через энтропию, для вычисления которой необходимо знать соответствующие вероятности.

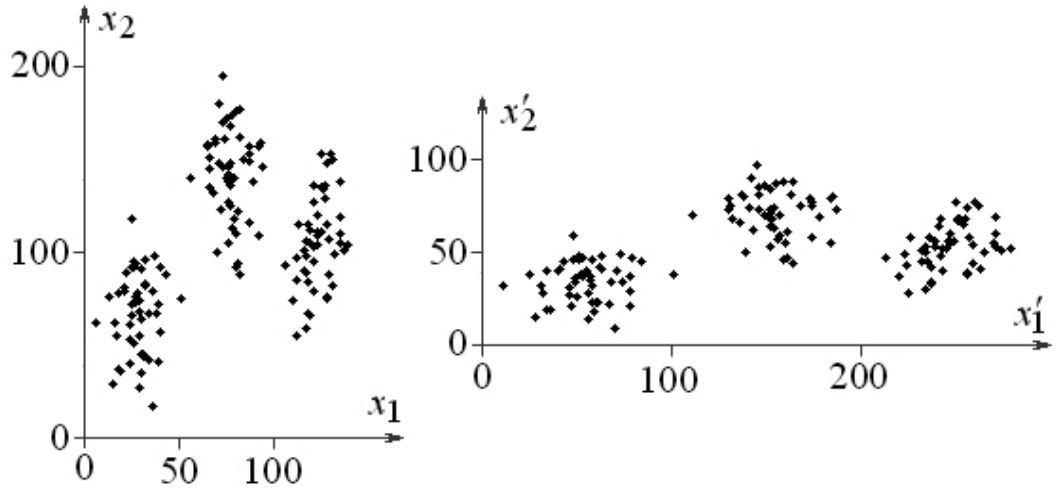


Рис. 33. Пример преобразования кластеризации. Слева в исходном пространстве признаков представлены образы, принадлежащие трем классам. По этим данным были определены веса признаков $q_1 \approx 4$ и $q_2 \approx 0{,}25$. Справа изображен результат преобразования кластеризации, представленный как переход к новым признакам $x_1' = \sqrt{q_1}\,x_1$ и $x_2' = \sqrt{q_2}\,x_2$

Поскольку речь идет об обучении с учителем, можно полагать, что плотности распределения вероятностей $p_k(\mathbf{x}), k = 1,...,d$ для каждого класса вычислены. Тогда аналог расстояние от образа $\mathbf{x}$ до класса k будет равен

$$I(\mathbf{x} \mid a_k) = -\log_2 p_k(\mathbf{x}), \qquad (84)$$

то есть собственному количеству информации, содержащейся в образе в предположении, что он принадлежит k-му классу. Связь количества

информации $I(\mathbf{x}\,|\,a_k)$ с расстоянием становится очевидной, если в качестве плотности вероятностей подставить нормальное распределение.

Теперь вместо среднего внутриклассового расстояния мы можем использовать энтропию, вычисляемую по формуле

$$H_k = -\int_{\mathrm{X}} p_k(\mathbf{x}) \log_2 p_k(\mathbf{x}) d\mathbf{x}. \qquad (85)$$

Величина H_k определяет среднее количество информации, содержащееся в произвольном образе *k*-го класса, если известна его принадлежность этому классу.

Аналог расстояния от класса *l* до класса *k* следует определить как среднее «расстояние» образов класса *l* до класса *k*:

$$H_{k,l} = -\int_{\mathrm{X}} p_l(\mathbf{x}) \log_2 p_k(\mathbf{x}) d\mathbf{x}. \qquad (86)$$

Эта величина характеризует среднюю длину оптимального кода, кодирующего произвольный образ класса *l* в предположении, что он принадлежит классу *k*.

Как и выше для случая евклидового расстояния, нас интересует ситуация, в которой расстояния между классами образов максимальны, а внутриклассовые расстояния минимальны. Тогда в качестве целевой функции (при рассмотрении двух классов) следует взять величину

$$J_{k,l} = H_{k,l} + H_{l,k} - H_k - H_l. \qquad (87)$$

С помощью несложных преобразований получаем

$$J_{k,l} = \int_{\mathrm{X}} \big(p_k(\mathbf{x}) - p_l(\mathbf{x})\big) \log_2 \frac{p_k(\mathbf{x})}{p_l(\mathbf{x})} d\mathbf{x}. \qquad (88)$$

Эта величина выражает полную среднюю информацию для различения двух классов и обычно называется *дивергенцией* двух классов.

Заметим, что критерий дивергенции можно свести к рассмотренным ранее критериям, основанным на функциях расстояния, если плотности вероятности положить равными нормальным распределениям частного вида. Можно также дать нестрогое пояснение, почему веса признаков (83) предпочтительнее весов (82). Действительно, в случае одномерного нормального распределения энтропия пропорциональна логарифму дисперсии. В уравнении для дивергенции энтропии складываются и вычитаются, поэтому внутриклассовые и межклассовые расстояния необходимо делить и перемножать между собой.

Таким образом, концепция дивергенции позволяет строго ввести и оперировать степенью различия между классами. При этом также учитываются стохастические модели классов $p_k(\mathbf{x})$, для конкретного вида которых не представляет сложности уточнить уравнение (88). Если в результате оно получается неинтегрируемым аналитически, то дивергенция может быть оценена путем замены интеграла по всему пространству признаков на суммирование по точкам обучающей выборки. Для определения весов признаков необходимо рассчитать величины

дивергенции не по самим плотностям вероятности $p_k(\mathbf{x})$, а по их сечениям по каждому признаку.

Описанные выше подходы позволяют упорядочивать признаки по весам, то есть по степени их эффективности, и выбирать из них некоторое количество наиболее значимых для уменьшения размерности пространства признаков. Однако при этом не происходит построение новых признаков. Можно было бы тестировать каждый новый признак с помощью одного из введенных выше критериев, но этот подход имеет очевидный недостаток. Пусть мы хотим, например, построить новый признак, являющийся линейной комбинацией уже существующих признаков, но ее коэффициенты неизвестны. Поскольку перебор всех возможных признаков, заданных таким параметрическим способом, бесполезен, возникает вопрос: как определить оптимальные коэффициенты линейной комбинации?

Для решения этой задачи выбора признаков ее можно поставить как задачу определения оптимального отображения $F : \mathrm{X} \to \mathrm{X}'$ из исходного пространства признаков в новое пространство. Отображение *F* выбирается принадлежащим некоторому параметрическому семейству $F(\mathbf{x}, \mathbf{w})$, где $\mathbf{w}$ – вектор параметров, оптимальное значение которых необходимо определить. Для этого максимизируем дивергенцию:

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \left[\int_{\mathrm{X}} [p_k(F(\mathbf{x},\mathbf{w})) - p_l(F(\mathbf{x},\mathbf{w}))] \log_2 \frac{p_k(F(\mathbf{x},\mathbf{w}))}{p_l(F(\mathbf{x},\mathbf{w}))} d\mathbf{x} \right]. \quad (89)$$

Даже для линейного преобразования координат: $F(\mathbf{x}, \mathbf{W}) = \mathbf{x}^T \mathbf{W} \mathbf{x}$, где $\mathbf{W}$ – матрица преобразования, аналитическое решение найдено лишь для случаев, когда $p_k(\mathbf{x})$ – нормальные распределения частного вида (например, с одинаковыми ковариационными матрицами для всех классов). В противном случае необходимо использовать приближенные методы, что ограничивает их практическое применение.

Проблема выбора признаков при обучении без учителя

При выборе признаков, как и при нахождении преобразования кластеризации, требуется найти некоторое оптимальное преобразование $F : \mathrm{X} \to \mathrm{X}'$, причем в дискриминантном подходе пространства имеют вид $\mathrm{X} = R^N$ и $\mathrm{X}' = R^n$. Однако здесь в качестве исходных данных выступает набор векторов $\{\mathbf{x}_i\}_{i=1}^N, \mathbf{x}_i \in R^N$ без информации об их принадлежности классам. Несложно заметить, что в постановке задачи нигде в явном виде не указывается, что исходные объекты, векторы признаков которых поступают на вход системы, могут относиться к различным классам. Чтобы связать качество признаков с эффективностью классификации, нам бы пришлось для каждого возможного набора признаков решать задачу группирования и оценивать дивергенцию формируемых классов. Очевидно, такой путь приводит к методам, обладающим высокой

вычислительной сложностью. Напротив, можно было бы сначала решить задачу группирования, а затем – выбора признаков с помощью преобразования кластеризации. Такой подход в ряде случаев допустим, но, в целом, при его применении теряется смысл процедуры выбора признаков как предварительной обработки данных, помогающей снизить размерность данных и решить последующие задачи распознавания.

В связи с этим обычно рассматривают критерии оптимальности преобразования F, напрямую не связанные с операциями распознавания образов. Методы снижения размерности, использующие такие критерии, применяются и при решении других проблем, например, при сжатии данных, в когнитивной графике, для подавления шумов измерений и т.д. Широкая применимость таких методов говорит в пользу корректности использования критериев, не опирающихся на качество классификации.

Исторически первыми и наиболее разработанными методами выбора признаков в случае обучения без учителя являются методы, использующие статистические моменты второго порядка. В этих методах в качестве критерия качества выступает точность, с которой образы описываются в новом пространстве признаков уменьшенной размерности. Потеря же точности описания трактуется с точки зрения евклидового расстояния. Применение этих методов в некоторых приложениях, в частности в распознавании образов, показало их ограниченность. Это вызвало интерес, подкрепленный некоторыми нейрофизиологическими данными, к методам, ведущим поиск наиболее «интересных» признаков, привлекая статистики более высоких порядков. В итоге исследователи пришли к критерию, указывающему, что наилучшие признаки должны быть статистически независимыми. Рассмотрим сначала методы второго порядка.

Анализ главных компонент и факторный анализ

Классические методы второго порядка, предназначенные для выбора признаков, – это *анализ главных компонент* (АГК; principal component analysis, PCA) и *факторный анализ* (ФА; factor analysis, FA). Эти методы очень похожи, если сравнивать их по результирующим формулам, поэтому они иногда отождествляются или один метод рассматривается в качестве частного случая другого метода, хотя исходные предпосылки в них различаются. Начнем описание с АГК.

Предположим, что мы хотим уменьшить размерность векторов признаков таким образом, чтобы по образам, описанным с помощью новых признаков, можно было бы как можно более точно восстановить исходные образы. Рассмотрим сначала случай $n = 1$.

Ограничимся линейными преобразованиями пространства X. Тогда новый признак должен являться линейной комбинацией исходных признаков, то есть должен определять некоторое направление $\mathbf{w}_1$ в пространстве X. Это направление называется первой *главной компонентой*. Минимизация потери точности эквивалентна максимизации

дисперсии проекций векторов обучающей выборки на искомое направление:

$$\mathbf{w}_1 = \arg\max_{\|\mathbf{w}\|=1} \sum_{i=1}^{M} \left(\mathbf{w}^T (\mathbf{x}_i - \mathbf{y})\right)^2, \qquad (90)$$

где $\mathbf{y} = \frac{1}{M}\sum_{i=1}^{M} \mathbf{x}_i$ – вектор средних.

Значение найденного таким образом признака для *i*-го вектора будет равно $x'_{i,1} = \mathbf{w}_1^T \mathbf{x}_i$. Однако поскольку вектор $\mathbf{w}_1$ соответствует некоторому направлению в исходном пространстве, величина $\mathbf{w}_1\mathbf{w}_1^T \mathbf{x}_i$ будет проекцией *i*-го вектора на данное направление, а $\mathbf{x}_i - \mathbf{w}_1\mathbf{w}_1^T \mathbf{x}_i$ – его проекцией на *N*–1-мерное пространство, перпендикулярное этому направлению. Это тот остаток от вектора $\mathbf{x}_i$, который не описывается новым признаком. В таком *N*–1-мерном пространстве можно найти следующее направление, проекция векторов обучающей выборки на которое обладает максимальной дисперсией. После *k*–1 таких итераций остатки будут иметь вид:

$$\mathbf{x}_i^{(k-1)} = \mathbf{x}_i - \sum_{j=1}^{k-1} \mathbf{w}_j \mathbf{w}_j^T \mathbf{x}_i, \qquad (91)$$

и на их основе можно будет найти очередную *k*-ю главную компоненту $\mathbf{w}_k$ абсолютно так же, как была найдена первая и все последующие компоненты. Отметим, что направления, соответствующие главным компонентам, получаются ортогональными (см. рис. 34).

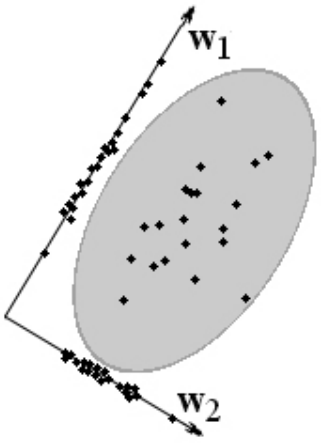


Рис. 34. Пример определения главных компонент. Проекция векторов выборки на направление $\mathbf{w}_1$ обладает максимальной дисперсией, то есть позволяет максимально полно объяснить вариативность данных. В данном случае главные компоненты адекватно отражают структуру данных

Оказывается, что поиск *n* главных компонент совпадает с нахождением *n* собственных векторов ковариационной матрицы

$\mathbf{C} = \frac{1}{M}\sum_{i=1}^{M}(\mathbf{x}_i - \mathbf{y})(\mathbf{x}_i - \mathbf{y})^T$, соответствующих n наибольшим собственным числам. Это дает возможность не искать последовательно главные компоненты, максимизируя дисперсию проекции векторов обучающей выборки, а использовать стандартные операции с матрицами для определения собственных векторов и чисел. Собственные векторы соответствуют направлению осей эллипсоида, вписанного в данные, а собственные числа – размерам осей (точнее, их квадратам).

В факторном анализе, в отличие от АГК, не минимизируются погрешности произвольного описания, а производится построение оптимальной модели объектов одного типа. Векторы обучающей выборки – это измерения характеристик объектов, но эти измерения выявляют не "истинные" свойства или признаки объектов (или скрытые факторы), которые недоступны наблюдателю, а некоторые внешние проявления этих факторов. В своих проявлениях, доступных наблюдателю, факторы смешаны друг с другом и зашумлены. Если предполагать, что факторы смешиваются линейным образом, то модель, описывающей измерение признаков некоторого объекта, будет иметь вид

$$\mathbf{x} = \mathbf{W}\boldsymbol{\chi} + \boldsymbol{\nu}, \qquad (92)$$

где $\boldsymbol{\chi}$ – вектор скрытых факторов (признаков), $\mathbf{W}$ – матрица, определяющая связь между скрытыми признаками и наблюдаемыми признаками, а случайный вектор $\boldsymbol{\nu}$ описывает шум. Разные объекты обладают различными значениями скрытых признаков, но одной и той же матрицей $\mathbf{W}$, поскольку она описывает природу признаков.

Поскольку ФА – метод второго порядка, при отсутствии шума следовало бы минимизировать величину

$$\varepsilon = \frac{1}{M}\sum_{i=1}^{M}\left(\mathbf{x}_i - \mathbf{W}\boldsymbol{\chi}_i\right)^2, \qquad (93)$$

определяющую точность, с которой модель описывает процесс порождения данных. Это выражение полностью соответствует критерию, выводимому в АГК. Помимо того, что этот факт позволяет применять и в ФА прием, использованный в АГК, а именно, позволяет искать скрытые факторы (или "истинные" признаки объектов) как собственные векторы ковариационной матрицы $\mathbf{C}$, он также имеет крайне важное следствие. Суть этого следствия в том, что поиск оптимального представления данных (осуществляемый в АГК) идентичен поиску оптимальной модели источника, порождающего эти данные (что реализуется в ФА).

Следует, однако, заметить, что, исходя из уравнения для погрешности (93), факторы определяются неоднозначно. Действительно, если взять подпространство $\mathrm{X}' \subseteq \mathrm{X}$, натянутое на факторы, то любая полная система векторов в этом подпространстве, называемым *факторным* или *АГК подпространством*, будет соответствовать минимуму погрешности (93). Таким образом, помимо минимизации погрешности необходимо

использовать дополнительный критерий для выбора конкретных факторов внутри этого подпространства.

Еще одна деталь, отличающая ФА от АГК, заключается в том, что в АГК размерность n пространства X' является либо заданной, либо может быть вычислена, если задана погрешность, с которой в новом пространстве признаков описываются образы. Иными словами, в АГК размерность n определяется тем, где и как будет использоваться новое представление. Например, в задачах когнитивной графики обычно n=2. Факторный же анализ претендует на восстановление истинных признаков объектов, поэтому число таких признаков должно определяться вместе с ними самими, для чего приходится привлекать дополнительные критерии.

И, наконец, отличие ФА и АГК, которое сказывается на конечных формулах, заключается в ведении слагаемого $\mathbf{v}$, описывающего шум. При наличии шума, который обычно предполагается гауссовым, необходимо искать собственные векторы и собственные числа не ковариационной матрицы $\mathbf{C}$, а матрицы $\mathbf{C} - \mathbf{C}[\mathbf{v}]$, где $\mathbf{C}[\mathbf{v}]$ – ковариационная матрица шума. Если эта матрица известна, то содержательно задача не меняется, в противном случае необходимы более сложные методы анализа.

В АГК и ФА неявно предполагается, что все образы обучающей выборки относятся к объектам одного класса. Это является очевидным ограничением, особенно в задачах распознавания образов (см. рис. 35).

Несмотря на свою ограниченность, методы второго порядка обладают определенной привлекательностью. Такие методы опираются лишь на информацию из ковариационной матрицы и вектора средних, вычислительно просты и используют лишь классические операции с матрицами, не требуя разработки процедур поиска в пространстве параметров преобразования. В связи с этим для образов, содержащих очень большое количество признаков, АГК и ФА могут стать наиболее подходящими методами предварительного выбора признаков. И конечно, такие методы будут являться оптимальными, если векторы действительно распределены нормально.

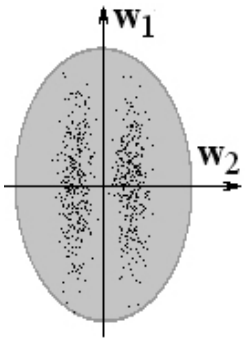


Рис. 35. Классический пример ситуации, в которой применение анализа главных компонент повлечет потерю важной информации. Векторы разделяются на два класса. Выбор же первой главной компоненты, обладающей максимальной дисперсией, приведет к полной неразделимости классов

Уменьшение избыточности и поиск интересных направлений

В общем случае плотность вероятностей не описывается своими вторыми моментами, и возникает необходимость использовать более сложные методы. Исторически такие методы разрабатывались в двух направлениях: уменьшение избыточности данных и поиск «интересных» направлений в пространстве признаков. Методы, разработанные в рамках обоих подходов, оказались тесно связанными и впоследствии привели к подходу, называемому анализом независимых компонент.

Происхождение идеи выбора интересных направлений может стать понятным из рассмотрения рис. 35. Если посмотреть на этот рисунок, то видно, что проекции векторов на направление $\mathbf{w}_1$ распределены нормально, в то время как распределение их проекций на направление $\mathbf{w}_2$ сильно отличаются от нормального закона. И хотя дисперсия вдоль второй главной компоненты меньше, чем вдоль первой, это направление кажется более значимым или интересным. Если пытаться найти такие направления, не производя в явном виде кластеризацию, то это приводит к концепции «интересных» направлений в пространстве признаков как направлений, вдоль которых распределение данных наиболее сильно отличается от гауссова распределения, считающегося наименее интересным.

Нормальное распределение обладает максимальной энтропией при фиксированной ковариационной матрице. Для любого другого распределения, энтропия меньше. В связи с этим одним из наиболее популярных критериев интересности направления является энтропия вдоль него при фиксированной дисперсии. Пусть $\boldsymbol{X}$ – случайный вектор, реализациями которого являются векторы обучающей выборки. Тогда наиболее интересное направление будет определяться по формуле

$$\mathbf{w}_1 = \underset{\mathbf{w}:\sigma\left(\mathbf{w}^T\boldsymbol{X}\right)=1}{\arg\min} H(\mathbf{w}^T\boldsymbol{X}), \qquad (94)$$

где σ – дисперсия, а H – энтропия случайной величины.

Во втором подходе, в рамках которого исследуются методы, не ограничивающиеся анализом ковариационной матрицы, выбираются компоненты, позволяющие уменьшить избыточность данных. Этот подход был развит во многом на основе нейрофизиологических данных. Согласно этим данным, в естественных нейронных сетях (выполняющих первичную обработку сенсорной информации) производится уменьшение избыточности в том смысле, что нейроны одного слоя нейронной сети настраивают свои связи таким образом, чтобы как можно реже активироваться совместно. Иными словами, нейроны выделяют максимально независимые признаки, в связи с чем их активность является некоррелированной. Эти идеи, почерпнутые из естественных нейронных сетей, были перенесены на искусственные нейронные сети, из-за чего данный критерий развивался преимущественно в рамках нейросетевого подхода.

На языке теории вероятности избыточность признаков можно интерпретировать как наличие статистической зависимости между ними. Стремление уменьшить эту избыточность соответствует поиску статистически независимых признаков. Именно эта цель преследуется в методе, известном как *анализ независимых компонент* (АНК; independent component analysis, ICA).

Наиболее популярный критерий статистической независимости опирается на понятия взаимной информации и совместной энтропии. Пусть $\boldsymbol{X}' = (X'_1,...,X'_n)$ – случайный вектор, полученный из случайного вектора $\boldsymbol{X} = (X_1,...,X_N)$ в результате применения регулярного преобразования F. Статистическая независимость случайных величин $X'_1,...,X'_n$ равносильна условию

$$H(\boldsymbol{X}') = H(X'_1) + ... + H(X'_n). \qquad (95)$$

Отсюда можно получить критерий, указывающий степень статистической независимости:

$$\mathrm{I}(X'_1,...,X'_n) = H(X'_1) + ... + H(X'_n) - H(\boldsymbol{X}'), \qquad (96)$$

что является средней взаимной информацией случайных величин $X'_1,...,X'_n$. Средняя взаимная информация является численной оценкой информационной избыточности признаков, поэтому данный критерий можно было бы получить сразу же, если вместо статистической избыточности была бы рассмотрена информационная. Более того, если обратиться к принципу минимальной длины описания, то становится очевидным, что в данном критерии не учитывается сложность преобразования *F*. Если будет использоваться одно параметрическое семейство преобразований, то недоучет длины описания преобразования не будет играть роли. Однако если рассматриваются нелинейные преобразования пространства признаков произвольной сложности, то существует опасность переобучения. К счастью, принцип МДО позволяет устранить эту опасность.

Помимо тесной связи с методами уменьшения избыточности АНК является и обобщением методов второго порядка. Отсутствие корреляции между признаками является частным случаем их статистической независимости. Факторный анализ можно рассматривать именно как поиск признаков, между которыми отсутствует корреляция. Значит, ФА получается из АНК при предположении о нормальном распределении случайных величин.

Вопросы и упражнения

1. Как называется задача выбора признаков в случае обучения с учителем?
2. В чем сущность понятия дивергенции классов?
3. К какой операции сводятся анализ главных компонент и факторных анализ?

4. В чем отличие факторного анализа от анализа главных компонент?
5. Какой критерий используется для определения понятия интересного направления в пространстве признаков?
6. На основе какого критерия чаще всего определяется степень независимости направлений в анализе независимых компонентов?

15. Восстановление формальных грамматик

Ранее были рассмотрены вопросы обучения на примере задач распознавания образов. При этом объекты описывались векторами вещественных признаков. То есть исходное представление информации имело численную природу. Помимо этого широко распространена проблема обучения в рамках символьных представлений. Здесь существует большое разнообразие методов в зависимости от выходного представления, в качестве которого могут выступать фреймы, системы продукций, формальные грамматики, деревья решений и т.д. Мы рассмотрим проблему обучения в рамках формальных грамматик.

Напомним, что формальная грамматика – это четверка $G = (V_T, V_N, P, S)$, где V_T – алфавит терминальных символов; V_N – алфавит нетерминальных символов, причем $V_N \cap V_T = \varnothing$; $V_N \cup V_T = V$ – алфавит грамматики G; P – множество правил подстановки; S – начальный символ, $S \in V_N$.

Как и в случае численных данных, обучение формальным грамматикам основывается по некоторой исходной информации. Здесь обучающей выборкой является совокупность символьных строк, которая может содержать как положительные, так и отрицательные примеры.

Пусть Γ_0 – некий изучаемый язык. Введем следующие определения.

Информационной последовательностью $I(\Gamma_0)$ языка Γ_0 будем называть последовательность цепочек, каждая из которых принадлежит одному из множеств $\{\alpha^+ \mid \alpha^+ \in \Gamma_0\}$ или $\{\alpha^- \mid \alpha^- \in V_T^* \setminus \Gamma_0\}$ с указанием того, к какому множеству принадлежит та или иная цепочка. Последовательность цепочек языка $\{\alpha^+ \mid \alpha^+ \in \Gamma_0\}$ будем называть *положительной информационной последовательностью* $I^+(\Gamma_0)$, а последовательность цепочек из дополнения языка $\{\alpha^- \mid \alpha^- \in V_T^* \setminus \Gamma_0\}$ будем называть *отрицательной информационной последовательностью* $I^-(\Gamma_0)$.

Информационная последовательность $I(\Gamma_0)$ называется *полной*, если $I^+(\Gamma_0)$ содержит все цепочки языка Γ_0, а $I^-(\Gamma_0)$ содержит все цепочки, не принадлежащие языку Γ_0.

Грамматика G *согласована* с грамматикой G_0, если порождаемые ими языки совпадают $\Gamma(G) = \Gamma(G_0)$.

Пусть $C = \{G_i\}$ – класс грамматик. Класс языков $\Gamma(C) = \{\Gamma(G) \mid G \in C\}$ называется *идентифицируемым*, если для любой грамматики $G \in C$ и любой полной информационной последовательности $I(\Gamma(G))$ существует некоторое число N и алгоритм, который бы, получая на входе подпоследовательность $I(\Gamma(G))$, содержащую не менее N цепочек, на выходе давал бы грамматику, согласованную с грамматикой G.

Наряду с понятием информационной последовательности используется понятие образца (или выборки) языка Γ_0. *Образцом* S_t языка Γ_0 будем называть последовательность цепочек $\{\alpha_i\}_{i=1}^{t}$, для каждой цепочки которой известно, принадлежит ли она языку Γ_0 или его дополнению $V_T^* \setminus \Gamma_0$. *Положительным образцом* будем называть множество $S_t^+ = S_t \cap \Gamma_0$, а *отрицательным образцом* – множество $S_t^- = S_t \cap (V_T^* \setminus \Gamma_0)$.

Грамматика G называется *совместимой* с образцом S_t, если она порождает все положительные примеры этого образца и не порождает ни одного отрицательного примера.

Структурно полный образец $S_t(\Gamma(G))$ языка $\Gamma(G)$ – это образец, содержащий такие цепочки, при построении которых каждое правило подстановки грамматики G использовалось хотя бы по одному разу.

Структурная полнота образца является необходимым условием возможности восстановления всех правил грамматики, в то время как полнота информационной последовательности может выступать в качестве достаточного условия идентифицируемости некоторых классов языков. На практике структурная полнота образца достигается существенно легче, чем полнота информационной последовательности, но обоснование структурной полноты также возможно далеко не во всех случаях (чтобы убедиться в этом, попробуйте составить структурно полный образец русского языка).

При решении задачи грамматического вывода различают *текстуальное* (текстовое) представление, при котором имеются лишь положительные примеры, и *информаторное* представление, при котором есть как положительные, так и отрицательные примеры.

Если дан образец языка или информационная последовательность, то проблема обучения, очевидно, заключается в построении грамматики, совместимой с данным образцом, или в идентификации некоторой истинной грамматики. Иными словами, требуется построить формальную грамматику, которая бы как можно лучше описывала структуру языка Γ_0.

Существует два варианта постановки задачи восстановления грамматик.

В первой формулировке (на которую будем ссылаться как на *проблему согласования*) предполагается, что есть некоторая истинная грамматика G_0, и требуется по информационной последовательности построить такую грамматику G, которая была бы согласована с грамматикой G_0, то есть $\Gamma(G) = \Gamma(G_0)$.

Во второй формулировке (на которую будем ссылаться как на *проблему грамматического вывода*) считается, что по образцу S_t необходимо построить такую грамматику G, которая бы порождала все цепочки положительного образца S_t^+ (и, возможно, бесконечное множество других цепочек) и не порождала цепочки отрицательного образца S_t^- (и, возможно, бесконечное множество других цепочек), то есть была бы совместима с этим образцом.

Обычно выделяют два класса алгоритмов восстановления грамматик: перечислением и индукцией. Эти два класса алгоритмов имеют определенную связь с двумя приведенными формулировками задачи восстановления грамматик. Сначала мы кратко рассмотрим восстановление грамматик перечислением.

Восстановление грамматик перечислением

При формулировании проблемы согласования грамматик предполагается, что существует некая истинная грамматика, и целью является нахождение грамматики, согласованной с этой истинной грамматикой. Требование согласованности является очень сильным, так что вызывает сомнение возможность достижения этой цели в достаточно общем случае. Возникает вопрос, при каких ограничениях проблема согласования разрешима?

При теоретическом исследовании этого вопроса оказывается необходимым предполагать, что даны *полные* информационные последовательности (либо только положительная, либо положительная и отрицательная).

Требование полноты позволяет строить теоремы о существовании алгоритмов восстановления грамматик при различных условиях и давать их формальные доказательства. К сожалению, полных информационных последовательностей на практике не встречается. Доказательства существования алгоритмов обычно неконструктивны (либо в них строятся алгоритмы, непригодные из-за проблемы комбинаторного взрыва). Такие теоремы полезны тем, что говорят, для решения каких вариантов задачи грамматического вывода алгоритмы, в принципе, не стоит искать.

Для практики эти результаты, однако, оказываются не слишком полезными. Например, из теоретических исследований известно, что не существует алгоритма, который бы решал проблему согласования для любой грамматики из класса регулярных грамматик при текстуальном представлении. Более того, если рассматривается класс грамматик,

включающий все конечные грамматики и лишь одну бесконечную грамматику, то для этой грамматики проблема согласования также не может быть решена на основе текстуального представления. Несмотря на такой суровый приговор, на практике приходится решать проблемы восстановления грамматик на основе текстуального представления, даже используя неполную положительную информационную последовательность, вовсе не ограничиваясь при этом простейшими регулярными грамматиками (при этом восстановление «истинной» грамматики, конечно, не гарантируется).

С упомянутыми теоретическими работами тесно связаны алгоритмы восстановления грамматик перечислением. При восстановлении грамматик перечислением осуществляется перебор (перечисление) всех грамматик из некоторого заданного класса до тех пор, пока не будет найдена грамматика, согласованная с информационной последовательностью языка. Естественно, для бесконечных языков на практике не может использоваться полная информационная последовательность. В связи с этим используется некоторая конечная ее подпоследовательность. Если находится несколько грамматик, совместимых с такой подпоследовательностью, то размер подпоследовательности увеличивается и происходит дальнейший отсев грамматик. При таком полном переборе «истинная» грамматика находится гарантированно, коль скоро рассматривается идентифицируемый класс языков.

Такое свойство алгоритмов восстановления грамматик не может не являться привлекательным. Однако на практике в полной мере встает проблема комбинаторного взрыва при перечислении грамматик. В связи с этим многие работы по восстановлению грамматик перечислением нацелены на оптимизацию перебора. Одним из базовых приемов является установление наиболее приемлемого порядка перечисления грамматик. Таковым оказывается *оккамовское перечисление*, при котором грамматики рассматриваются в порядке возрастания сложности. Эффективность оккамовского перечисления может быть обоснована не только на основе эвристических соображений, но и строго на основе принципа МДО.

Однако несмотря на попытки уменьшить вычислительные затраты при восстановлении грамматик перечислением, эти методы остаются мало приемлемыми на практике. Это свидетельствует о том, что формальное требование к идентифицируемости языка является крайне сильным.

В отличие от восстановления грамматик перечислением при восстановлении грамматик индукцией отсутствует требование к нахождению «истинного» (точного) решения. Хотя алгоритмы восстановления грамматик перечислением и индукцией могут и не содержать фундаментальных различий, эти два подхода принципиально различны с методологической точки зрения. В первом случае в качестве отправной точки берется вопрос о существовании «истинного» решения проблемы при тех или иных ограничениях, несмотря на то, что проверка удовлетворения этим ограничениям на практике недостижима (понятие

идентифицируемости относится к классу языков; но как достоверно определить, к какому классу принадлежит неизвестный язык?). Во втором случае базовым является понятие модели, которая (при описании с ее помощью некоторого феномена реального мира) признается принципиально неточной или имеющей вероятностный характер.

Эвристические процедуры грамматического вывода

Рассмотрим задачу грамматического вывода. В этой задаче не обязательно строить грамматику, согласованную с некоторой истинной грамматикой, а достаточно вывести грамматику, совместимую с данным образцом языка $S_t = \{\alpha_i\}_{i=1}^{t}$, то есть ставится проблема поиска лучшего решения на основе *имеющейся* информации. Если для восстановления грамматик перечислением наличие информатора крайне желательно, то при грамматическом выводе чаще ограничиваются текстуальным представлением. Мы также рассмотрим задачу восстановления по положительному образцу S_t^+.

Однако является ли совместимость с образцом языка удовлетворительным критерием при восстановлении грамматики? Для текстуального представления имеем два очевидных тривиальных решения: *ad hoc* и «беспорядочную» грамматики. Первая грамматика допускает только данные цепочки и содержит *t* правил $\{S \to \alpha_i\}_{i=1}^{t}$, вторая допускает вообще все цепочки (например, содержит правила вида $\{S \to a_i S \mid a_i \in V_T\}; S \to \Lambda$).

Если бы требовалось просто построить грамматику, которая порождала данные цепочки, то эта задача решалась бы тривиально. Но такие решения нас интуитивно не устраивают. Что же мы хотим от грамматического вывода?

Обратим внимание, что предложенные тривиальные решения обладают следующими особенностями. Первое решение дает грамматику, не порождающую возможные отрицательные предложения, но при этом ни один новый положительный пример, не вошедший в образец, также порождаться не будет. Второе решение дает грамматику, порождающую любой новый положительный пример, но также порождающую и любое отрицательное предложение.

Здесь в завуалированном виде присутствует проблема обобщения, которая характерна для всех задач обучения. Казалось бы, нам нужна любая грамматика, удовлетворяющая набору цепочек. Но зачем нужна эта грамматика? Произвольная грамматика, просто порождающая данные положительные цепочки и не порождающая отрицательные цепочки, будет бесполезна (если информационная последовательность не является полной). Естественно, хотелось бы, чтобы грамматика *предсказывала*, какие новые цепочки возможны, а какие не должны порождаться. В приведенных тривиальных грамматиках либо обобщение отсутствует, либо

выполнено чрезмерное обобщение, и, как следствие, теряется предсказательная сила.

Прежде чем строго сформулировать проблему восстановления грамматик как проблему обобщения, посмотрим, как работают эвристические методы. В отличие от методов восстановления грамматик перечислением, в индуктивных методах не осуществляется перебор грамматик. Вместо этого производится постепенное упрощение грамматики, совместимой с данным образцом языка. Отличие разных методов грамматического вывода заключается в тех эвристических правилах упрощения, которые используются в том или ином методе.

Рассмотрим типичный эвристический метод грамматического вывода на следующем классическом примере положительного образца:

$$S_7 = \{caaab, bbaab, caab, bbab, cab, bbb, cb\}.$$

На первом шаге формируются нетерминальные символы и правила таким образом, чтобы они порождали исходные цепочки. Для первой цепочки можно ввести такие правила:

$$caaab: S \to cA_1, A_1 \to aA_2, A_2 \to aA_3, A_3 \to aA_4, A_4 \to b.$$

На основе приведенных правил может быть сформирована цепочка $caaabb$ и только она. Аналогичным образом вводятся правила подстановки для следующих цепочек. Для второй цепочки это будут правила:

$$bbaab: S \to bA_5, A_5 \to bA_6, A_6 \to aA_7, A_7 \to aA_7, A_8 \to b.$$

Для третьей цепочки первое правило должно было бы иметь вид $S \to cA_7$, но уже на первом шаге эвристического алгоритма может производиться упрощение грамматики. Это упрощение заключается в том, что вместо введения нового нетерминального символа и нового правила подстановки используется уже введенный символ A_1 с соответствующим правилом $S \to cA_1$. Тогда для порождения третьей цепочки потребуются правила:

$$caab: S \to cA_1, A_1 \to aA_2, A_2 \to aA_3, A_3 \to b,$$

из которых новым будет только правило $A_3 \to b$.

Для оставшихся цепочек формируются следующие правила:

$$bbab: S \to bA_5, A_5 \to bA_6, A_6 \to aA_7, A_7 \to b;$$

$$cab: S \to cA_1, A_1 \to aA_2, A_2 \to b;$$

$$bbb: S \to bA_5, A_5 \to bA_6, A_6 \to b;$$

$$cb: S \to cA_1, A_1 \to b.$$

С учетом повторяющихся правил получаем следующую систему правил:

$$S \to cA_1, S \to bA_5 \quad A_1 \to aA_2, A_1 \to b \quad A_2 \to aA_3, A_2 \to b$$

$$A_3 \to aA_4, A_3 \to b \quad A_4 \to b \quad A_5 \to bA_6$$

$$A_6 \to aA_7, A_6 \to b \quad A_7 \to aA_7, A_7 \to b \quad A_8 \to b$$

На втором шаге алгоритма осуществляется редукция (упрощение) сложной грамматики на основе некоторых эвристических правил. Рассмотрим такое правило: объединить (отождествить) такие

нетерминальные символы A и B, которые входят в правила подстановки вида $A \to \alpha$, $B \to \alpha$. Под объединением символов A и B понимается такая процедура, при которой вводится новый нетерминальный символ C, и все вхождения символов A и B заменяются этим символом. После объединения оба правила $A \to \alpha$, $B \to \alpha$ приобретают вид $C \to \alpha$, то есть число правил уменьшается (возможно также, что и некоторые другие правила после замены каждого из символов A и B на символ C будут тождественными).

Поскольку в построенной грамматике имеются правила подстановки

$$A_1 \to b, A_2 \to b, A_3 \to b, A_4 \to b, A_6 \to b, A_7 \to b, A_8 \to b,$$

символы $A_1, A_2, A_3, A_4, A_6, A_7, A_8$ объединяются. Заменим все их вхождения символом C. Обычно в процессе грамматического вывода объединение символов осуществляется попарно, но здесь для сокращения записи мы воспользуемся объединением сразу семи символов. После такого объединения в грамматике останутся следующие правила:

$$S \to cC, S \to bA_5 \quad C \to aC, C \to b \quad A_5 \to bC.$$

В данном случае полученный результат является окончательным, но вполне могло оказаться так, что объединение некоторых нетерминальных символов привело бы к возможности объединения новых нетерминальных символов, и процесс редукции грамматики продолжился бы.

Сделаем некоторые замечания по данному примеру.

Рассмотренный грамматический вывод начинается с *ad hoc* грамматики, совместимой с заданным образцом и порождающей наиболее узкий язык, содержащий только предложения из образца. На каждом шаге редукции грамматики порождаемый язык расширяется (или, по крайней мере, не сужается), то есть происходит постепенное обобщение на основе образца языка. В случае попарного объединения нетерминальных символов смысл этого обобщения прост: классы символов или грамматические категории, которые обозначаются некоторыми нетерминальными символами, объединяются с целью формирования более общих классов и категорий. При этом в процессе вывода текущая грамматика все время остается совместимой с образцом. Эти моменты (использование *ad hoc* грамматики в качестве нулевого приближения и ее постепенное упрощение с сохранением совместимости с образцом) характерны для большинства эвристических алгоритмов грамматического вывода. Если на каждом шаге вывода производится обобщение, то возникает вопрос в правомерности такого обобщения при выполнении того или иного эвристического условия. Этот вопрос, в конечном счете, сводится к выработке корректного критерия в машинном обучении.

Еще одним аспектом эвристических методов грамматического вывода, требующим детального рассмотрения, является выбор операций по преобразованию грамматики в процессе ее редукции. Мы рассмотрели пример одной такой операции – операции по объединению двух нетерминальных символов при выполнении достаточно жесткого условия

(если есть правила вида $A \to \alpha$, $B \to \alpha$, то символы A и B следует объединить). Могут быть использованы и другие эвристические правила. Например, применяется такая эвристика: если есть правила $A \to ab$, $B \to aC$ и $C \to b$, то нетерминальный символ A может быть заменен символом C, а правило $A \to ab$ удалено. Использование разных наборов эвристик приводит к тому, что пути грамматического вывода оказывается различными. При этом нет гарантии, что разные пути вывода приведут к одному и тому же конечному результату – как и в случае градиентного спуска, алгоритм вывода может «застрять» в локальном минимуме. Выбор ограниченного числа эвристических приемов оказывается ненадежным.

Также обратим внимание на следующее. Исходная грамматика, строящаяся в приведенном примере, является очень частным видом грамматик, а именно автоматной грамматикой. При применении любого правила объединения символов тип грамматики не меняется (если быть более точным, то грамматика может из нерекурсивной превратиться в рекурсивную). Возникает необходимость введения таких операций по преобразованию грамматик, которые бы позволяли расширять тип грамматик (или приводили бы к правилам подстановки нового вида). Такой является, например, операция конструирования. При выполнении этой операции на основе правил вида $A \to \alpha B$ и $B \to \beta C$ формируется правило вида $A \to \alpha\beta C$; старые правила удаляются только в том случае, если при порождении образца языка они всегда применялись вместе.

Еще одна операция, приводящая к изменению вида правил подстановки, – это операция по удалению нетерминального символа: если символ A входит в левую часть только одного правила подстановки вида $A \to \alpha$, то все вхождения этого символа в других правилах могут быть заменены цепочкой α, а правило $A \to \alpha$ удалено. В нашем примере удаление символа A_5 приведет к системе правил $S \to cC, S \to bbC, C \to aC, C \to b$. При применении операций конструирования и удаления порождаемый язык не расширяется. При этом операция конструирования позволяет уменьшить сложность вывода, в то время как операция удаления – внутреннюю сложность грамматики, что еще раз ставит вопрос о введении адекватного критерия качества грамматик в задаче грамматического вывода. Приведенных операций все еще не хватает, чтобы выйти за рамки КС-грамматик (из-за этого и из-за сложности задачи синтаксического разбора для НС-грамматик при решении задачи грамматического вывода часто ограничиваются восстановлением КС-грамматик).

Итак, эвристические методы восстановления грамматик индукцией являются гораздо более эффективными с вычислительной точки зрения, чем методы восстановления грамматик перечислением. С другой стороны, эти методы не имеют строгого обоснования, так что при их использовании получение не только оптимального, но и сколько-нибудь приемлемого решения не гарантировано. Обоснованию подлежит как критерий

оптимальности гипотез, так и алгоритм поиска в пространстве гипотез (эвристические операции преобразования грамматик неявно задают пространство гипотез, которое при восстановлении грамматик перечислением описывалось в явном виде).

Байесовский вывод стохастических грамматик

Корректный критерий качества грамматики может быть найден в рамках байесовского подхода. Здесь критерий качества некоторой грамматики G по отношению к данному образцу S_t – это ее апостериорная вероятность $P(G \mid S_t) \propto P(G)P(S_t \mid G)$.

Считая предложения в образце независимыми, имеем

$$P(S_t \mid G) = \prod_{i=1}^{t} P(\alpha_i \mid G). \qquad (97)$$

Для обычных порождающих грамматик величины $P(\alpha_i \mid G)$ являются неопределенными, однако в рамках статистического подхода естественно привлечь стохастические грамматики, в которых каждому правилу подстановки приписывается некоторая вероятность. Вероятность порождения цепочки α_i (однозначной) стохастической грамматикой G определяется через произведение вероятностей правил подстановки, использующихся при выводе этой цепочки.

Априорные вероятности грамматик $P(G)$ в рамках чисто статистических методов не определяются. Для их задания привлекается внутренняя сложность грамматики, определенная тем или иным образом. Отметим, что правдоподобие $P(S_t \mid G)$ связано со сложностью вывода образца языка в рамках данной грамматики. Таким образом, в байесовском подходе к восстановлению грамматик разрешается дилемма, минимизировать ли внутреннюю сложность грамматики или сложность вывода.

Апостериорная вероятность как критерий качества грамматики может использоваться при восстановлении грамматик как перечислением, так и индукцией. И в том, и в другом случае алгоритмы могут быть легко модифицированы с целью использования байесовского критерия качества, но для этого необходимо для правил вывода уметь назначать вероятности. Это легко делается на основе подсчетов числа применений каждого из правил при выводе предложений образца. При этом максимизируется правдоподобие образца.

Пример. Рассмотрим уже знакомый нам образец языка

$$S_7 = \{caaab, bbaab, caab, bbab, cab, bbb, cb\}$$

и грамматику, содержащую правила

$$S \to cC, S \to bA \quad C \to aC, C \to b \quad A \to bC.$$

Запишем вывод каждой цепочки образца:

$$S \overset{S\to cC}{\Rightarrow} cC \overset{C\to aC}{\Rightarrow} caC \overset{C\to aC}{\Rightarrow} caaC \overset{C\to aC}{\Rightarrow} caaaC \overset{C\to b}{\Rightarrow} caaab$$

$$S \overset{S\to bA}{\Rightarrow} bA \overset{A\to bC}{\Rightarrow} bbC \overset{C\to aC}{\Rightarrow} bbaC \overset{C\to aC}{\Rightarrow} bbaaC \overset{C\to b}{\Rightarrow} bbaab$$

$$S \overset{S\to cC}{\Rightarrow} cC \overset{C\to aC}{\Rightarrow} caC \overset{C\to aC}{\Rightarrow} caaC \overset{C\to b}{\Rightarrow} caab$$

$$S \overset{S\to bA}{\Rightarrow} bA \overset{A\to bC}{\Rightarrow} bbC \overset{C\to aC}{\Rightarrow} bbaC \overset{C\to b}{\Rightarrow} bbab$$

$$S \overset{S\to cC}{\Rightarrow} cC \overset{C\to aC}{\Rightarrow} caC \overset{C\to b}{\Rightarrow} cab$$

$$S \overset{S\to bA}{\Rightarrow} bA \overset{A\to bC}{\Rightarrow} bbC \overset{C\to b}{\Rightarrow} bbb$$

$$S \overset{S\to cC}{\Rightarrow} cC \overset{C\to b}{\Rightarrow} cb$$

Далее произведем подсчеты применений правил:

$$S \to cC : 4; S \to bA : 3; C \to aC : 9, C \to b : 7; A \to bC : 3.$$

Откуда получим условные вероятности:

$$P(cC \mid S) = 4/7; P(bA \mid S) = 3/7; P(aC \mid C) = 9/16; P(b \mid C) = 7/16; P(bC \mid A) = 1.$$

Теперь несложно определить правдоподобие образца языка в рамках данной грамматики и сравнить его с правдоподобием образца в рамках других грамматик. Очевидно, правдоподобие данных в рамках *ad hoc* грамматики будет заметно выше. В связи с этим без учета априорных вероятностей грамматик обойтись невозможно.

При обсуждении байесовского подхода к восстановлению грамматик перечислением нельзя не отметить следующий прием. При выполнении перечисления грамматик лучшая найденная грамматика G_0, совместимая с образцом, накладывает ограничение $q = P(G_0)P(S_t \mid G_0)$ на априорные вероятности грамматик, которые имеет смысл рассматривать далее. Действительно, так как для любой грамматики верно $P(G)P(S_t \mid G) < P(G)$, то грамматика с априорной вероятностью $P(G) < q$ будет заведомо хуже уже найденной грамматики. В случае перечисления грамматик в порядке убывания их априорных вероятностей (что соответствует оккамовскому перечислению) этот прием позволяет установить надлежащий критерий остановки перебора. При этом параметр q подобен параметру α, а априорная вероятность $P(G)$ – оптимистической оценке качества в алгоритме альфа-отсечения (хотя аналогия не полная).

Грамматический вывод на основе принципа МДО

Рассмотрим применение принципа МДО для грамматического вывода в случае текстового представления. Информационная последовательность или образец языка являются исходными данными D. Грамматика выступает в роли генеративной модели этих данных. Чтобы описать

конкретные данные с помощью такой модели, необходимо для каждого предложения указать последовательность применения правил вывода.

Представим, что есть отправитель и получатель сообщения, содержащего закодированный образец языка. Вместо того чтобы передавать сами предложения, передаются порождающие их последовательности грамматических правил (как описание данных в рамках модели). Будем считать, что в генеративной модели при порождении предложений применение последующих правил не зависит от того, какие правила были применены до этого. Тогда каждое грамматическое правило кодируется с помощью уникального кодового слова некоторой длины. Выбор кодовых слов произволен. С точки зрения принципа МДО, этот выбор должен осуществляться таким образом, чтобы минимизировать общую длину сообщения.

Рассмотрим пример

$$S_8 = \{aaaaaaaba\ , aabbbbbbab\ a, aabbbbaba\ , aaaba\ ,$$
$$bbbbbbbbab\ a, bbbbbbaba\ , bbbbaba\ , bbaba\}$$

и рассмотрим в качестве модели грамматику, включающую правила

$$G_1: \quad \begin{array}{l} \Pi_{S1}: S \to A, \Pi_{S2}: S \to B, \\ \Pi_{A1}: A \to aaA, \Pi_{A2}: A \to bbB, \Pi_{A3}: A \to aaC, \\ \Pi_{B1}: B \to bbB, \Pi_{B2}: B \to bbC, \Pi_C: C \to aba. \end{array}$$

Вместо передачи последовательности S_8, отправитель передает цепочки правил:

$$\Pi_{S1}\Pi_{A1}\Pi_{A1}\Pi_{A3}\Pi_C, \Pi_{S1}\Pi_{A1}\Pi_{A2}\Pi_{B1}\Pi_{B2}\Pi_C,$$
$$\Pi_{S1}\Pi_{A1}\Pi_{A2}\Pi_{B2}\Pi_C, \Pi_{S1}\Pi_{A3}\Pi_C,$$
$$\Pi_{S2}\Pi_{B1}\Pi_{B1}\Pi_{B1}\Pi_{B2}\Pi_C, \Pi_{S2}\Pi_{B1}\Pi_{B1}\Pi_{B2}\Pi_C,$$
$$\Pi_{S2}\Pi_{B1}\Pi_{B2}\Pi_C, \Pi_{S2}\Pi_{B2}\Pi_C$$

Поскольку в данном примере в любой цепочке всегда присутствует только один нетерминальный символ (если вывод еще не окончен), на каждом шаге вывода могут быть применены только те правила, у которых текущий нетерминальный символ стоит в левой части. Значит, для каждой группы правил можно использовать собственный префиксный код.

Пусть $\Pi_{S1} = 0, \Pi_{S2} = 1, \Pi_{A1} = 0, \Pi_{A2} = 10, \Pi_{A3} = 11, \Pi_{B1} = 0, \Pi_{B2} = 1,$ $\Pi_C = \lambda$. Тогда закодированные цепочки правил будут иметь вид

$$00011,001001,00101,011,10001,1001,101,11,$$

что заметно короче, чем S_8. По каждому коду можно однозначно восстановить цепочку символов, зная, что вывод начинается с начального символа S.

Отметим, что использование таких кодов, как, например, $\Pi_{A1} = 10, \Pi_{A2} = 0, \Pi_{A3} = 11$ привело бы к большей длине сообщения. Минимизация длины передаваемого сообщения посредством выбора оптимальных кодовых слов для правил подстановки, очевидно,

соответствует построению кодов Хаффмана для каждой группы правил. Таким образом, при привлечении принципа МДО естественным образом возникает эквивалент стохастических грамматик (каждое правило кодируется кодом, длина которого зависит от частоты применения правила).

Сам код при этом строить не обязательно – достаточно воспользоваться оценкой длины кодового слова как минус логарифмом от частоты его встречаемости в сообщении. Для правила $\alpha \to \beta$ это величина $P(\beta \mid \alpha)$. Тогда длину описания некоторого предложения α посредством грамматики G можно оценить как $-\log_2 P(\alpha \mid G)$. Отметим, что разделение предложений, присутствующих в сообщении, осуществляется автоматически: как только новое считанное правило приводит к терминальному предложению, следующее правило трактуется как начало следующего вывода.

Суммарная длина описания образца S_t в рамках грамматики G будет выражаться через минус логарифм правдоподобия $P(S_t \mid G) = \prod_{i=1}^{t} P(\alpha_i \mid G)$. Этот результат уже был получен для байесовского подхода. Однако теоретико-информационный подход позволяет лучше понять некоторые моменты. Во-первых, становится ясным, почему при грамматическом выводе стохастические грамматики использовать предпочтительнее. Выше уже отмечалось, что при применении теоретико-информационного подхода к восстановлению грамматик вероятности применения правил возникают с необходимостью, так как передается не последовательность самих правил, применение которых дает искомое предложение, а их коды, длина которых соответствует частоте встречаемости правил. Длинное правило может применяться гораздо чаще, чем короткое, и код его будет короче. Становится понятно, почему рассматриваются условные вероятности – достаточно передавать лишь информацию о том, какое правило на текущем этапе следует выбрать среди правил, которые могут быть применены (если на данном шаге вывода применимо лишь одно правило, то его можно вообще не описывать).

Во-вторых, несколько расплывчатое определение условных вероятностей правил становится более четким через длину кода: правила должны кодироваться так, чтобы по цепочке этих кодов можно было однозначно восстановить исходную цепочку. Если грамматика допускает построение структурных деревьев, и последовательность, в которой заменяются нетерминальные символы, не имеет значения, то можно считать, что всегда заменяется самый левый нетерминальный символ. Если же такое ограничение не выполняется, то схема кодирования должна быть усложнена (например, помимо закодированного правила может оказаться необходимым передавать место в цепочке, к которому это правило применяется в процессе вывода).

И, в-третьих, в рамках теоретико-информационного подхода проще штрафовать сложность грамматики, связанную с ее априорной вероятностью. Действительно, чтобы получатель сообщения был способен расшифровать закодированный образец языка, он также должен получить и описание грамматики, с помощью которой этот образец закодирован.

Описание грамматики должно включать:

- таблицу кодов для терминальных и нетерминальных символов;
- описание правил подстановки с помощью кодов символов;
- коды правил подстановок, использующиеся для описания информационной последовательности.

Коды символов формируются в соответствии с частотой использования каждого из символов в правилах вывода. Например, в рассмотренном выше примере символы встречаются следующее число раз: $S:2, A:5, B:5, C:3, a:6, b:7$. Символ '$\to$' не кодируется, так как в КС-грамматике левая часть правила подстановки всегда состоит из одного символа (однако для большей точности следовало бы учитывать необходимость кодирования длины каждого правила). Остается определенный произвол в способе описания правил грамматики, например, вместо записи вида $A \to \alpha_1, A \to \alpha_2, A \to \alpha_3 ...$ можно воспользоваться записью $A \to \{\alpha_1 \mid \alpha_2 \mid \alpha_3 ...\}$. В соответствии с частотами символов строится код Хаффмана, определяются длины таблицы кодов символов и описания правил подстановки. Для простоты можно считать все символы равновероятными, тогда длину описания собственно грамматики можно оценить как $L(G) \approx L_P \log_2 \left(|V_T| + |V_N| \right)$, где L_P – суммарное число символов, использованных при описании правил.

В нашем примере $L_P = 28, |V_T| + |V_N| = 6, L(G) \approx 72$ бита, в то время как длина описания самой информационной последовательности в рамках данной грамматики равна $L(S_8 \mid G) = 33$ бита. Если не учитывать слагаемые второго порядка малости (длины описания таблиц перекодировки символов и правил подстановки), то суммарную длину описания можно грубо оценить как $L(S_8 \mid G) + L(G) \approx 105$ бит.

Рассмотрим несколько альтернативных грамматик.

- *Ad hoc* грамматика будет содержать восемь равновероятных правил $S \to \alpha_i$. Каждое из восьми предложений будет кодироваться тремя битами, то есть $L(S_8 \mid G) = 24$ бита. С помощью трех символов (двух терминальных и одного начального) записываются восемь правил, содержащих 74 символа, то есть $L(G) \approx 74 \cdot \log_2 3 \approx 117$ бит. $L(S_8 \mid G) + L(G) \approx 141$.
- «Беспорядочная» грамматика содержит три правила $S \to aS, S \to bS, S \to \lambda$, которые полагаются равновероятными. Для их описания требуется $L(G) \approx 7 \cdot \log_2 3 \approx 11$ бит. Предложение длины

n порождается путем применения $n+1$ таких правил, то есть $L(S_8 \mid G) \approx 74 \cdot \log_2 3 \approx 117$ бит. $L(S_8 \mid G) + L(G) \approx 128$.

- *Ad hoc* и «беспорядочная» грамматики практически совпадают по результирующей длине описания, но в первом случае основной вклад вносится длиной описания грамматики, а во втором – длиной описания исходных примеров в рамках грамматики. Эти грамматики представляют два крайних случая: чрезмерная конкретизация и чрезмерное обобщение. Наилучшая грамматика должна быть компромиссом между этими двумя случаями.
- Приведенная в качестве примера грамматика G_1 может быть изменена без изменения языка, порождаемого этой грамматикой. В частности, может быть удален символ C и правило $C \to aba$, тогда изменятся два других правила: $\Pi_{A3} : A \to aaaba, \Pi_{B2} : B \to bbaba$. В данном случае $L(S_t \mid G)$ не изменяется, но изменяется $L(G)$. Если грубо оценивать $L(G) \approx L_P \log_2 \left(|V_T| + |V_N| \right)$, то величина $|V_T| + |V_N|$ при этом уменьшается, а множитель L_P может как уменьшаться, так и увеличиваться в зависимости от того, насколько часто удаляемый символ использовался в других правилах. В данном случае, удаление символа C приводит к уменьшению длины описания грамматики, то есть к ее улучшению. Но разве может быть иначе? Казалось бы, с точки зрения порождения языка, символ C бесполезен, так как он всегда заменяется одной и той же цепочкой символов, то есть не выполняет никакой конструктивной функции. Однако если представить, что во многих правилах встречается одна и та же (достаточно длинная) цепочка символов, то отведение под эту цепочку нового нетерминального символа приведет не только к уменьшению длины описания, но и к формированию новой грамматической категории, которой можно будет оперировать на более абстрактном уровне.
- Для приведенного примера можно предложить и более общую грамматику, порождающую более широкий язык. Примером такой грамматики может служить грамматика G_2: $S \to aaS, S \to bbS, S \to aba$. Для нее $L(G) \approx \log_2 \left(|V_T| + |V_N| \right) \cdot L_P$, т.е. $L(G) \approx 12 \log_2 3 \approx 19$. Поскольку $P(aaS \mid S) = 6/29$, $P(bbS \mid S) = 15/29$ и $P(aba \mid S) = 8/29$, имеем $L(S_8 \mid G) \approx 43$ бита. Хотя длина $L(S_8 \mid G)$ несколько увеличилась, эта грамматика оказывается предпочтительнее исходно рассмотренной за счет значительного уменьшения длины $L(G)$. Увеличение длины $L(S_t \mid G)$ обычно свидетельствует о расширении языка. Действительно, если первоначально порождался язык $(aa)^n (bb)^m aba$, то теперь в языке цепочки aa и bb могут произвольно чередоваться. $L(S_8 \mid G) + L(G) \approx 62$.

- Рассмотрим еще одну грамматику G_3: $S \to aaS, S \to B,\ B \to bbB,$ $B \to aba$. $L(G) \approx 14\log_2 4 = 28$. Для нее частоты правил будут $P(aaS \mid S) = 6/14, P(B \mid S) = 8/14,\ P(bbB \mid B) = 15/23,\ P(aba \mid B) = 8/23$. Тогда $L(S_8 \mid G) \approx 35$ бит. $L(S_8 \mid G) + L(G) \approx 63$. При учете длины описания таблицы перекодировки для правил вывода эта грамматика сравнивается с предыдущей грамматикой (с точностью до целых битов). В то же время эти две грамматики порождают весьма разные языки. Достоверный выбор между ними может быть сделан только путем расширения образца языка.
- Добавим в образец языка цепочку *aaaabbaba*. Для грамматики G_2 вероятности правил станут $P(aaS \mid S) = 8/33;\ P(bbS \mid S) = 16/33$ $P(aba \mid S) = 9/33$, а $L(S_9 \mid G_2) \approx 50$. Вероятности правил грамматики G_3 изменятся таким образом: $P(aaS \mid S) = 8/17, P(B \mid S) = 9/17,$ $P(bbB \mid B) = 16/25, P(aba \mid B) = 9/25,$ а $L(S_9 \mid G_3) \approx 40{,}5$. Таким образом, грамматика G_3 станет несколько предпочтительнее, чем G_2. Различие между ними будет и дальше увеличиваться при добавлении новых предложений в образец языка, даже если совместимость этих грамматик с образцом не будет нарушена.

Принцип МДО позволяет ввести критерий качества грамматики при данном образце языка. Этот критерий может использоваться как при восстановлении грамматик перечислением, так и в эвристических процедурах грамматического вывода. В последнем случае эвристические правила по преобразованию грамматик могут быть заменены более строгим анализом. Например, ранее были упомянуты следующие условия слияния нетерминальных символов:

- если есть продукции вида $A \to \alpha$, $B \to \alpha$, то символы A и B должны быть объединены;
- если есть продукции вида $A \to ab$, $B \to aC$ и $C \to b$, то символы A и C должны быть объединены.

Эти и другие условия могут быть заменены одним:

- если в результате слияния нетерминальных символов уменьшается общая длина описания $L(S_t \mid G) + L(G)$, то эти символы должны быть объединены.

Простое применение эвристических правил (при достаточном их разнообразии) без проверки критерия качества получающейся при этом грамматики может приводить к чрезмерному обобщению. Представим, например, грамматику, содержащую (помимо прочих) продукции: $A \to до$, $A \to от$, $A \to перед$ и $C \to до$, $C \to ми$, $C \to ля$. Поскольку имеются продукции $A \to до$ и $C \to до$, символы A и C в каком-либо эвристическом методе грамматического вывода могут быть объединены. Однако видно, что символы A и C могут трактоваться как различные категории, которые лишь частично перекрываются по включенным в них цепочкам символов.

При решении вопроса об объединении нетерминальных символов на основе длины описания учитываются все правила, в которые эти символы входят, а также учитывается то, сколько раз эти правила использовались при порождении образца языка.

Помимо операции объединения нетерминальных символов могут использоваться и другие операции преобразования грамматик. Выше отмечалась возможность использования таких операций, как удаление символа и конструирование. Эвристические правила, устанавливающие условия применения той или иной операции, могут быть существенно ослаблены (полное их удаление на практике приводит к существенному замедлению процедуры поиска оптимальной грамматики), если окончательное решение о применении правила принимается на основе критерия длины описания. Вопрос о выборе самих операций все еще остается открытым.

Итак, при восстановлении грамматик индукцией исходно конструируется некоторая *ad hoc* или «беспорядочная» грамматика, которая заведомо совместима с образцом языка. Далее производится последовательное улучшение этой грамматики: на каждом шаге выбирается такая операция, применение которой дает наибольший прирост критерия качества.

Вопросы и упражнения

1. В чем отличие в постановке задачи восстановления грамматик при информаторном и текстуальном представлениях?
2. Какое требование ставится к грамматике при ее восстановлении перечислением?
3. В чем основной недостаток методов восстановления грамматик перечислением?
4. В связи с чем при восстановлении грамматик предпочтение отдается наиболее коротким грамматикам?
5. В чем принципиальное отличие постановки восстановления грамматик индукцией по сравнению с восстановлением грамматик перечислением?
6. Какие операции по преобразованию грамматик используются при их восстановлении индукцией?
7. Грамматики какого типа получаются при восстановлении грамматик индукцией, если используется только операция объединения нетерминальных символов?

16. Автоматическое построение наборов правил и деревьев решений

Построение наборов правил

В продукционных системах основным элементом представления знаний является правило вида $A \Rightarrow B$, которое зачастую описывает такие

связи, как <условие>–<действие> или <причина>–<следствие>. Такого типа связи могут устанавливаться на основе опыта. Иными словами, построение набора правил может выступать в качестве задачи машинного обучения. Продукционные системы могут конструироваться вручную при участии людей-экспертов. Такой подход часто применяется при разработке экспертных систем, например, в целях медицинской диагностики. Однако использование методов машинного обучения для автоматического построения наборов правил по примерам представляется более интересным с точки зрения искусственного интеллекта. Оно является также более удобным на практике (хотя и имеет определенные ограничения). Например, вместо того, чтобы просить врачей-экспертов в явном виде формулировать правила <симптомы>–<болезнь>, можно использовать данные по ранее поставленным диагнозам для автоматического формирования соответствующих правил. Поставим задачу автоматического построения набора правил по примерам более строго.

Пусть есть дискретное пространство признаков $\mathrm{X} = X_1 \times X_2 \times ... \times X_N$, где каждое из множеств X_i содержит конечное число элементов. В задаче построения набора правил признаки часто называются *атрибутами* (для обозначения переменных, множество значений которых конечно, также можно встретить название «категориальные переменные»). Пусть также дано конечное множество классов $\mathrm{A} = \{a_1, a_2, ..., a_d\}$, где d – количество классов. По обучающей выборке $D = ((x_1, c_1), (x_2, c_2), ..., (x_M, c_M))$, где $x_i \in \mathrm{X}, c_i \in \mathrm{A}$, требуется построить классификационный алгоритм, который бы для нового объекта по его атрибутам предсказывал номер класса, к которому он относится. Выходной класс может соответствовать как некоторому действию, так и отдельному концепту. В последнем случае обычно говорят о задаче *изучения понятий* или *обучения концептам* (concept learning).

Как видно, мы приходим к формулировке классической задачи распознавания образов (случай обучения с учителем). Отличие, однако, заключается в том, что в рассмотренных нами дискриминантных методах распознавания признаки считались вещественными величинами, в то время как в данном случае атрибуты являются дискретными. Таким образом, наборы порождающих правил (как и деревья решений) могут использоваться как средство решения задачи распознавания с учителем в дискретном пространстве признаков.

Весьма интересен частный случай, при котором все величины являются бинарными. Такое ограничение ведет к логическим методам распознавания (или к задаче восстановления пропозициональных формул). Отличие логических методов распознавания в том, что антецеденты (левые части правил) содержат не просто перечень значений атрибутов, при которых эти правила могут быть применены, но представляют собой логические выражения, в которых используются операторы «И», «НЕ»,

«ИЛИ» и т.д., например, $X_1 \,\&\, X_2 \vee \neg X_1 \,\&\, \neg X_2 \Rightarrow a$. Мы, однако, такой способ описания условий рассматривать не будем.

Формальные грамматики предоставляют более широкие возможности задания исходных описаний объектов и более широкое пространство моделей, поскольку допускает произвольные структурные взаимосвязи между атрибутами. Здесь же атрибуты оказываются в определенном смысле независимыми: изучаемые концепты инвариантны по отношению к порядку атрибутов. Задачу построения набора правил (в приведенной выше постановке) можно свести к задаче восстановления грамматики, однако для многих задач наборы правил являются более естественным (и более простым) представлением.

Сформулируем некоторые особенности представления концептов в виде набора правил. Для этого рассмотрим следующий пример. Пусть

- имеются следующие атрибуты:
  цвет: X_1={желтый=0, синий=1, зеленый=2, красный=3};
  форма: X_2={0=круглый, 1=квадратный, 2=вытянутый};
  размер: X_3={0=маленький, 1=большой};
  вес: X_4={0=легкий, 1=тяжелый};
  степень съедобности: X_5={0=съедобный, 1=несъедобный};
- даны классы:
  a_1=яблоко; a_2=мяч; a_3=гиря; a_4=банан;
- и даны правила:
  1. X_2=0 & X_5=0 → a_1;
  2. X_2=0 & X_4=1 → a_3; (98)
  3. X_2=0 → a_2;
  4. X_1=0 & X_2=2 & X_5=0 → a_4.

Отметим следующие особенности. Во-первых, количество атрибутов N может быть достаточно большим, в связи с чем явное описание допустимых значений всех атрибутов в левой части правила не используется. Вместо этого в условии указываются лишь те атрибуты, значения которых релевантны данному концепту. От остальных атрибутов происходит абстрагирование. Например, правило «X_2=0 & X_5=1 → a_1» говорит о том, что яблоко – это нечто круглое и съедобное независимо от цвета, веса и размера (для данного набора классов).

Во-вторых, в наборе правил важна последовательность применения правил, так как для одних и тех же значений атрибутов может найтись несколько подходящих правил с разными правыми частями. Например, третье правило (X_2=0 → a_2) классифицирует все круглые объекты как мяч, но, несмотря на это, если оно применяется после первого и второго правил, то результат его применения будет корректным.

В-третьих, в наборе правил может не найтись ни одного правила, условие применения которого удовлетворяет данным значениям атрибутов. Из-за этого возникает новый вид ошибки, который не встречался в дискриминантных методах распознавания. Например, набор

атрибутов (X_1=2 & X_2=2 & X_5=0) не подпадет ни под одно правило. Этот вид ошибки называется *пропуском*.

В задаче построения набора правил, как и в других задачах машинного обучения, одной из центральных проблем является проблема критерия качества, в соответствии с которым можно было бы выбирать лучшее решение. Часто в качестве целевой функции используется размер набора правил. При этом сначала строится такой набор, который просто повторяет данные примеры: $\{x_i \to c_i\}_{i=1}^{M}$, и его необходимо максимально упростить, сохранив при этом корректную классификацию обучающих примеров. Этот классический подход основывается на том эмпирической факте, что минимальный набор правил, совместимый с примерами, является наиболее полезным.

Смысл этой полезности вполне очевиден в рамках принципа минимальной длины описания. Наименьший набор правил, совместимый с обучающей выборкой, обладает наибольшей предсказательной силой (с наибольшей вероятностью предсказывает истинные классы для примеров, не вошедших в выборку). Упрощение правил соответствует их обобщению. Поясним это на очень простом примере.

Пусть в выборке даны следующие примеры:

круглый, легкий, зеленый → мяч

круглый, легкий, синий → мяч

круглый, легкий, красный → мяч

Упрощение этих правил может дать правило «круглый, легкий → мяч», совместимое со всеми примерами. Упрощенное правило не только заметно короче трех исходных правил, но также позволяет корректно классифицировать такой пример, как «круглый, легкий, желтый», который до этого ни под одно правило не подпадал и, что принципиально, в исходном наборе правил считался бы пропуском.

На практике интерес представляет случай зашумленных данных, то есть таких выборок, примеры в которых могут содержать ошибки. Обучение правилам – типичное обучение с учителем. Такое обучение может происходить по примерам, полученным от разных людей (например, экспертов-медиков, ставящих разные диагнозы при похожих симптомах; или людей, выполняющих различные действия в одинаковых ситуациях). При этом возникает необходимость обобщения. Таким образом, выводимый набор правил должен не просто минимизироваться по размеру, но в этом процессе должно также учитываться количество примеров-исключений, не соответствующих правилам.

Вполне очевидно, что здесь может быть применен принцип МДО. Исходя из этого принципа, наилучшим (при имеющихся данных) является тот набор правил, для которого минимизируется сумма

$$DL = DL_{rules} + DL_{exceptions}, \qquad (99)$$

где DL_{rules} – длина описания набора правил, а $DL_{exceptions}$ – длина описания исключений (ошибок).

Для более аккуратного анализа задачи удобно предполагать существование отправителя и получателя сообщения. При обучении с учителем, как правило, полагается, что в сообщении отправитель передает информацию только о правых частях обучающих примеров, в то время как левые части получателю известны априори (такая постановка часто встречается в задачах обучения с учителем). Формула вида (99) как раз и соответствует передаче сообщения, несущего информацию о правых частях примеров обучающей выборки.

Рассмотрим слагаемые более подробно для случая, когда число классов равно двум (примеры, относящиеся к разным классам, при этом часто называются положительными и отрицательными).

Сначала рассмотрим кодирование ошибок (исключений) для определения слагаемого $DL_{exceptions}$. Ошибки могут быть двух типов. Во-первых, применение правил может приводить к неверной классификации некоторых примеров. Чтобы получатель мог восстановить правильные значения отсутствующих правых частей, ему должен быть передан список исключений (в сообщении просто необходимо указать, какие именно примеры классифицированы неправильно, так как количество классов равно двум). Чтобы закодировать индексы M_{FP} неверно классифицированных примеров из всех M примеров, необходимо примерно $\log_2 C_M^{M_{FP}}$ бит информации.

Поясним это выражение чуть подробнее. Всего существует $C_M^{M_{FP}}$ сочетаний из M примеров по M_{FP} примеров. Если пронумеровать все возможные сочетания в лексикографическом порядке, то для указания номера конкретного сочетания, включающего M_{FP} примеров, потребуется $\log_2 C_M^{M_{FP}}$ бит информации. Для однозначного выбора требуется также указать количество примеров M_{FP}, для чего необходимо примерно $\log_2 M_{FP}$ бит. Последнее слагаемое обычно игнорируют, так как оно существенно меньше первого (за исключением случая $M_{FP} \approx M$).

Как отмечалось ранее, некоторые примеры могут не подпадать ни под одно из правил. Такие примеры – это второй тип ошибок, обозначаемых как пропуски и требующих отдельного кодирования. Является ли некоторый пример пропуском, получатель может легко определить, имея соответствующий набор правил. В связи с этим отправитель должен лишь закодировать информацию о том, какие из пропусков должны быть классифицированы как положительные, а какие – как отрицательные (для случая двух классов d=2). Пусть M_{NS} – число пропусков – примеров, не покрытых набором правил, и пусть M_{FN} – количество пропусков, которые должны быть классифицированы как положительные. Тогда для их

указания требуется порядка $\log_2 C_{M_{NS}}^{M_{FN}}$ бит информации. Отсюда можно получить следующую формулу (впервые предложенную Квинланом):

$$DL = DL_{rules} + \log_2 C_M^{M_{FP}} + \log_2 C_{M_{NS}}^{M_{FN}} . \qquad (100)$$

Длина описания правил DL_{rules} часто берется просто пропорциональной суммарной длине правил, однако может использоваться более строгая оценка этой длины, которая будет рассмотрена позднее.

Иногда отмечаются следующие определенные недостатки формулы (100):

1. Формула для вычисления длины описания исключений симметрична, то есть набор правил, неправильно классифицирующий все примеры, будет иметь такую же длину описания исключений, равную нулю, что и набор правил, правильно классифицирующий все примеры.
2. Если количество положительных примеров существенно больше (существенно меньше) количества отрицательных примеров, то выведенный набор правил будет иметь тенденцию обобщать в недостаточной (избыточной) степени изучаемый концепт, в особенности в присутствии шума или при обучающей выборке малого объема.
3. При обучении по большой выборке примеров в присутствии шума все еще остается тенденция следовать этому шуму (создавать на основе исключений дополнительные правила).

На самом деле, первый недостаток таковым не является. Действительно, какое-то понятие может быть выучено как отрицание другого понятия. Например, понятие «нечто, не являющееся яблоком» сложно выучить через описание всех атрибутов, которые могут присутствовать у не-яблок. Также следует отметить, что симметрия несколько нарушается, если при выводе формулы (100) быть более аккуратными и не опускать слагаемые $\log_2 M_{FP}$ и $\log_2 M_{NP}$, которыми обычно можно пренебречь, но именно для этого крайнего случая они имеют видимый эффект. В результате описание отрицания некоторого концепта оказывается несколько длиннее, чем описание самого концепта, что вполне соответствует интуитивному пониманию подобного рода ситуаций.

Для устранения следующих двух недостатков может быть предложена более изощренная (хотя все еще простая с вычислительной точки зрения) схема кодирования. Обратим внимание, что в формуле (100) кодирование ошибок, сделанных при применении всех правил, осуществляется совместно, так как отправителем передается номер сочетания из M примеров по M_{FP} неверно классифицированных примеров. Раздельно кодируя ошибки, допущенные при применении разных правил, мы в общем случае уменьшим длину описания. Это подтверждается на практике, а также очевидно из общих соображений: некоторые правила могут не иметь исключений, в то время как другие правила могут иметь

много исключений; разумно описывать исключения для таких правил независимо. Для примеров, не покрытых набором правил, можно ввести дополнительное правило с пустой левой частью (условием) и негативной правой частью (она может быть взята и положительной – это не имеет значения). Такое правило должно находиться в самом конце набора и применяться после проверки условий других правил. Это обеспечивает более единообразную схему кодирования как правил, так и ошибок. Тогда общая длина описания составит

$$DL = \sum_i (L(rule_i) + L(err_i)). \qquad (101)$$

Длина описания ошибок для каждого из правил может вычисляться следующим образом. Каждое правило выделяет некоторое подмножество примеров, которые под это правило подпадают. Пусть отправитель передает получателю закодированные правые части этих примеров. Для этого необходимо $L(err_i) = m_i H(e_i)$ бит информации, где m_i – число примеров, подпадающих под i-е правило, а $H(e_i)$ – энтропия, посчитанная на основе гистограммы правых частей примеров для данного правила.

Например, если под правило с условием X_2=0 & X_4=1 подпадает два примера, относящихся к классу a_2, и шесть примеров, относящихся к классу a_3, то длина описания ошибок $L(err_i) = -2\log_2 2/8 - 6\log_2 6/8 \approx 6.5$ бит. Если все примеры, подпадающие под данное правило, имеют одну и ту же правую часть, то, очевидно, $L(err_i) = 0$. Теперь становится более явным, почему раздельное описание ошибок для разных правил предпочтительнее.

Обратим внимание на следующее: при таком представлении не имеет значения, что именно стоит в правой части правила, так как кодируются любые значения правых частей (это становится наиболее явным, когда правые части могут принимать не два, а более значений). Тогда $L(err_i)$ принимает смысл не длины закодированных ошибок, а длины закодированных правых частей, а правила оказываются стохастическими, то есть допускающими разные правые части с разными вероятностями (аналогично правилам в стохастических грамматиках). Подобные правила исследуются отдельно, в чем есть смысл: неопределенность действия в какой-то ситуации может быть явно выражена в наборе правил. Таким образом, длины кодов для правых частей каждого из правил могут использоваться не только для оценки критерия качества набора правил, но и для последующей классификации, сообщая об апостериорном распределении вероятностей по классам для каждого нового примера.

Для кодирования самих правил также может быть использована более строгая схема. Длину описания каждого правила можно считать следующим образом. Условие применения правила – это набор тестов, выполняемых над атрибутами примера (правило «X_2=0 & X_4=1 → a_3» содержит два теста). Пусть условие i-го правила содержит L_i тестов. И пусть всего возможно N_{pt} различных тестов. Тогда для описания условия i-

го правила необходимо примерно $\log_2 C_{N_{pt}}^{L_i}$ бит информации (с учетом того, что порядок проверки тестов значения не имеет и описываться не должен).

Следует отметить, что одно слагаемое здесь также было пропущено – это длина таблиц перекодировок правых частей для подмножеств примеров каждого из правил. Действительно, если $L(rule_i)$ – только длина кодирования условия применения правила, а $L(err_i)$ – только оценка длины кодов Хаффмана для возможных правых частей правила, то у получателя не будет связи между закодированными по Хаффману правыми частями и их исходными значениями. Иными словами, необходимо передавать таблицу перекодировки, которая, по сути, является частью правила (ее длина не зависит от количества примеров) и описывает возможные правые части правил с указанием их вероятностей. Длину таблицы перекодировки можно грубо оценить как $L(t_i) = t_i \log_2 t_i$, где t_i – число различных правых частей в *i*-м правиле.

Чтобы получатель смог корректно декодировать набор правил, ему необходимо знать, где заканчивается сообщение. Иными словами, необходимо знать общее число правил. Для этого необходимо добавить еще одно небольшое слагаемое, которое грубо можно оценить как $\log_2 N_{rules}$, где N_{rules} – общее количество правил в наборе. Таким образом, окончательная формула примет вид

$$DL = \log_2 N_{rules} + \sum_i \left(\log_2 C_{N_{pt}}^{L_i} + L(t_i) + m_i H(e_i) \right). \qquad (102)$$

Часто некоторые слагаемые опускают (сознательно или нет), что приводит к менее точной оценке длины описания и некоторому снижению обобщающей способности метода построения набора правил. Также существует возможность различного рода модификаций схемы кодирования (именно так мы перешли от формулы (100) к формуле (102)), что приведет к смещению распределения априорных вероятностей.

Идя путем изменения схемы кодирования для получения более компактных описаний набора правил, можно прийти к другим представлениям информации, содержательно отличающимся от наборов правил. Обратим внимание на то, что в наборе многие условия применения разных правил могут содержать общие элементы. Возникает соблазн сжать описание правил таким образом, чтобы одинаковые блоки тестов не дублировались. Очевидно, такое сжатие набора правил приведет к тому, что похожие правила будут предпочтительнее, чем такое же количество разных правил. Иными словами, произойдет смещение распределения априорных вероятностей, что скажется на обобщающей способности набора (для многих, но не всех задач – положительным образом). Таким представлением, выступающим в роли альтернативы наборам правил, являются деревья решений.

Критерий качества для задачи построения дерева решений

В задаче построения дерева решений, как и в задаче построения набора порождающих правил, дано конечное множество атрибутов $\mathrm{X} = X_1 \times X_2 \times ... \times X_N$, где N – число атрибутов, и конечное множество классов $\mathrm{A} = \{a_1, a_2 ... a_d\}$, где d – число классов. Как и раньше, по обучающей выборке $D = ((x_1, c_1), (x_2, c_2), ..., (x_M, c_M))$, где $x_i \in \mathrm{X}, c_i \in \mathrm{A}$, требуется восстановить отображение (алгоритм), действующее из X в A. Задача построения деревьев решения отличается способом представления этого алгоритма. В данном случае классификационный алгоритм задается в виде дерева, в каждом узле которого осуществляется проверка одного из атрибутов, и в зависимости от значения этого атрибута происходит переход к соответствующему дочернему узлу (см. рис. 36). В листьях дерева решений располагаются номера классов, которые и выступают в качестве результата классификации. Узлы такого дерева также называются узлами принятия решений, а проверка значения атрибута в узле – тестовой процедурой.

Деревья решений были предложены Квинланом в качестве представления информации в рамках машинного обучения для решения задачи изучения понятий. Тем не менее, деревья решений, как и наборы правил, пригодны также для описания связи <условие>–<действие>. Эти два представления имеют одинаковую выразительную силу, хотя компактность описания одних и тех же понятий у них может быть весьма различной (ср. рис. 36 и набор правил 98). Помимо этого основное отличие деревьев решений от наборов правил заключается в том, что в первых проверка атрибутов осуществляется последовательно, в то время как во вторых – работа осуществляется сразу со всей совокупностью атрибутов.

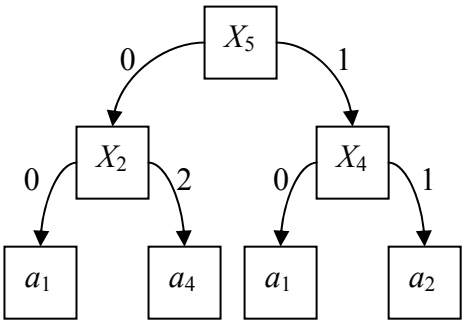


Рис. 36. Пример дерева решений, заменяющего систему правил (98). Здесь X_5 – степень съедобности (0-съедобный, 1-несъедобный); X_2 – форма (0-круглый, 2-вытянутый); X_4 – вес (0-легкий, 1-тяжелый); классы: a_1=яблоко; a_2=мяч; a_3=гиря; a_4=банан

Как следствие, возникают некоторые отличия в использовании этих подходов. В частности, деревья решений оказываются несколько более удобными для случая, при котором не все атрибуты доступны с самого начала и имеют разную сложность получения (в смысле материальных ресурсов). В качестве примера можно привести задачу геологической разведки (например, нефтяных месторождений). В этой задаче существуют

разные способы обнаружения месторождений, которые имеют разную стоимость и разную эффективность. Оказывается выгоднее сначала проверить более дешевые в получении, хотя и менее информативные признаки, например, связанные с сейсмическими характеристиками местности. Более надежные способы проверки такие, как бурение скважин, существенно дороже, и их следует использовать лишь после того, как вероятность наличия нефти повысилась в результате проверки других признаков. Похожим образом осуществляется диагностика и в других областях. Например, при поиске неисправности автомобиля в первую очередь осуществляются наиболее доступные тесты (например, проверка наличия бензина), даже если заранее известно, что эти тесты с высокой вероятностью не помогут выявить неисправность.

Если каждый атрибут обладает некоторой стоимостью, определяемой связанным с ним действием, то должно строиться такое дерево, которое минимизирует математическое ожидание затрат.

Если информация о затратах отсутствует, то есть атрибуты считаются в этом смысле равноправными, то минимизация затрат будет соответствовать минимизации размера дерева решений, его длины описания. Интересно, что минимальное дерево решений обеспечивает минимальные вычислительные затраты на классификацию новых примеров, то есть строится не только самый короткий, но и самый быстрый классификационный алгоритм (к сожалению, для более широких классов алгоритмов минимизация длины описания не совпадает с минимизацией вычислительных затрат).

Последовательная проверка атрибутов также полезна в задачах планирования, в которых значения некоторых атрибутов появляется лишь после принятия решения о выполнении действия, связанного с проверкой предыдущего атрибута. К примеру, упрощенная инструкция по переходу дороги в виде набора правил может формулироваться так:

Если есть работающий светофор, то подождать зеленый свет.

Если горит зеленый свет, то переходить дорогу.

Если нет работающего светофора, то посмотреть, есть ли машины.

Если машин нет, то переходить дорогу.

Видно, что в виде линейной системы правил такая информация представима хуже, чем в виде дерева (см. рис. 37), так как в последнем порядок, в котором измеряются значения атрибутов, задается более явно.

Таким образом, существуют задачи, в которых деревья решений предпочтительнее наборов правил, поэтому деревья решений заслуживают отдельного изучения. Мы рассмотрим задачу восстановления деревьев решений, в которой с атрибутами не связана стоимость получения их значения, и значения всех атрибутов известны заранее. Такая ситуация более характерна для задачи обучения концептам, чем для задачи планирования.

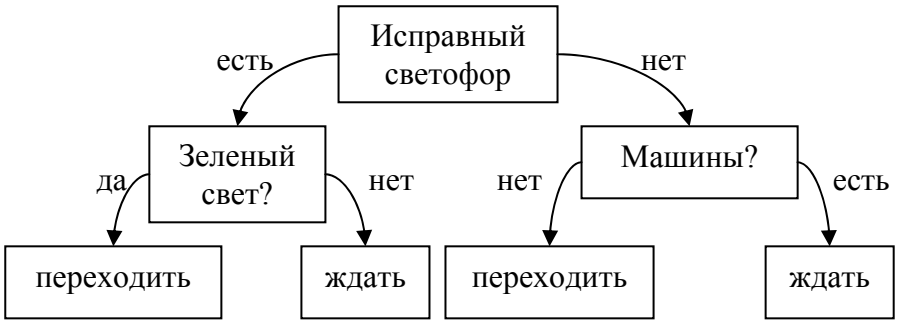


Рис. 37. Дерево решений, задающее порядок проверки атрибутов для упрощенного алгоритма перехода дороги

Дерево решений представляет собой модель, в которой должны отражаться закономерности в наблюдательных данных. Как и при использовании набора правил, здесь может быть применен принцип МДО. Сначала перепишем уравнение (100)

$$DL = DL_{tree} + DL_{exceptions}, \qquad (103)$$

где DL_{tree} – длина описания соответствующего дерева решений, а $DL_{exceptions}$ – как и раньше, длина описания исключений. Если вернуться к аналогии с отправителем и получателем сообщения, то видно, что здесь, как и раньше, полагается, что левые части примеров обучающей выборки получателю известны априори. Отправитель должен передать модель (дерево решений), связывающую значения атрибутов с выходными классами, а также отклонения примеров обучающей выборки от этой модели (исключения).

В простом случае производится поиск только среди деревьев, которые точно описывают исходные данные (корректно классифицируют все имеющиеся примеры). Тогда задача построения дерева решений сводится к поиску наипростейшего дерева, согласованного с обучающей выборкой, а сообщение должно включать лишь закодированное дерево решений.

Рассмотрим сначала вопрос о кодировании дерева решений с использованием классической схемы. Кодирование будем начинать с корня дерева и далее будем записывать информацию о каждом узле в порядке обхода дерева (например, обход будет в ширину). Для каждого узла записывается следующая информация:

- является ли узел конечным (0, 1);
- если узел не конечный, записывается атрибут, проверяемый в данном узле;
- если узел конечный, записывается номер выходного класса.

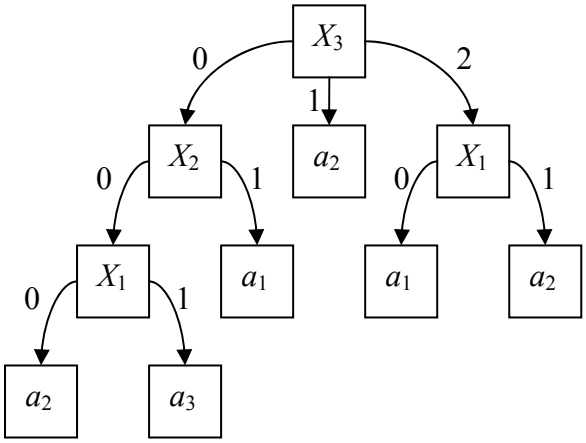


Рис. 38. Дерево решений, включающее проверку трех атрибутов X_1, X_2, X_3 и приводящее к выбору одного из трех классов a_1, a_2, a_3

На рис. 38 представлен пример дерева решений для трех атрибутов x_1, x_2 и x_3, первые два из которых являются бинарными, а третий может принимать три значения. Число классов равно трем: $\mathrm{A}=\{a_1,a_2,a_3\}$. Покажем по шагам, как будет формироваться описание этого дерева в виде строки:

Корень: $1x_3$
Первый уровень: $1x_20a_21x_1$
Второй уровень: $1x_10a_10a_10a_3$
Третий уровень: $0a_20a_3$
Полное описание: $1x_31x_20a_21x_11x_10a_10a_10a_30a_20a_3$

Это описание сжимаемо. Первый символ в описании узла указывает, является ли узел листом. Пусть n_0 – это количество листьев (в которые записаны результирующие классы), а n_1 – это число узлов, в которых осуществляется выбор. В нашем примере n_0=6 и n_1=4. Тогда требуется $-\log_2[n_0/(n_0+n_1)]$ бит для указания листа и $-\log_2[n_1/(n_0+n_1)]$ бит для указания неконечного узла. Для всех узлов на указание того, являются ли они листьями или нет, потребуется

$$-n_0\log_2[n_0/(n_0+n_1)]-n_1\log_2[n_1/(n_0+n_1)] \text{ бит.}$$

Общую суммарную длину этого элемента описания узлов можно грубо оценить, как n_0+n_1 бит (эта оценка точна, когда в дереве одинаковое количество узлов и листьев n_0=n_1). Иногда этой оценкой и ограничиваются.

Поскольку листья могут содержать только номера классов, то для их описания требуется $n_0H(\mathrm{A})$ (без учета длины таблицы перекодировки), где энтропия $H(\mathrm{A})=-\sum_{a\in\mathrm{A}}P(a)\log_2 P(a)$, $P(a)$ – вероятность того, что в листе содержится указание на выбор класса a. В нашем примере $P(a)=1/3$ для каждого из классов. Для упрощенной схемы кодирования в предположении о равновероятном появлении классов получаем $H(\mathrm{A})=\log_2 d$, где d – общее число классов.

Также неконечные узлы могут содержать только указание на проверяемый атрибут. Для описания всех таких узлов требуется $-n_1 \sum_{i=1}^{N} P_i \log_2 P_i$, где P_i – частота проверки *i*-го атрибута в дереве решений. В нашем примере $P_1 = 2/4$, $P_2 = P_3 = 1/4$. Для упрощенной схемы кодирования в предположении $(\forall i, j) P_i = P_j$ это слагаемое будет равно $\log_2 N$.

Таким образом, длина описания дерева решений оценивается как

$$DL_{tree} = n_0 \left[-\log_2 \frac{n_0}{n_0 + n_1} + H(\mathrm{A}) \right] + n_1 \left[-\log_2 \frac{n_1}{n_0 + n_1} - \sum_{i=1}^{N} P_i \log_2 P_i \right]. \quad (104)$$

В случае упрощенной схемы кодирования результирующая формула примет вид

$$DL_{tree} = (n_0 + n_1) + n_0 \log_2 d + n_1 \log_2 N. \quad (105)$$

В нашем примере длина описания дерева решений, вычисленная по формуле (104), примерно равна 25 битам, а вычисленная по приближенной формуле (105), – 26 бит. Формула (105) используется чаще благодаря своей простоте, однако в более сложных случаях она может давать заметное отклонение от более точной формулы (104), что может привести к выбору менее предпочтительного дерева решений.

Длина описания дерева решений не содержит явной зависимости от размера обучающей выборки. Если дерево решений правильно предсказывает классы для всех обучающих примеров, то выражение (104) является целевой функцией, которую нужно минимизировать выбором подходящего дерева решений.

Данное представление информации можно расширить, разрешив отправителю передавать такие деревья решений, которые позволяют корректно классифицировать не все обучающие примеры. Тогда помимо самого дерева решений отправитель должен передавать и информацию об ошибках классификации – только тогда получатель сможет правильно восстановить недостающую у него информацию о номерах классов.

Мы можем воспользоваться схемой кодирования исключений, уже описанной для наборов правил. Для случая двух классов имеем

$$DL_{exceptions} = \log_2 C_M^{M_{FP}} + \log_2 M_{FP}. \quad (106)$$

Напомним, что M – общее число примеров обучающей выборки, M_{FP} – количество неверно классифицированных примеров. Слагаемое, связанное с кодированием пропусков, для деревьев решений обычно отсутствует, поскольку классификационный алгоритм на основе деревьев решений всегда дает на выходе какой-либо класс. Как мы увидим ниже, возможность пропусков несложно реализовать и для деревьев решений, просто введя дополнительный класс. Это бывает полезно, так как обучающая выборка может не покрывать всех значений атрибутов.

Если для набора правил исключения могли кодироваться отдельно для каждого правила, то для деревьев решений может применяться кодирование исключений отдельно для каждого листа. Для этого все примеры обучающей выборки разделяются на подвыборки в зависимости от того, какая из ветвей дерева решений используется при классификации конкретного примера (или в каком из листьев заканчивается анализ). В нашем примере (рис. 38) было бы шесть таких подвыборок. Далее кодирование ошибок осуществляется раздельно для каждой подвыборки, так что оценка длины описания ошибок будет включать сумму из выражений вида (106). Это может быть полезным, если размеры обучающей выборки достаточно большие по сравнению с числом листьев в дереве решений.

Не представляет сложности расширить эту схему кодирования и на случай многих классов (взяв соответствующее слагаемое из уравнения (102)):

$$DL_{exceptions} = \sum_i \left(L(t_i) + m_i H(e_i) \right), \qquad (107)$$

где суммирование теперь осуществляется не по правилам из набора, а по листьям дерева решений. При таком способе кодирования в листьях дерева решений содержится не отдельный «истинный» класс, а перечень всех классов с указанием той вероятности, с которой соответствующий класс должен быть выбран, если классификационный алгоритм останавливается в данном листе.

Как и для всех задач машинного обучения, в задаче построения дерева решений есть два крайних случая.

1. Чрезмерно упрощенное дерево решений («беспорядочная» гипотеза) имеет только корень, который допускает все классы с вероятностями $P(c = a_i)$. Здесь $P(c = a_i)$ – вероятность того, что некоторый пример обучающей выборки относится к классу a_i. Таким деревом любому примеру приписывается класс с той вероятностью, с которой этот класс встречается в обучающей выборке. Тогда суммарная длина описания примерно будет $m_0 H(e_0) = -M \sum_{i=1}^{d} P(c = a_i) \log_2 P(c = a_i)$. Классификация не зависит от значений атрибутов.
2. Чрезмерно усложненное дерево решений (*ad hoc* гипотеза): имеет количество листьев, равное числу примеров обучающей выборки $n_0 = M$. Все примеры таким деревом классифицируются правильно (если только нет примеров с одинаковыми значениями атрибутов, но разными значениями классов), что, однако, не помогает классифицировать новые примеры. Для такого дерева порядок, в котором проверяются атрибуты, не имеет значения, и его длина описания может быть оценена одним слагаемым из выражения (104):

$n_0 H(\text{A})$, где $n_0 = M$, а $H(\text{A}) = -\sum_{i=1}^{d} P(c = a_i) \log_2 P(c = a_i)$. То есть это дерево дает столь же высокую длину описания, что и максимально простое дерево решений.

Оптимальное решение, как и в других задачах, будет находиться между этими двумя крайностями.

Вопросы и упражнения

1. В чем различаются методы восстановления наборов правил и деревьев решений?
2. Какие два типа ошибок могут возникать в восстановленных деревьях решений?
3. Всегда ли при восстановлении деревьев решений лучшим будет то дерево, которое правильно воспроизводит все правила из обучающей выборки?
4. На какой характеристике обучения в первую очередь сказывает размер восстановленного дерева решений?
5. Каким алгоритмом, с точки зрения эвристического программирования, является алгоритм *ID*3?
6. Что не учитывается в алгоритме *ID*3 с точки зрения принципа минимальной длины описания?

Приложения

Далее приводятся материалы для самостоятельной работы студентов (СРС). Тему СРС.1 «Концепция метасистемных переходов В.Ф. Турчина» рекомендуется изучать после лекции №6 «Искусственная жизнь и аниматы». Тему СРС.2 «Основы языка Пролог» рекомедуется изучать после лекции №7 «Логические системы представления знаний». Тему СРС.3 «Общие сведенья об экспертных системах» рекомендуется изучать после лекции № «Фреймы и объектно-ориентированный подход в представлении знаний». Тему СРС.4 «Задачи регрессии и сегментации и их решение в рамках машинного обучения» рекомендуется изучать после лекции №14 «Выбор признаков». Тему СРС.5 «Приложение методов восстановления формальных грамматик к задачам анализа естественного языка» рекомендуется изучать после лекции №15 «Восстановление формальных грамматик». Тему СРС.6 «Алгоритм *ID*3 автоматического построения деревьев решений» рекомендуется изучать после лекции №16 «Автоматическое построение наборов правил и деревьев решений».

СРС.1. Концепция метасистемных переходов В. Ф. Турчина

Одной из центральных проблем в направлении «Искусственная жизнь» является проблема эмерджентности. Понятие эмерджентности тесно связано с понятием системы. Под системой понимается такая совокупность взаимодействующих элементов, которая обладает свойствами, не сводимыми к свойствам ее отдельных элементов, не связанных системообразующими связями. Появление новых свойств при объединении отдельных элементов в систему и называется эмерджентностью.

Само по себе наличие у систем новых свойств не является удивительным и для каждой конкретной системы может быть объяснено: системы, состоящие из одинаковых элементов, отличаются по своим свойствам за счет разной структуры (пространственного или логического взаимного расположения элементов). Гораздо более удивительным и непонятным является сам процесс образования новых систем в природе, где отдельные элементы самопроизвольно объединяются в ранее не существовавшие системы.

Почему это является такой большой проблемой, понятно на примере клеточных автоматов. Клеточные автоматы состоят из отдельных простых элементов, *допускающих* существование очень сложных систем – конфигураций активных клеток, ведущих себя, как единое целое. В частности, существуют конфигурации, перемещающиеся в пространстве, но сохраняющие свою форму, осциллирующие конфигурации и даже самовоспроизводящиеся конфигурации. Все эти конфигурации можно спроектировать на основе чрезвычайно простых элементов и правил взаимодействия, в которые, казалось бы, не закладывалось существование

новых систем априори. Проблема, однако, заключается в том, что из случайной конфигурации в клеточных автоматах не происходит образование сложных конфигураций, то есть не происходит самопроизвольного системообразования, как это имеет место в природе.

Действительно, в природе из элементарных частиц образовались атомы, из атомов – молекулы, из молекул – более сложные органические молекулы и т.д. А ведь согласно современным космологическим представлениям Вселенная исходно находилась в состоянии, в котором наибольшими системами были элементарные частицы, равномерно случайно распределенные в пространстве. Но как видно на примере клеточных автоматов, потенциальная возможность существования сложных систем, вовсе не гарантирует их образования из исходных случайных конфигураций. В природе же не просто возникли новые системы, но эти системы объединились в более сложные системы, которые также обладали свойствами, позволяющими им объединиться в еще более сложные системы и т.д. При этом процесс системогенеза продолжается до сих пор.

Именно в этом и заключается проблема эмерджентности, изучаемая в искусственной жизни: не удается создать искусственным мир с такими «физическими» законами, на основе которых происходило бы неограниченное усложнение систем (как правило, усложнение происходит на один–два уровня организации, явно или неявно заложенных априорно).

Здесь мы, однако, не будем разбирать саму проблему эмерджентности и причины сравнительной неудачности ее моделирования, а примем как данность то, что системообразование происходит в природе, и с этих позиций посмотрим на вопрос возникновения и строения интеллекта. Для этого мы обратимся к классической работе В.Ф. Турчина «Феномен науки. Кибернетический подход к эволюции», которая была написана в 1970 году, но вопросы, рассмотренные в ней, все еще являются актуальными.

Поскольку в этой книге эволюция рассматривается с кибернетических позиций, сформулируем основные понятия кибернетики. Термин «кибернетика» ввел в середине прошлого века Норберт Винер, книга которого «Кибернетика или управление и связь в животном и машине», по сути, положила начало формированию новой научной дисциплины. Сам термин имеет происхождение из древнегреческого языка, где он обозначал искусство управления кораблем. Сейчас же под кибернетикой понимают науку об управлении в системах (как естественных, так и искусственных).

Система в кибернетике рассматривается как некий материальный или информационный объект, состоящий из других объектов – его *подсистем*. Система, как и ее подсистемы, может находиться в различных состояниях и переходить с течением времени из одних состояний в другие. Основной изучаемый в кибернетике вопрос – это влияние состояний одних подсистем на изменение состояний других подсистем в некоторых конкретных системах. Такое влияние называется *управлением*. В.Ф.

Турчин рассматривает эволюцию интеллекта как эволюцию систем управления, в наиболее общем виде представленную на рис. 39.

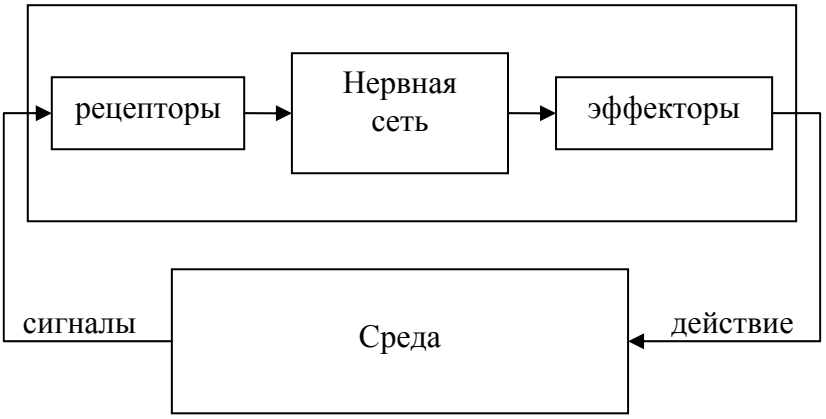


Рис. 39. Нервная сеть как система управления

Видно большое сходство данной схемы с общей схемой функционирования анимата в рамках современного направления «Адаптивное поведение». Таким образом, исследование интеллекта как системы управления вполне адекватно. Вопрос же заключается в том, какой структурой обладает эта система, и как она эволюционировала.

Как и другие сложные системы в природе, интеллект возник в процессе системообразования, то есть иерархического объединения более простых систем в более сложные системы. Для обозначения процесса образования нового уровня в системе управления Турчин ввел понятие *метасистемного перехода*. Прежде, чем переходить к более детальному описанию структуры метасистемного перехода, рассмотрим некоторые его примеры.

Управление положением = движение.

Управление движением = раздражимость (простой рефлекс).

Управление раздражимостью = (сложный) рефлекс.

Управление рефлексами = ассоциации (условный рефлекс).

Управление ассоциациями = (человеческое) мышление.

Управление человеческим мышлением = культура.

В качестве базовой управляемой подсистемы здесь выступает положение тела и его частей. Способность к изменению положения тела означает способность к движению. Однако для этого необходима система эффекторов (двигательный аппарат).

Сама способность к перемещению приносит не слишком большую пользу, если нет возможности управлять движением на основе информации, поступающей от среды. Простейший способ такого управления заключает в непосредственной передаче информации от рецепторов к эффекторам. В качестве примера может быть рассмотрена гидра – типичный представитель кишечнополостных. Гидра обладает раздражимостью: если прикоснуться к ее поверхности иголкой, она сжимается: информация от расположенных у поверхности рецепторов

передается к эффекторам, заставляющим мышцы действовать. Таким образом, раздражимость – это (непосредственное) управление движением на основе текущей информации от среды. Если от одного рецептора идет сигнал к сразу нескольким эффекторам, то возможно и более сложное поведение, например, схватывание добычи сгибаемыми щупальцами.

Однако когда появляется большее число рецепторов и эффекторов, сигналы от одних рецепторов могут вступать в конфликт с сигналами от других рецепторов и оказывается невозможным организовать сложное согласованное движение многих эффекторов. В связи с этим возникает необходимость управления элементарными системами раздражимостей. Такую систему управления Турчин назвал сложным рефлексом в противовес простому рефлексу, при котором сигнал непосредственно идет от одного рецептора к эффекторам. Как простой, так и сложный рефлексы относятся к безусловным рефлексам.

На рис. 40 представлены общие схемы для простого и сложного рефлексов. Если при простом рефлексе сигнал от рецептора непосредственно передается эффектору, то при сложном рефлексе этот сигнал преобразуется системой управления и передается эффекторам в виде последовательности команд, которые могут представлять собой достаточно сложную программу действий (она, однако, запускается сразу при получении сенсорных сигналов и не меняется в течение жизни организма).

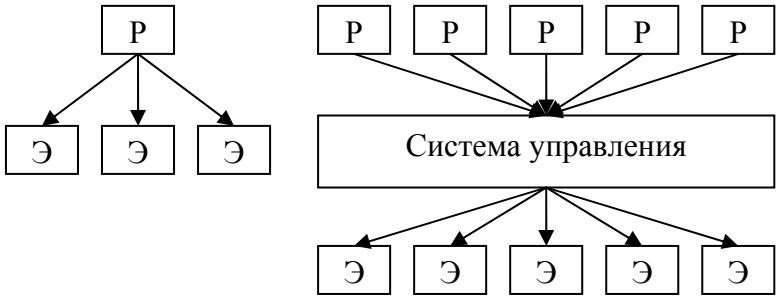


Рис. 40. Общие схемы простого и сложного рефлекса: Э – эффекторы, Р – рецепторы

Общая представленная схема может быть уточнена на случай нескольких различных типов рецепторов, тогда структура системы управления будет конкретизирована (будет состоять из нескольких подсистем, получающих информацию от определенных групп рецепторов и передавать ее дальше на подсистемы, интегрирующие информацию от более ранних подсистем).

Хотя система управления (нервная сеть), реализующая условный рефлекс, может быть весьма сложной, она всегда обладает одним нежелательным свойством: неизменностью во времени. Состояния этой системы могут меняться, но ее структура остается постоянной. Иными словами, животное с такой системой управления не сможет учиться на

опыте; его поведение в повторяющихся ситуациях будет одинаковым, даже если это поведение не оптимально.

Чтобы поведение животного могло меняться, система управления, реализующая сложный рефлекс, должна иметь способность к изменению своей структуры, причем процесс этого изменения должен управляться со стороны другой подсистемы. Многие животные обладают способностью к образованию условных рефлексов, то есть к образованию новых сложных рефлексов, которые в исходной архитектуре нервной сети отсутствуют. Турчин подобную способность к управлению сложными рефлексами (проявляющуюся в виде условных рефлексов) назвал *ассоциированием*. Условный рефлекс – это сложный рефлекс, возникший в результате ассоциирования, то есть установления переменных связей между рецепторами, эффекторами и имеющимися подсистемами управления раздражимостью.

Следует иметь в виду, что ассоциирование – это очень широкое понятие, и условный рефлекс представляет собой лишь простейший пример ассоциирования. После каждого метасистемного перехода эволюция вовсе не останавливается до следующего перехода, а продолжает работать в рамках данного уровня организации. Таким образом, раз возникнув, способность к ассоциированию постепенно усложнялась. В результате, ассоциирование – это связывание любых понятий или сенсорных представлений и программам действия, но только в том случае, когда они совместно появляются в опыте животного. В этом и заключается основное ограничение ассоциирования: оно происходит только в пределах текущей ситуации.

Следующий метасистемный переход заключается в возникновении системы управления ассоциированием. Турчин обосновывает, что эта система управления – воображение, то есть способность выйти за рамки текущей ситуации и представить, *что было бы, если…* Почему воображение претендует на роль системы управления ассоциированием, достаточно понятно: ассоциирование работает только на основе возникающих одновременно представлений; воображение же совмещает не возникавшие, но способные возникнуть одновременно представления, «обманывая» подсистему ассоциирования тем, что соответствующее событие как будто действительно имело место. Конечно, при переходе на такой высокий уровень организации систем управления неизбежно теряется четкость, которая имелась на уровне простых рефлексов, однако такое предположение выглядит правдоподобно.

Примитивное воображение, видимо, есть уже у животных, что проявляется в их играх, где реальные ситуации заменяются их моделями, но вполне развитое воображение есть только у человека, что позволяет ему преодолевать ту контекстную зависимость поведения, которая имеется у животных.

Механизмы управления ассоциированием лежат в основе ряда чрезвычайно важных функций, таких как, например, планирование.

Наиболее важная из них – это способность к овладению языком. Способность к ассоциированию элементов опыта с абстрактными знаками дает человеку мощнейшее средство моделирования мира, в связи с чем уровень управления ассоциациями можно считать уровнем мышления. Также этот уровень является основой для следующего метасистемного перехода – социальной интеграции, то есть объединения человеческих индивидов в некую систему нового уровня: человеческое общество.

В качестве системы управления мышлением Турчин рассматривает культуру, одной из основных частей которой является язык, а также большая совокупность накапливаемых человечеством знаний. Действительно, мышление, как мы его привыкли воспринимать, – это социальный продукт. Если рассматривать с точки зрения кибернетики человеческий разум как подсистему, входящую в систему человеческого социума, то состояние этой подсистемы подвержено существенному влиянию со стороны общей системы. Действительно, ни язык, ни большой объем знаний, которыми располагает отдельный человек, не были изобретены им, а преимущественно были заимствованы из культуры. При этом, естественно, носителями культуры являются люди, без которых она умирает.

Как и в случае других метасистемных переходов, системы, возникшие в результате данного перехода, были исходного примитивными, но способными к дальнейшему развитию. Это развитие очень сложное и многоплановое, и его анализу посвящена значительная часть книги В.Ф. Турчина; особое внимание уделяется развитию науки (на примере математики) и метанауки, и рассматриваются внутренние метасистемные переходы в ходе ее развития. Таким образом, концепция метасистемных переходов позволяет описать в единых терминах различные уровни мышления: от простейших рефлексов до логического мышления, – которые возникли на разных этапах эволюции жизни (см. табл. 2).

Таблица 2. Этапы эволюции по В.Ф. Турчину

Химическая эра	Химические формы жизни
	Движение
Кибернетическая эра	Раздражимость (простой рефлекс)
	Нервная сеть (сложный рефлекс)
	Ассоциирование (условный рефлекс)
Эра разума	Мышление
	Культура, социальная интеграция

Помимо указания этапов эволюции большой интерес представляет общая схема метасистемного перехода, разработанная В.Ф. Турчиным,

который заметил во всех перечисленных переходах одну общую черту. На каждом этапе биологическая или социальная система имеет подсистему, которая образовалась позднее всего в эволюции и которая может рассматриваться как текущий верхний уровень управления.

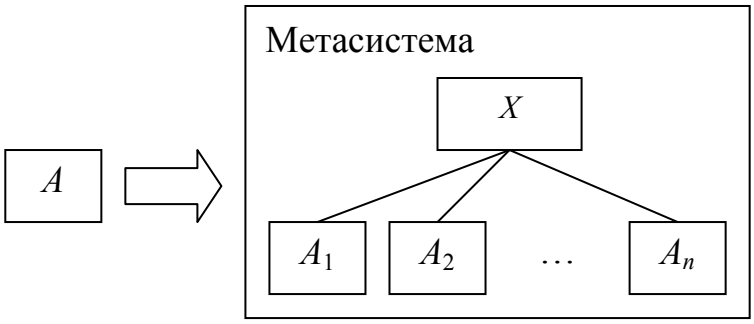


Рис. 41. Общая схема метасистемного перехода; *A* – система предыдущего уровня; *X* – новый верхний уровень в системе управления; A_i – продублированные подсистемы *A* в результате образования метасистемы

Метасистемный переход происходит путем многократного дублирования этой подсистемы верхнего уровня и объединения копий подсистемы путем образования следующего уровня управления. В ходе дальнейшей эволюции продублированные подсистемы могут специализироваться, переставая быть полностью идентичными, а новая подсистема верхнего уровня эволюционирует, постепенно повышая качество управления. Общая схема метасистемного перехода представлена на рис. 41.

Эта схема описывает различные метасистемные переходы, начиная с образования органов зрения (путем дублирования единичных рецепторов и постепенным образованием иерархической системы зрительного восприятия) и заканчивая социумом, который также образуется из исходно «однородных» элементов (которые затем специализируются) и неразвитой системы управления (которая постепенно эволюционирует и усложняется). Конечно, данная схема весьма неконкретна и не описывает эти процессы подробно, однако в общем виде такое описание вряд ли вообще возможно.

Здесь мы рассмотрели концепцию метасистемных переходов лишь в самых общих чертах, опустив многие важные детали, которые присутствуют в крайне интересной книге В.Ф. Турчина. В заключение отметим, что эта концепция легла в основу идеи суперкомпиляции в области языков программирования, на ее основе также строятся модели искусственной жизни и аниматы, а также функционирует проект «Principia Cybernetica» (http://pespmc1.vub.ac.be/), в котором вопросы, связанные с человеком и человечеством, рассматриваются с позиций кибернетической эволюции.

Вопросы и упражнения

1. Изучением каких вопросов занимается кибернетика?

2. В чем заключается смысл понятия эмерджентности в теории систем и проблема эмерджентности при моделировании искусственной жизни?
3. Что такое метасистемный переход и какова его структура, согласно В.Ф. Турчину? Как концепция метасистемных переходов соотносится с проблемой эмерджентности?
4. Какие основные метасистемные переходы происходили в процессе эволюции? Какие основные этапы эволюции можно выделить на этой основе?
5. Можно ли помимо указанных основных метасистемных переходов выделить какие-то дополнительные переходы в более частных системах, например, в подсистемах, из которых состоит человеческих социум?
6. Как вы считаете, завершилась ли цепочка метасистемных переходов в эволюционном процессе, или можно предположить, что будут продолжать образовываться какие-то системы следующих уровней?
7. С какими концепциями в искусственном интеллекте теория метасистемных переходов согласуется, а с какими вступает в противоречие?

СРС.2. Основы языка Пролог

Язык программирования Пролог представляет собой наиболее широко распространенную реализацию принципов логического представления знаний и заслуженно называется одним из языков искусственного интеллекта, хотя отношение к этому языку различное. Помимо положительного мнения о Прологе, существует также и его критика.

Критика Пролога имеет две стороны. Во-первых, он критикуется многими исследователями ИИ за то, что в нем реализуется лишь логическое представление знаний и лишь достоверный вывод, в то время как в большинстве случаев требуется использовать рассуждения в условиях неопределенности. Во-вторых, разработчиками прикладных систем Пролог критикуется за недостаточное удобство и гибкость, из-за которых создание программных продуктов с использованием этого языка часто оказывается более трудоемким и требующим специфической квалификации разработчиков, чем с использованием языков высокого уровня общего назначения.

В то же время, для решения ряда задач Пролог оказывается гораздо более эффективным, чем, например, C++ или Java. В частности, внутренне присущий Прологу декларативный стиль представления знаний обеспечивает более легкую поддержку программ в течение их жизненного цикла. В связи с этим Пролог следует рассматривать в качестве специализированного языка программирования, который может использоваться в экспертных системах или для быстрого создания

прототипов интеллектуальных систем, то есть следует не противопоставлять его языкам общего назначения, а использовать как дополнительный инструмент. Хотя Пролог связан лишь с одним из вопросов ИИ и не покрывает многих других вопросов, он заслуживает изучения.

Язык Пролог был разработан в начале 1970-х гг. На настоящий момент существуют многочисленные диалекты и расширения этого языка. Но поскольку здесь описываются лишь общие сведенья об этом языке, при изложении будет использоваться его базовая версия.

Программа на языке Пролог преимущественно состоит из списка логических утверждений, которые можно разделить на факты и общие правила. Эти утверждения составляются из имен предикатов, переменных и их значений, которые представляют собой символьную строку, начинающуюся с буквы, а также совокупности логических операций. В конце каждого утверждения ставится точка ".". Переменные обязательно начинаются с прописной буквы, поэтому все значения должны начинаться со строчной буквы. Использующиеся в Прологе обозначения логических операций представлены в табл. 3. При этом кванторы существования и всеобщности в Прологе опускаются: хотя они фактически используются, в явном виде в утверждениях (общих правилах, в которых фигурируют переменные) они не указываются. Отметим также, что для предикатов, переменных и их значений отсутствуют объявления, как это имеет место во многих языках общего назначения.

Таблица 3. Логические операции в Прологе

Имя операции	Логическая операция	Обозначение в Прологе
и	$\wedge$	,
или	$\vee$	;
если	$\Leftarrow$	:–
не	$\neg$	not

Рассмотрим некоторые примеры утверждений на языке Пролог. Ниже приведен пример списка фактов, не содержащих переменных.

```
man(serg).
man(alex).
father(serg, alex).
mother(kat, serg).
```

Эти факты говорят, что предикаты "man", "father", "mother" истинны при указанных значениях аргументов. Пролог работает в рамках предположения о замкнутости базы знаний, означающей, что все значения предикатов заданы или могут быть вычислены без получения новой информации (и истинность предикатов не меняется в процессе исполнения программы). Для реализации замкнутости базы знаний в Прологе предикатам, не определенным для каких-либо значений переменных, по умолчанию приписываются ложные значения.

Общие правила в Прологе записываются так же, как и факты, но в них присутствуют переменные. При этом подразумевается, что общее правило верно при всех значениях всех входящих в него переменных. Иными словами, правило

p1(X) :– p2(X,Y).

означает логическое выражение

$$(\forall X,Y)p_2(X,Y) \Rightarrow p_1(X).$$

Однако поскольку в Прологе подобные выражения используются для получения значения p1(X) при некотором X, то это выражение удобнее заменить выражением

$$(\forall X)p_1(X) \Leftarrow (\exists Y)p_2(X,Y),$$

которое означает, что $p_1(X)$ истинно, если существует хотя бы одно значение Y, при котором $p_2(X,Y)$ истинно. Такая форма гораздо удобнее для вывода, поскольку выражение $p_2(X,Y) \Rightarrow p_1(X)$ истинно при любых значениях $p_1(X)$, если при данном Y предикат $p_2(X,Y)$ ложен. То есть на основе истинности всего выражения мы не можем установить истинность $p_1(X)$. Отметим, однако, следующее. Если дано правило

p1(X,Y) :– p2(X).

то оно означает вовсе не выражение

$$(\forall X)p_2(X) \Rightarrow (\exists Y)p_1(X,Y),$$

а выражение

$$(\forall X)p_2(X) \Rightarrow (\forall Y)p_1(X,Y),$$

поскольку операция "$\Leftarrow$" несимметрична.

Рассмотрим некоторые примеры общих фактов, записанных на языке Пролог, и приведем соответствующие формулы в логике предикатов.

father(X, Y) :– man(X), parent(X, Y).

$$(\forall X,Y)man(X) \wedge parent(X,Y) \Rightarrow father(X,Y)$$

Если некто X является мужчиной и родителем Y, то X – отец Y.

parent(X, Y) :– mother(X, Y); father(X, Y).

$$(\forall X,Y)mother(X,Y) \vee father(X,Y) \Rightarrow parent(X,Y)$$

X является родителем Y, если X – отец или мать Y.

grandfather(X, Y) :– father(X, Z), parent(Z, Y).

$$(\forall X,Y)(grandfather(X,Y) \Leftarrow (\exists Z)\, father(X,Z) \wedge parent(Z,Y))$$

X является дедушкой Y только тогда, когда X является отцом (некоторого) родителя Y.

woman(X) :– mother(X, Y).

$$(\forall X)(woman(X) \Leftarrow (\exists Y)mother(X,Y))$$

Если X приходится кому-то матерью, то она является женщиной.

brother(X, Y) :– man(X), parent(Z, X), parent(Z, Y).

$$(\forall X,Y)(brother(X,Y) \Leftarrow (\exists Z)man(X) \wedge parent(Z,X) \wedge parent(Z,Y))$$

Если у X и Y общий родитель и X – мужчина, то X – брат Y.

Приведем еще пример одного "плохого" правила.

mother(X, Y) :– woman(X).

$$(\forall X)(woman(X) \Rightarrow (\forall Y) mother(X, Y)).$$

Если X является женщиной, то она мать любого Y.

Как уже отмечалось, программа на языке Пролог представляет собой список общих правил и фактов и не наделена функциональностью, а декларативно представляет некие знания. За исполнение программы отвечает интерпретатор, который взаимодействует с пользователем (в качестве которого также вполне может выступать и программа на каком-то другом языке), интерактивно отвечая на его запросы.

Запрос к интерпретатору также представляется в виде логического выражения, истинность которого требуется установить. Рассмотрим следующую простую программу.

```
mother(X, Y) :– woman(X), parent(X, Y).
father(X, Y) :– man(X), parent(X, Y).
grandfather(X, Y) :– father(X, Z), parent(Z, Y).
man(serg).
man(alex).
woman(kat).
parent(serg, victor).
parent(kat, victor).
parent(victor, alex).
```

И посмотрим на следующие запросы, начинающиеся с приглашения "?–", на которые интерпретатор возвращает некоторый ответ.

```
?– father(serg, victor).
        Yes
?– grandfather(serg, alex).
        Yes
?– mother(kat, alex).
        No
?– father(victor, alex).
        No
```

Как видно, для обработки этих запросов интерпретатору необходимо выполнить логический вывод, а не просто обратиться к базе правил. Также видно, как действует предположение о замкнутости базы знаний: поскольку в базе нет фактов об истинности parent(kat, alex) и man(victor), на последние два запроса возвращается ответ "Нет".

В Прологе существуют также и запросы в виде выражений с переменными. Посмотрим, как работают эти запросы.

```
?– mother(kat, X).
        X = victor
        Yes
?– grandfather(X, alex).
        X = serg
        Yes
?– grandfather(X, Y).
```

X = serg, Y = alex
Yes
?– grandfather(X, kat).
No
?– man(X).
X = serg
Yes

Здесь можно отметить следующее. В подобного рода запросах предполагается квантор существования для входящих в него переменных. Выражение истинно, если найдется хотя бы одно подходящее значение X (как именно обозначать переменные в запросе, не имеет значения). В действительности, при нахождении в базе первого значения, для которого введенное выражение истинно, интерпретатор спрашивает пользователя, продолжать ли поиск. Поиск продолжается при нажатии команды ";" (или) до тех пор, пока не будет дан ответ "Нет" в связи с исчерпанием всех возможностей, например,

?– man(X).
X = serg ;
X = alex ;
No

Более сложные выражения в запросах также допустимы, например,

?– mother(kat, X), father(Y, X).
X = victor, Y = serg
Yes

Этот запрос позволяет определить, кто является отцом сына kat.

Стоит отметить, что интерпретатор не только отвечает на запросы, но также позволяет интерактивно изменять базу правил и фактов, для чего могут использоваться команда assert для добавления записи (asserta и assertz позволяют добавлять записи в начало и конец списка соответственно) и команда retract для удаления записи. Существует также набор команд интерпретатора для отладки базы и процесса вывода, например, команды listing, trace и spy.

Теперь кратко рассмотрим вопрос о том, как интерпретатор Пролога выполняет запросы. В Прологе используется фундаментальный метод доказательства теорем – метод опровержения резолюций. Суть идеи доказательства путем опровержения тривиальна: к имеющемуся набору фактов и правил добавляется отрицание утверждения, которое нужно доказать, и производится попытка прийти к противоречию (доказательство от противного). Если в процессе доказательства устанавливается, что обратное утверждение не совместимо с имеющимися аксиомами (фактами и правилами), то исходное утверждение должно быть истинно. В методе опровержения резолюций используется единственное правило вывода – принцип резолюций, который имеет следующую форму:

$$\frac{a \vee c, \quad b \vee \neg c}{a \vee b}.$$

Этот логический вывод означает: если истинны выражения $a \vee c$ и $b \vee \neg c$, то должно быть истинным и выражение $a \vee b$, которое называется резольвентой. Здесь как a, так и b могут быть сложными выражениями. Как частный случай, a или b могут быть пустыми. Если оба высказывания пусты, то в качестве резольвенты получается пустое высказывание, что также означает противоречие (высказывания c и $\neg c$ несовместимы).

Для применения принципа резолюций удобно все выражения представлять в дизъюнктивной нормальной форме, то есть форме, в которой используются только логические операции "$\vee$" и "$\neg$". Например, следующее выражение, характерное для Пролога,

$a \Leftarrow b \wedge c$ (в Прологе: "a :– b, c.")

в дизъюнктивной форме будет:

$a \vee \neg b \vee \neg c$.

Перепишем в дизъюнктивной форме нашу базу знаний о родственных отношениях.

mother(X, Y) $\vee$ $\neg$woman(X) $\vee$ $\neg$parent(X, Y)

father(X, Y) $\vee$ $\neg$man(X) $\vee$ $\neg$parent(X, Y).

grandfather(X, Y) $\vee$ $\neg$father(X, Z) $\vee$ $\neg$parent(Z, Y).

Факты останутся без изменения.

man(serg).

man(alex).

woman(kat).

parent(serg, victor).

parent(kat, victor).

parent(victor, alex).

И докажем методом опровержения резолюций утверждение:

grandfather(serg, alex).

Для этого рассмотрим отрицание этого утверждения, для которого резолюцию можно составить с третьим общим правилом из базы.

$\neg$grandfather(serg, alex), grandfather(X, Y)$\vee \neg$father(X, Z)$\vee \neg$parent(Z, Y)

резольвента: $\neg$father(serg, Z)$\vee \neg$parent(Z, alex).

В исчислении предикатов принцип резолюции подобен принципу резолюций в исчислении высказываний, но несколько более сложный из-за использования кванторов и переменных. В данном случае общее правило верно для любых значений X, Y, Z, а значит оно должно быть верно и для X=serg, Y=alex. Резольвента же должна быть верна для любых значений Z. Далее, используя факт parent(victor, alex), получим резолюцию:

$\neg$father(serg, Z)$\vee \neg$parent(Z, alex), parent(victor, alex)

резольвента: $\neg$father(serg, victor).

Далее

$\neg$father(serg, victor), father(X, Y)$\vee \neg$man(X)$\vee \neg$parent(X, Y)

резольвента: $\neg$man(serg)$\vee \neg$parent(serg, victor).

Затем

$\neg$man(serg)$\vee \neg$parent(serg, victor), man(serg)

резольвента: $\neg$parent(serg, victor).

И, наконец,

$$\neg \text{parent(serg, victor)}, \qquad \text{parent(serg, victor)}$$

резольвента: пусто (ложь).

Таким образом, пришли к противоречию, а значит заданное выражение grandfather(serg, alex) истинно.

Поскольку предлагаемое интерпретатору утверждение может оказаться как ложным, так и истинным, а при расширении имеющихся фактов отрицанием ложного высказывания противоречие получено быть не может и порождение резольвент может оказаться длительным, то могут проводиться параллельно два доказательства: для отрицания данного утверждения и для него самого.

Из нашего примера видно, что на некоторых шагах вывода может быть составлено несколько различных резолюций. При этом резольвенты могут быть как короче, так и длиннее исходного утверждения. Для нашего простого примера с тремя правилами вывод был достаточно короткий и почти однозначный (в плане выбора следующей резолюции). Однако при увеличении базы правил число возможных вариантов продолжения вывода может сильно возрастать, и далеко не все резольвенты будут приближать нас к построению опровержения. В связи с этим, в методе резолюций требуется использовать поиск (а не просто применять первую возможную резолюцию). При этом сложность поиска в общем случае возрастает экспоненциально с ростом размера базы, поэтому оказывается необходимым также привлекать различные эвристики поиска. Стратегия поиска сильно зависит от реализации в конкретном интерпретаторе; зачастую это поиск в глубину с возвратом.

К сожалению, такой поиск в общем случае оказывается неработоспособным. К примеру, дополним нашу базу правилом:

woman(X) :– mother(X, Y).

или в дизъюнктивной форме: $\text{woman(X)} \vee \neg \text{mother(X, Y)}$. Это правило значит, что если X является чьей-то матерью, то X – женщина (что выглядит вполне разумным).

Однако теперь, если мы попытаемся доказать, например, утверждение woman(victor), то могут быть последовательно построены две резолюции:

$$\neg \text{woman(victor)}, \qquad \text{woman(X)} \vee \neg \text{mother(X, Y)}$$

резольвента: $\neg \text{mother(victor, Y)}$

и

$$\neg \text{mother(victor, Y)}, \qquad \text{mother(X, Y)} \vee \neg \text{woman(X)} \vee \neg \text{parent(X, Y)}$$

резольвента: $\neg \text{woman(victor)} \vee \neg \text{parent(victor, Y)}$.

Последняя резольвента опять требует проверки $\neg$woman(victor). Таким образом, два вполне естественных правила приводят к бесконечному циклу. При этом данная резольвента не совпадает с исходным утверждением $\neg$woman(victor), что не позволяет прервать вывод. Хорошо известно, что получение ответа на запрос в Прологе является в общем случае алгоритмически неразрешимой задачей. При этом проблема

останова является замаскированной от разработчиков программ на Прологе, поскольку механизм вывода скрыт внутри интерпретатора.

Особенно часто данная проблема проявляется при одновременном использовании как некоторого понятия, так и его отрицания. Эта проблема усугубляется тем, что значения по умолчанию (ложь) для разных предикатов могут входить в противоречие. Например, предикат woman – это отрицание предиката man. Пусть в базе есть два вполне естественных правила:

man(X) :– not(woman(X)).

woman(X) :– not(man(X)).

Но для victor у нас не было установлено ни значение man(victor), ни значение woman(victor). Поскольку оба эти значения полагаются ложными, вся база будет содержать противоречие, а работа интерпретатора будет непредсказуемой. Это противоречие плохо тем, что оно завуалировано и его не так просто может оказаться выявить. Сравните, например, с явным противоречием, записанным в декларативной форме:

something(X) :– not(something(X)).

Если база содержит такое общее правило, то запрос к интерпретатору something(victor) приведет к зацикливанию (как правило, интерпретатор прекращает работу, если вывод занимает больше некоторого времени или превышает некоторую глубину). Но подобные противоречия в самой базе правил, по крайней мере, видны разработчику.

Итак, проблема отрицаний, усугубляемая предположением замкнутости базы знаний, требует от программиста на языке Пролог высокой компетенции, чтобы в создаваемой системе правил учитывались неявные предположения, делаемые интерпретатором, для устранения возможности алгоритмически неразрешимых запросов. Для этого также могут оказаться полезными упоминавшиеся интерактивные средства отладки, предоставляемые интерпретатором Пролога.

В то же время, при корректном проектировании систем правил и фактов Пролог требует от программиста заметно меньших усилий. К примеру, реализация базы родственных отношений с возможностью вывода по ней на языке C++ потребовала бы существенно большего труда, чем простое описание соответствующих предикатов и их отношений.

Мы рассмотрели лишь самые базовые возможности языка Пролог. Опишем кратко некоторые дополнительные возможности.

Помимо логических выражений Пролог допускает арифметические выражения, включающие операции сложения "+", вычитания "–", умножения "*", деления "/" и остатка от целочисленного деления "mod". Для того чтобы арифметическое выражение вычислялось, необходимо использовать оператор is. Например, запрос интерпретатору может выглядеть следующим образом

?– X is 8+3*12.

X = 44

Yes

То есть интерпретатор производит «поиск» такого значения переменной X, при котором введенное выражение истинно.

Арифметические выражения могут использоваться также и для описания связи переменных, входящих в предикаты. Пусть в базе записано следующее правило:

```
f(X, Y, Z) :– Z is X*Y.
```

Тогда возможны следующие запросы:

```
?– f(3, 2, 6).
        Yes
?– f(5, 7, Z).
        Z = 35
        Yes
```

Однако запрос вида f(5, Y, 35) является некорректным, поскольку интерпретатор требует, чтобы переменные, входящие в арифметическое выражение после оператора is были определены на момент вычисления. Таким образом, поиск в действительности не осуществляется, а происходит простое вычисление.

Поместим теперь в базу следующие правила:

```
fr(1, M) :– M is 1.
fr(N, M) :– N > 1, N1 is N - 1, fr(N1, M1), M is N * M1.
```

Здесь мы заставляем интерпретатор вычислять значение M на основе такого значение M1, для которого будет истинным предикат fr(N-1, M1). Несложно убедиться, что этот предикат позволяет вычислять значение факториала:

```
?– fr(5, X).
        X = 120
        Yes
?– fr(8, X).
        X = 40320
        Yes
?– fr(3, 8).
        No
```

Приведем еще в качестве примера список правил, позволяющих вычислять числа Фибоначчи.

```
fibonacci(0, M) :– M is 1.
fibonacci(1, M) :– M is 1.
fibonacci(N, M) :– N > 1,
        N1 is N-1, fibonacci(N1, M1),
        N2 is N-2, fibonacci(N2, M2),
        M is M1 + M2.
```

Из этого примера также видно, что в качестве условия можно использовать сравнения ">" или "<" (а также прочие операции сравнения). Это может применяться, например, следующим образом. Расширим нашу базу родственных отношений предикатом age, для которого введем следующие факты:

age(serg, 50).
age(kat, 48).
age(victor, 25).
age(alex, 3).

Тогда возможен следующий запрос.

?– age(X, Y), Y>26.
X = serg ;
X = kat
Yes

Помимо арифметических операций в Прологе также могут использоваться списки – упорядоченные множества элементов, которые являются чрезвычайно важными. Списки в Прологе имеют определенное сходство со списками в Лиспе – другом языке искусственного интеллекта, основой которого они являются. Список в Прологе задается перечислением своих элементов через запятую внутри квадратных скобок:

[] – пустой список;

[serg, kat, victor, alex] – четырехэлементный список;

[[serg, kat], [victor, alex]] – список из двух элементов, которые сами по себе являются списками.

Поскольку в Прологе все основано на предикатах, для объявления некоторого списка нужно ввести соответствующий предикат. Список как элемент базы фактов может выглядеть, например, следующим образом:

family([serg, kat, victor, alex]).
family([nick, ann, igor]).

Основной операцией по работе со списками является их расщепление на голову (первый элемент списка) и хвост (оставшиеся элементы списка). Эта операция осуществляется заданием шаблонов вида [X|Y], где X – голова списка, а Y – его хвост. Поясним это на примере следующих запросов.

?– family([serg|X]).
X = [kat, victor, alex]
Yes
?– family([nick, ann|X]).
X = [igor]
Yes
?– family([X, Y|[victor, alex]]).
X = serg, Y = kat
Yes

Здесь следует обратить внимание на то, что хвост списка также является списком, а голова списка – это термы.

Итак, шаблоны списков имеют следующий смысл:

[X|Y] – список, состоящий не менее чем из одного элемента

[X, Y] – список, состоящий ровно из двух элементов

[X, Y|Z] – список, состоящий не менее чем из двух элементов

[a|X] – список, первым элементом которого является «a».

[X|[a]] – список, состоящий из двух элементом, второй элемент – «а». На основе шаблонов можно создавать общие правила для списков.

Приведем простой пример предиката, проверяющего принадлежность элемента списку.

member(X, [X|L]).

member(X, [Y|L]) :– member(X, L).

Этот предикат работает следующим образом: если X – голова списка, имеющего хвост L, то X является членом списка XL; если X является членом хвоста списка L, то он является также и членом полного списка YL, где Y – произвольный хвост.

Пусть, к примеру, мы делаем запрос:

?– member(serg, [serg, kat, victor, alex]).

Тогда интерпретатор, обращаясь к правилу member(X, [X|L]) делает подстановку: X=serg, [X|L]=[serg|kat, victor, alex], подстановка оказывается успешной, и он возвращается истинное значение. Сделаем запрос

?– member(serg, [kat, serg, victor, alex]).

Интерпретатор не сможет выполнить унификацию для первого правила и обратится ко второму: member(X, [Y|L]) :– member(X, L). Теперь X=serg, Y=kat, L=[serg, victor, alex]. После этого он будет проверять истинность для member(serg, [serg, victor, alex]).

Теперь мы можем воспользоваться введенными ранее фактами для предиката family и общими правилами для предиката member и организовать такой запрос:

?– member(kat, X), family(X).

X=[serg, kat, victor, alex]

Yes

То есть мы сможем получать список по входящему в него элементу.

Существует ряд встроенных предикатов для работы со списками. Например, предикат length(L, N) является истинным, если N – длина (количество элементов) списка L. Этот предикат можно использовать как при организации других предикатов, так и в запросах:

?– length(L, X), family(L).

L = [serg, kat, victor, alex], X = 4 ;

L = [nick, ann, igor], X = 3

Yes

?– length(X, 4), family(X).

X = [serg, kat, victor, alex]

Yes

Другие имеющиеся в распоряжении программиста предикаты:

member(X, L) в действительности является встроенным предикатом;

append(L1, L2, L3) истинен, когда L3 является объединением двух списков L1 и L2;

last(X, L) истинен, когда X – последний элемент в списке L;

reverse(L1, L2) истинен, когда L2 – это обращенный список L1.

Естественно, все эти предикаты могут служить не только для проверки истинности конкретных значений, но также и для выполнения операций по обращению списков, их объединению или определению последних элементов. Все они, хотя и являются встроенными, легко реализуются «вручную», как и рассмотренный предикат member (встроенные предикаты, однако, эффективнее в использовании).

Мы рассмотрели лишь самые базовые возможности языка Пролог, которые, в действительности, существенно шире. Например, на основе списков и механизмов вывода в Прологе строятся прочие абстрактные типы данных: стеки, очереди, множества. Помимо различных структур данных Пролог также поддерживает некоторые механизмы управления поиском, которых мы не касались. Современные же версии Пролога имеют также обширные библиотеки, например, для выполнения и обработки http-запросов, что существенно расширяет сферу применения данного языка.

Вопросы и упражнения

1. В чем преимущества и недостатки языка Пролог по сравнению с языками общего назначения? Корректно ли их сравнивать?
2. Каким образом интерпретатор языка Пролог отвечает на запросы пользователя?
3. Из каких основных элементов состоит программа на языке Пролог?
4. Пусть программа на языке Пролог имеет следующий вид:
   ```
   f(1, M) :– M is 1.
   f(N, M) :– N > 1, N1 is N - 1, f(N1, M1), M is N * M1.
   ```
   Какие запросы к этой программе можно сформировать, и каков будет результат этих запросов? Какого вида запросы будут некорректными?
5. Напишите программу на языке Пролог, которая бы позволяла находить наибольший общий делитель двух чисел.
6. Какая проблема возникает при использовании двух предикатов, которые являются отрицанием друг друга? Как можно сформулировать проблему отрицания в Прологе?
7. Что из себя в Прологе представляют списки? Какие возможности они предоставляют и как могут быть использованы?

СРС.3. Общие сведенья об экспертных системах

Одним из наиболее широких практических применений ранних результатов в области искусственного интеллекта стали экспертные системы (ЭС). В этих системах был произведен синтез двух парадигм ИИ: поиск и представление знаний. Прикладная направленность ЭС подразумевала, что они должны быть полезны человеку при решении определенных задач. В результате, вместо создания одной универсальной

системы решения задач (такой как Общий Решатель Задач) практичнее оказалось создавать большое количество систем, каждая из которых справлялась бы с задачами из некоторой конкретной предметной области. Таким образом, был установлен компромисс между универсальностью и способностью к решению единственной задачи (что было характерно для классического программирования). В результате подобные прикладные интеллектуальные системы по сфере своей компетенции стали напоминать людей-экспертов, которые обладают обширными знаниями и могут решать любые задачи (или консультировать менее опытных специалистов), но лишь в своей предметной области.

Итак, экспертная система должна обладать знаниями из некоторой предметной области и быть способной использовать эти знания для решения задач или выработки рекомендаций. При этом ЭС может выступать как в роли самостоятельного специалиста, полностью заменяя человека в принятии решений, так и входить в человеко-машинную систему, находясь в непрерывном интерактивном взаимодействии с пользователем. Отсюда следует, что функциональность экспертной системы должна включать следующие элементы.

Представление знаний. Экспертные системы – это системы, основанные на знаниях. Информация о предметной области представляется в них в виде знаний – совокупности взаимосвязанных элементов, описанных на формальном языке, обладающем четкими синтаксисом и семантикой. Четкость синтаксиса означает однозначную интерпретируемость автоматическими средствами манипуляции знаниями, а четкая семантика – возможность отображения элементов знания на объекты и события реального мира (или, говоря более конкретно, предметной области). Экспертные системы часто используют одновременно несколько представлений знаний, среди которых могут быть семантические сети, фреймы, наборы правил, логические представления, объектно-ориентированные представления, сценарии и т.д.

К представлениям знаний, как правило, предъявляется два противоречивых требования: большая выразительная сила и наличие операций по манипулированию знаниями. Большая выразительная сила означает способность представлять сложные системы понятий и адекватно описывать отношения между этими понятиями. К примеру, в логике предикатов первого порядка могут быть представлены в форме предикатов родственные отношения: отец, сын, брат, сестра, внук и т.д., но сложно описать отношение «родственник», подразумевающее наличие любого из родственных отношений (такое отношение может быть легко описано как предикат второго порядка). С другой стороны, обычная логика предикатов обладает четкими и мощными механизмами вывода, позволяющими решать задачи на основе этого представления. В более общих представлениях, обладающих большей выразительной силой, механизмы вывода оказываются либо гораздо сложнее, либо гораздо слабее.

Помимо указанных двух требований к представлениям знаний в рамках ЭС часто также предъявляется еще одно дополнительное требование: удобство для человека, работающего с ЭС, то есть естественность (с точки зрения человека) нотации, использующейся для представления знаний. В частности, обычно требуется представление знаний в декларативной форме.

Вывод на основе знаний. Знания, сколь бы обширными они ни были, являются бесполезными, если на их основе не осуществляется решение задач. Решение конкретной задачи, как правило, может быть представлено как поиск, который организуется на основе имеющихся знаний о предметной области. В зависимости от представления знаний поиск может принимать различный вид: при использовании логических представлений это поиск доказательства методом резолюций, при использовании набора правил это поиск такой цепочки продукций, которая приводит из начальной ситуации к конечной цели, и т.д.

Практически для всех нетривиальных задач решение проблемы поиска требует обхода дерева вариантов, число узлов в котором растет экспоненциально с увеличением глубины поиска. В связи с этим оказывается необходимым привлекать эвристики, позволяющие эффективно ограничивать поиск. Подобные эвристики могут быть двух уровней: эвристики, характерные для всей области в целом, и дополнительные эвристики, требуемые для решения некоторой задачи. Эвристики первого типа могут быть заложены в ЭС на этапе проектирования и могут составлять часть базы знаний, отвечая на вопрос «как?», а не на вопрос «что?». Особое место здесь также занимают метазнания, то есть знания системы о собственных знаниях. Если система имеет метаинформацию о своих знаниях, она может предполагать, какие именно знания ей могут потребоваться для решения той или иной задачи, и каких знаний у нее не хватает. Второй тип эвристик подразумевает наличие в ЭС некоторых механизмов, позволяющих пользователю управлять процессом поиска решений. В простейшем случае это может означать, например, что у пользователя есть возможность указать для ЭС приоритет в использовании имеющихся у нее эвристик, которые могут противоречить друг другу и из которых следует выбрать наиболее полезные для решения текущей задачи.

Разъяснение решения. Экспертная система, как правило, должна не только решить какую-то задачу, но и предоставить человеку описание самого процесса ее решения или разъяснить это решение. Разъяснение решений может быть необходимым по нескольким причинам.

- Если ЭС осуществляет консультационные функции, то пользователю, как правило, необходимо не только конечное решение какой-то проблемы, но и разъяснение, почему это решение должно быть именно таким.
- Предоставление пояснений к решению на этапе функционирования ЭС может быть необходимым для проверки его правильности, что

позволяет убедиться в корректности работы ЭС и повышает доверие к автоматически полученным результатам.

- В процессе проектирования и сопровождения ЭС описание системой процесса решения задач необходимо как мощное средство отладки, позволяющее находить и исправлять ошибки в базе знаний и настраивать эвристики поиска. Это может быть полезным, как экспертам, которые являются источником знаний о предметной области, так и программистам, реализующим базовую функциональность ЭС.
- Интерактивность может быть полезным в демонстрационных целях, например, в процессе передачи ЭС заказчику.
- Разъяснение решений может быть полезным также для организации более эффективного управления поиском в целях повышения общей производительности ЭС.

Перечисленные выше элементы ЭС относятся преимущественно к этапу функционирования. ЭС должны обладать еще одной функцией, необходимой при их проектировании.

Приобретение знаний. Эта функция заключается в передаче экспертной системе потенциально полезного опыта решения проблем в некоторой предметной области от некоторого источника знаний и представлении его в виде, который позволяет использовать эти знания в программе. Автоматические методы приобретения знаний до сих пор являются крайне ограниченными, поэтому этот процесс выполняется т.н. инженером по знаниям, участвующим в проектировании ЭС. Инженер по знаниям осуществляется *извлечение знаний*, проводя собеседования с одним или несколькими экспертами в данной предметной области. Прежде, чем переходить к собеседованию (опросу) с экспертами, инженер по знаниям должен познакомиться с терминологией в данной области и основными ее концепциями, в противном случае опрос не будет эффективным. Часто выделяют следующие дальнейшие шаги в извлечении знаний.

- Идентификация. Устанавливается, какие проблемы должна решать будущая ЭС, на основе каких данных, и руководствуясь какими критериями качества решения.
- Концептуализация. Определяются основные понятия и концепции, которыми должна оперировать ЭС, устанавливаются связи между этими концепциями (например, в форме типизированных отношений в семантических сетях).
- Формализация. На основе выявленных понятий и установленных классов проблем формализуется структура пространства вариантов, в котором должен производиться поиск при решении конкретных задач, и определяется, какими методами этот поиск может осуществляться. Оценивается степень соответствия полученной формализации реальным условиям функционирования ЭС.

- Реализация. Создаются программы, воплощающие алгоритмы поиска; формируются и заполняются базы знаний, включая эвристики и методы управления поиском.
- Тестирование. Осуществляется проверка работоспособности ЭС на репрезентативной выборке задач, отладка и настройка базы знаний и эвристик поиска с выявлением недостающих или противоречивых элементов знания, неэффективных эвристик и т.д.

С учетом перечисленных функций общую схему экспертной системы можно представить в виде, изображенном на рис. 42.

Рассмотрим теперь основные проблемы, которые могут решаться экспертными системами. Условно можно выделить два основных класса проблем – анализ и синтез. Проблемы, обычно выделяемые в рамках каждого из классов, представлены на рис. 43 и 44.

Рис. 42. Общая структура экспертной системы

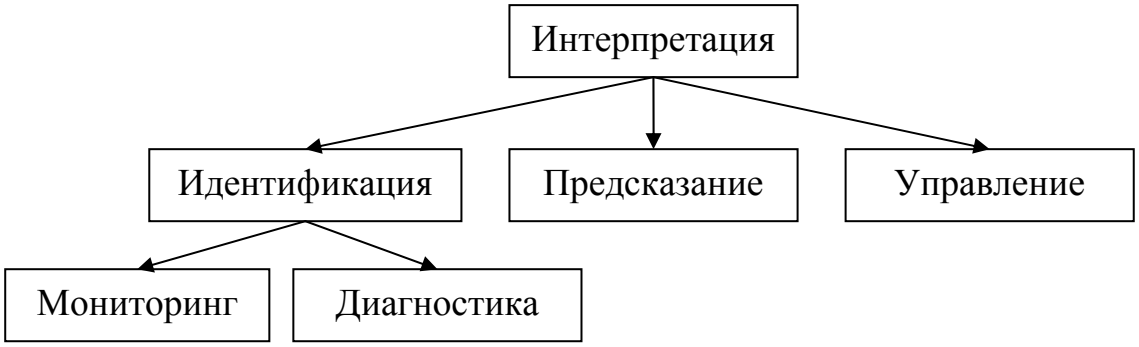


Рис. 43. Условное разделение проблем анализа на подклассы

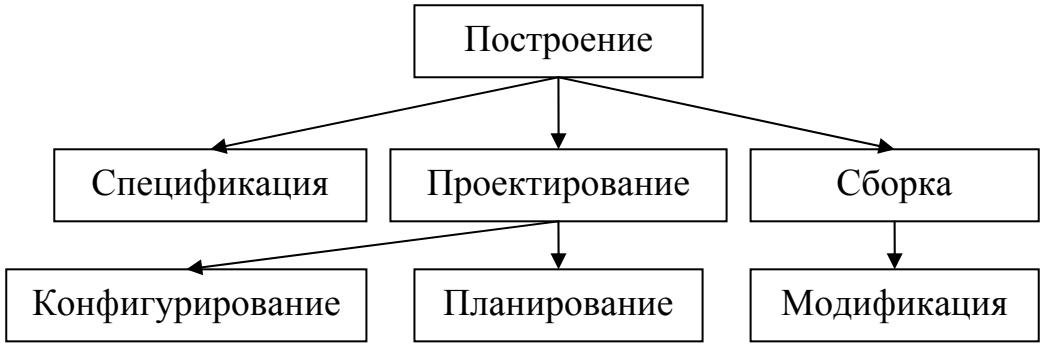


Рис. 44. Условное разделение проблем синтеза на подклассы

В связи с таким разнообразием задач существует также и большое количество ЭС разного назначения.

- *Системы интерпретации* предназначены для анализа данных (в частности результатов наблюдений или сенсорных данных), в результате которого строится описание ситуации. Например, некоторые системы компьютерного зрения строятся с применением технологий экспертных систем. В круг задач, решаемых подобными системами, входят также и такие, как определение структуры химических веществ (например, белков) или клеток.
- *Системы прогнозирования* предназначены для предсказания поведения некоторой системы (политических отношений, финансового рынка, погоды) в будущем. В системах управления осуществляется не только пассивное прогнозирование, но также и анализ возможного влияния того или иного действия на дальнейшее развитие некоторой ситуации с выбором оптимального действия.
- *Системы диагностики* предназначены для выявления неисправностей технических систем по результатам измерений или испытаний, либо для определения болезни по симптомам. Зачастую системы этого типа требуют интерактивного взаимодействия с пользователем, вынося предложения по проверке того или иного показателя диагностируемой системы.
- *Системы мониторинга* предназначены для слежения за некоторым объектом и анализа его характеристик в целях выявления нарушений в его функционировании. Часто мониторинг сопровождается также прогнозированием, поскольку требуется заблаговременно определять возможные неисправности (например, при управлении транспортными потоками).
- *Системы планирования* предназначены для разработки планов действия (последовательности операций), необходимых для достижения какой-либо цели. В отличие от систем управления, запланированные действия выбираются заранее, без знания развития ситуации, поэтому носят обобщенный характер.
- *Системы проектирования* предназначены для выбора оптимальной конфигурации некоторого объекта на основе имеющихся

компонентов при заданных ограничениях. Зачастую при проектировании необходимо устанавливать пространственные отношения между отдельными элементами, в то время как при планировании требуется определять отношения во времени между действиями. Наиболее типичными задачами проектирования являются разработка электронных схем, помощь в выборе архитектурных решений, дизайн помещений.

Прочие задачи синтеза имеют близкий к проектированию и планированию смысл и отличаются преимущественно сложностью синтезируемых объектов и степенью интерактивности.

Помимо этого, в отдельный класс могут быть выделены обучающие системы, которые служат посредником в передаче знаний от экспертов другим людям (например, студентам). При этом такие системы производят анализ знаний студентов в данной области, выявляют пробелы в их знаниях и предоставляют недостающие знания.

Вопросы и упражнения

1. Какие основные функции выполняет экспертная система?
2. В чем основная задача инженера по знаниям?
3. Какие существуют основные типы экспертных систем?
4. Какие особенности у обучающих ЭС?
5. Из каких общих блоков состоит ЭС?
6. Какие выделяются разновидности проблемы анализа?
7. Какие выделяются разновидности проблемы синтеза?
8. Какие основные этапы процесса извлечения знаний?

СРС.4. Задачи регрессии и сегментации и их решение в рамках машинного обучения

Задача регрессии

Одним из наиболее типичных вопросов машинного обучения являются различные задачи распознавания образов. Само распознавание образов (в рамках дискриминантного подхода) может рассматриваться как поиск отображения из непрерывного множества значений в дискретное множество. Иной тип моделей строится в другой фундаментальной задаче статистического анализа и машинного обучения. Это задача регрессии, в которой производится поиск отображения из одного непрерывного множества в другое непрерывное множество. Хотя задача регрессии является не менее важной, чем распознавание или группирование, ее часто не относят к машинному обучению. Это связано с тем, что она долго исследовалась в рамках классических математических подходов. Однако, как будет видно дальше, при решении этой задачи возникают те же проблемы, что и при решении других задач машинного обучения.

Итак, сформулируем задачу регрессии. Пусть $\boldsymbol{Z}$ и $\boldsymbol{Y}$ – случайные векторы, причем $\boldsymbol{Z}$ – это вектор независимых переменных размерности N, а $\boldsymbol{Y}$ – вектор зависимых переменных размерности N_1. И пусть есть обучающая выборка – набор из M пар векторов $(\mathbf{z}_i, \mathbf{y}_i)_{i=1}^{M}$, являющихся отсчетами соответствующих случайных векторов. Необходимо построить модель, с помощью которой можно было бы по новым отсчетам вектора $\boldsymbol{Z}$ предсказывать значение вектора $\boldsymbol{Y}$. Для решения этой задачи вводится матрица факторов (или переменных) регрессии $\mathbf{X} = (\boldsymbol{X}_1, ..., \boldsymbol{X}_n)$ размера $N_1 \times n$, получающихся из независимых переменных в результате применения некоторого преобразования $\mathbf{X} = F(\boldsymbol{Z})$. В простейшем случае преобразование может быть тождественным, и факторы регрессии будут совпадать с независимыми переменными. Поскольку $\boldsymbol{Z}$ – случайный вектор, матрица $\mathbf{X}$ также является случайной, и она будет различной для разных реализаций вектора $\boldsymbol{Z}$.

Чаще всего рассматривается линейная регрессионная модель, которая вводится как

$$\boldsymbol{Y} = \mathbf{X}\mathbf{w} + \boldsymbol{R}, \qquad (108)$$

где $\mathbf{w}$ – вектор из n неизвестных коэффициентов регрессии, которые необходимо определить по обучающей выборке, а $\boldsymbol{R}$ – случайный вектор невязок (или ошибок регрессии).

Классический метод нахождения параметров регрессионной модели – это метод наименьших квадратов. В этом методе производится поиск модели, для которой достигается минимум суммы квадратов невязок. Вектор невязок для каждой реализации случайных векторов будет

$$\mathbf{r}_i = \mathbf{y}_i - \mathbf{X}_i\mathbf{w}. \qquad (109)$$

Тогда целевая функция задается в форме

$$L(\mathbf{w}) = \sum_{i=1}^{M} \|\mathbf{r}_i\|^2 = \sum_{i=1}^{M} \|\mathbf{y}_i - \mathbf{X}_i\mathbf{w}\|^2. \qquad (110)$$

В результате дифференцирования по $\mathbf{w}$ и приравнивания частных производных нулю получается система линейных уравнений, для решения которой существуют стандартные методы.

Метод наименьших квадратов обладает очевидными ограничениями. Критерий среднеквадратичного отклонения можно получить в результате применения метода максимального правдоподобия при использовании модели гауссова шума частного вида. Метод максимального правдоподобия при необходимости позволяет расширить метод наименьших квадратов на случай других моделей шума. Однако в классическом статистическом подходе остается еще одна проблема, к описанию которой мы и перейдем.

Проблема выбора факторов

В общем случае неизвестными в дополнение к параметрам регрессии являются также факторы. Иными словами, функциональная связь F между

независимыми переменными и факторами не дана априори, и ее требуется определить. Выбор факторов влияет на сложность регрессионной модели, и метод максимального правдоподобия, игнорирующий априорные вероятности моделей, перестает давать адекватные результаты, что и составляет корень проблемы классических статистических методов.

Как и в случае обобщенных решающих функций, служащих для формирования новых признаков, факторы обычно выбираются из множества, строящегося на основе некоторой системы функций $f_1, f_2, \ldots$, где $f_i : R^N \to R^{N_1}$. Для некоторого подмножества этого множества функций может быть сформулирована обычная задача регрессии:

$$\boldsymbol{Y} = w_1 f_{i_1}(\boldsymbol{Z}) + \ldots + w_n f_{i_n}(\boldsymbol{Z}) + \boldsymbol{R}. \qquad (111)$$

В этом смысле проблемы выбора факторов в регрессии и выбора признаков в распознавании являются очень близкими.

Часто в качестве системы функций $\{f_i\}$ используется некоторый базис в гильбертовом пространстве, например, система ортонормированных полиномов или вейвлет разложения. В итоге может быть построено сколь угодно много факторов.

Трудность же для классических статистических подходов заключается в том, что, взяв достаточно большое количество факторов регрессии, можно получить равную нулю дисперсию невязок $\mathbf{r}_i$. Причем этот результат может быть достигнут для сколь угодно большого числа регрессионных моделей, опирающихся на разные факторы. Осуществить выбор между этими моделями не представляется возможным, не привлекая дополнительных соображений. Таким образом, использование регрессионных моделей с большим числом факторов приводит к проблеме переобучения, характерной для всех методов машинного обучения, использующих критерии, основанные на точности описания данных и не учитывающие сложность модели.

Рассмотрим задачу полиномиальной аппроксимации в рамках регрессионного анализа (в одномерном случае $N = 1$ и $N_1 = 1$) в качестве наглядного примера, иллюстрирующего проблему переобучения. Для простоты факторы, среди которых осуществляется выбор, будут мономами произвольной степени $X_k = Z^k$ от единственной независимой переменной Z.

Обратимся к рис. 45, на котором представлен набор точек и четыре полинома разных степеней, обладающие минимальной среднеквадратичной ошибкой для данной степени. Параметры полиномов определялись по десяти точкам. Одиннадцатая точка (представленная пустым кружком) в обучающую выборку не входила. Несмотря на то, что полиномы с большим числом параметров n обладают меньшей среднеквадратичной ошибкой, они хуже предсказывают положение точек, не вошедших в обучающую выборку.

В классическом статистическом анализе проблема переобучения решается различными эвристическими методами, такими как методы перекрестной проверки, в которых лишь часть данных используется для оценивания параметров, а другая часть – для определения точности решения. Более элегантное и строгое решение заключается в оценивании априорных вероятностей моделей на основе принципа минимальной длины описания.

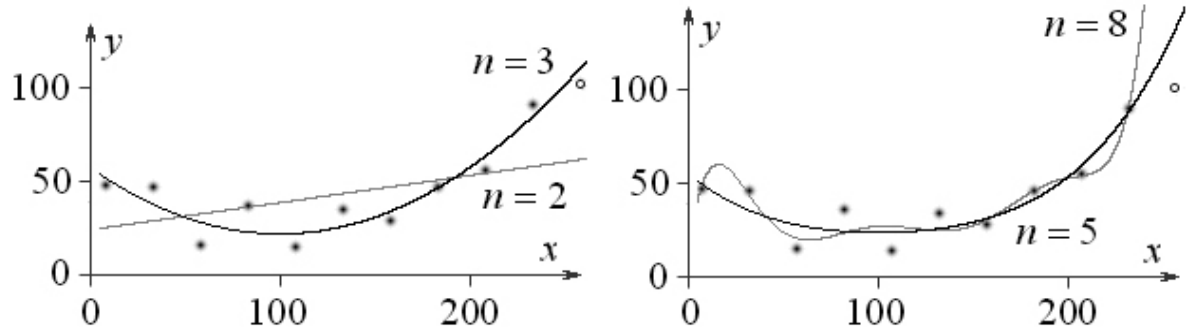


Рис. 45. Пример решения задачи регрессии с неизвестным набором факторов: аппроксимация набора точек полиномами произвольной степени (*n*-1). Минимальное СКО достигается при максимальной степени полинома, что приводит, однако, к наихудшему предсказанию значения функции для точек, не вошедших в выборку (такая точка показана на рисунке незаполненным кружком)

Рассмотрим теоретико-информационный подход к регрессии. Пусть есть источник и приемник информации. И пусть требуется передать значения векторов $\mathbf{y}_i$, при этом значения вектора $\mathbf{z}_i$ получателю сообщения известны. Вместо того чтобы передавать сами значения $\mathbf{y}_i$, можно передать модель, описывающую, как по величинам $\mathbf{z}_i$ восстановить $\mathbf{y}_i$. Чем лучше модель описывает данные (и чем меньше сама модель), тем более компактным будет передаваемое сообщение.

В случае линейной регрессионной модели необходимо передать описание преобразования *F*, вектор коэффициентов регрессии $\mathbf{w}$, а также невязки $\mathbf{r}_i$. Хорошей оценкой длины описания невязок является минус логарифм правдоподобия данных

$$L_r = -\sum_{i=1}^{M} \log_2 p(\mathbf{r}_i). \qquad (112)$$

В действительности, такое определение количества информации для непрерывной случайной величины является не совсем корректным, поскольку здесь не учитывающее необходимость введения опорной плотности. Особенно отчетливо это видно в случае нормального распределения невязок с одинаковой дисперсией по всем направлениям, тогда эту длину описания с точностью до аддитивной константы можно оценивать по формуле

$$L_r = M \log_2 \sigma(\mathbf{r}_i), \qquad (113)$$

где $\sigma(\mathbf{r}_i)$ – среднеквадратичное отклонение. Если дисперсия стремится к нулю, количество информации стремится к минус бесконечности, что является абсурдным результатом. Простым способом преодоления этой проблемы является использование другой оценки:

$$L_r = \frac{M}{2}\log_2\left(\sigma^2(\mathbf{r}_i) + \varepsilon^2\right), \qquad (114)$$

где ε – погрешность задания непрерывных величин. Если же она неизвестна, то можно воспользоваться «честным» способом вычисления длины описания, в явном виде применяя некоторую схему кодирования с вычислением длины получающегося сообщения. Здесь мы, однако, не будем останавливаться на этих технических деталях слишком подробно.

Второй частью сообщения является описание регрессионной модели, которое состоит из описания преобразования F и вектора коэффициентов $\mathbf{w}$. Длина этого описания соответствует минус логарифму априорной вероятности модели, то есть штрафует сложность модели. Наиболее простой оценкой длины описания модели является оценка $L_p = \frac{n}{2}\log_2 M$, которая также используется и в методах распознавания образов.

Этот критерий является сильно упрощенным. При более сложном кодировании может производиться оценивание оптимального числа бит, которые следует отводить под каждый коэффициент регрессионной модели при построении описания. Это позволяет не только более корректно определить длину описания модели, но и оценить точность каждого параметра. Здесь же все параметры считаются равноправными независимо от того, с каким фактором они перемножаются.

Продемонстрируем функционирование принципа МДО в случае самого простого критерия

$$MDL = L_r + L_p = M\log_2\sigma[\mathbf{r}_i] + \frac{n}{2}\log_2 M \qquad (115)$$

и используем его для примера, изображенного на рис. 45. В табл. 4 представлены суммарные длины описания, получающиеся для полиномов различной степени. Из таблицы видно, что минимальная длина описания достигается для полинома второй степени ($n = 3$). Эта модель не только дает наилучшую точность предсказания положения точек, не вошедших в обучающую выборку, но и соответствует истинной функции, использованной для порождения данных.

Таблица 4. Длины описаний для оптимальных полиномов восьми различных степеней, аппроксимирующих набор точек (рис. 45)

n	L_p	$\sigma(\mathbf{r}_i)$	L_r	MDL
1	1.66	20.83	43.81	45.47
2	3.32	18.05	41.74	45.06
3	4.98	8.36	30.64	35.62

4	6.64	8.13	30.23	36.87
5	8.30	7.96	29.93	38.23
6	9.97	7.62	29.32	39.28
7	11.63	7.55	29.16	40.79
8	13.29	6.65	27.33	40.62

Исторически регрессия была одной из первых задач наряду с распознаванием, для которых был применен принцип МДО. И сейчас продолжаются работы в данном направлении, в частности, с помощью принципа МДО решаются задачи выделения сигнала на фоне существенных шумов. Еще одно близкое направление исследований посвящено задаче сегментации, решение которой требует совместного осуществления группирования и регрессии.

Сегментация

Задача сегментации возникает всякий раз, когда требуется некоторый массив данных разбить на однородные порции, причем понятие однородности может быть достаточно сложным. Например, такая необходимость возникает при разделении слитной речи на слова, а также изображения – на различные объекты и фон. Другой пример – это сегментация экспериментальных кривых. В этой задаче, как правило, фиксируются изменяющиеся во времени характеристики некоторого объекта, и построенную таким образом кривую необходимо разбить на отдельные участки, в которых изменение характеристик объекта подвержено какой-то одной закономерности, а различие закономерностей на разных участках кривой связано с переходом между дискретными состояниями объекта.

Таким образом, при сегментации необходимо использовать модели, являющиеся гибридными моделями регрессии и группирования: отдельные элементы исходных данных должны быть объединены в группы, для каждой из которых строится собственная регрессионная модель. Как и в случае задачи регрессии, в качестве исходных данных выступает набор пар векторов $(\mathbf{z}_i, \mathbf{y}_i)_{i=1}^{M}$. Независимые переменные $\mathbf{z}$ обычно являются пространственными или временными координатами, а значениями зависимых переменных – измерения некоторых характеристик в данной точке пространства или в данный момент времени.

При формулировании задачи сегментации практически всегда используется предположение о пространственно-временной однородности нашего мира. Это предположение выражается в способе постановки задачи группирования в рамках сегментации, а именно, не допускается произвольное группирование элементов данных $(\mathbf{z}_i, \mathbf{y}_i)$. Вместо этого в пространстве независимых переменных должны быть выделены области, в каждой из которых применяется своя регрессионная модель (см. рис. 46).

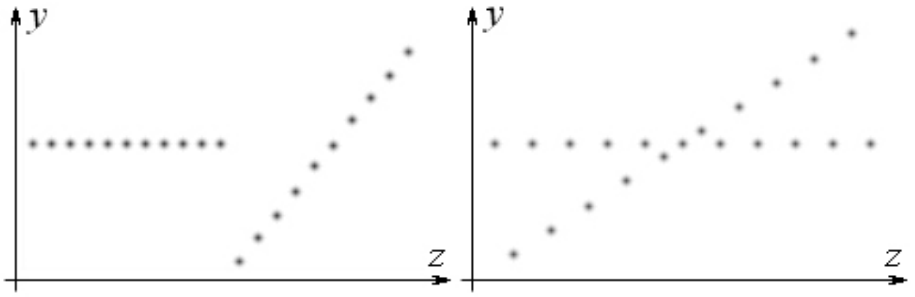


Рис. 46. Пример, иллюстрирующий ограничения, накладываемые на возможные области при постановке задачи сегментации: расположение точек, представленных на рисунке слева, может быть описано двумя регрессионными моделями, введенными на областях (отрезках на оси *z*). Однако группирование точек на основе выделения областей уже неприменимо для данных, изображенных на рисунке справа, хотя для этого случая также достаточно лишь двух регрессионных моделей

В такой постановке задача сегментация может быть сформулирована следующим образом. Пусть есть набор измерений: $(\mathbf{z}_i, \mathbf{y}_i)_{i=1}^{M}$, где $\mathbf{z}_i \in \mathrm{Z} = R^N$ и $\mathbf{y}_i \in \mathrm{Y} = R^{N_1}$. Необходимо сформировать области $G_1,...,G_d$, $G_k \subset \mathrm{Z}$, и для каждого набора пар $\{(\mathbf{z}_i, \mathbf{y}_i) : \mathbf{z}_i \in G_k\}$ построить собственную регрессионную модель: $g_k(\mathbf{z}, \mathbf{w}_k) : G_k \to \mathrm{Y}$, например,

$$\boldsymbol{Y} = w_{k,1} f_{k,1}(\boldsymbol{Z}) + ... + w_{k,n_k} f_{k,n_k}(\boldsymbol{Z}) + \boldsymbol{R}. \qquad (116)$$

Прямой метод решения задачи сегментации заключался бы в том, чтобы перебирать все возможные способы группирования и строить для каждого из них регрессионные модели. Однако это крайне ресурсоемко и непрактично, хотя подобный исчерпывающий поиск и будет давать оптимальный (с точки зрения выбранного критерия) результат. Вместо этого можно применять различные эвристики поиска, аналогичные тем, которые использовались в алгоритме *ISODATA*. В дополнение к приемам слияния и расщепления кластеров (а в данном случае сегментов), можно пользоваться тем, что достаточно часто точки $\mathbf{z}_i$ образуют прямоугольную сетку. Это позволяет определять те точки, которые располагаются на границах областей G_i, и улучшать качество сегментации, пытаясь перенести такие точки из одной области в другую и корректируя при этом параметры регрессионных моделей. Для каждой конкретной задачи могут вводиться и дополнительные эвристики.

Помимо организации поиска необходимо, конечно, решить и вопрос о критерии качества сегментации. Невязки модели сегментации имеют вид

$$\mathbf{r}_i = \mathbf{y}_i - g_k(\mathbf{z}_i, \mathbf{w}_k), \text{ где } k : \mathbf{z}_i \in G_k. \qquad (117)$$

Оценив плотность распределения вероятностей невязок, можно использовать метод максимального правдоподобия. Однако если число сегментов априорно неизвестно, то этот метод будет иметь тенденцию выделять под каждый элемент данных отдельный сегмент с собственной регрессионной моделью, точно описывающей единственную попавшую в

сегмент пару $(\mathbf{z}_i, \mathbf{y}_i)$. Если же допускаются и регрессионные модели разной степени сложности, то другой крайностью будет объединение всех точек в один сегмент с регрессионной моделью, содержащей большое число факторов. Помимо различных искусственных приемов, позволяющих избежать этих двух вырожденных случаев, решением проблемы выбора оптимальной модели сегментации может быть использование теоретико-информационных критериев.

Информационный критерий качества сегментации

Для определения критерия качества модели сегментации представим, что есть отправитель и получатель сообщения. Отправителю известны как величины $\mathbf{z}_i$, так и $\mathbf{y}_i$, в то время как получателю известны значения лишь независимых переменных. Какую информацию необходимо включить отправителю сообщения, чтобы получатель смог однозначно восстановить значения $\mathbf{y}_i$? Для этого получателю необходимо знать параметры всех регрессионных моделей $\mathbf{w}_k$ (и соответствующие факторы) для каждого сегмента, информацию о самих областях G_k, чтобы для каждой точки $\mathbf{z}_i$ была возможность узнать, какую именно регрессионную модель использовать, а также величины невязок $\mathbf{r}_i$.

Правдоподобие данных связано лишь с длиной описания невязок. Методы, не учитывающие другие составляющие, разумеется, будут иметь склонность к переобучению. Можно, например, опустить описание областей, тогда общая длина описания сведется к сумме длин описаний регрессионных моделей каждого сегмента. Способ вычисления этих длин был представлен выше (см., напр., уравнение 115). Такое упрощение часто бывает вполне допустимо, поскольку несильно мешает выбору подходящего числа сегментов. Действительно, так как каждому сегменту соответствует регрессионная модель, которая должна быть описана, то чем больше выбрано сегментов, тем больше будет и суммарная длина описания модели сегментации. Небольшой недоучет сложности модели, состоящей из большого числа сегментов, возникающий при отбрасывании длины описания областей, позволяет получить вычислительно более простые алгоритмы. Еще более сильное упрощение заключается в том, чтобы использовать единую стохастическую модель для описания невязок, задаваемых уравнением (117), а сложность модели вычислять через суммарное число параметров во всех регрессионных моделях. Тогда можно непосредственно использовать уравнение (115).

Тем не менее, в ряде задач именно описание областей G_k является наиболее существенным. Это характерно, например, для задач сегментации изображений. Здесь процесс описания областей по смыслу соответствует определению формы объектов, которая содержит важнейшую информацию, необходимую в задачах анализа изображений. В

связи с этим в общей постановке задачи сегментации вопрос описания областей игнорировать нельзя.

Области могут быть представлены своими границами. Обычно предполагается, что эти границы гладкие, поэтому они могут быть аппроксимированы с помощью некоторого параметрического семейства гладких функций. Это задача регрессии. Если же считать, что поверхности областей гладки почти всюду, то придем к постановке задачи описания областей через их границы в качестве задачи сегментации, но в пространстве размерности на единицу меньше, чем исходное пространство. Эти рассуждения можно продолжить и дальше: в итоге получится цепочка взаимосвязанных общим критерием длины описания задач сегментации в пространствах уменьшающейся размерности.

Рассмотрим два «соседних» уровня:

$$L = \sum_k L\big((\mathbf{z}_i, \mathbf{y}_i) : \mathbf{z}_i \in G_k\big) + \sum_k L(\delta G_k), \qquad (118)$$

где через величину $L\big((\mathbf{z}_i, \mathbf{y}_i) : \mathbf{z}_i \in G_k\big)$ обозначена длина описания регрессионной модели для точек, попавших в область G_k (вычисляемая, например, с помощью уравнения 115), а через величину $L(\delta G_k)$ – длина описания границы соответствующей области.

Если величины $L(\delta G_k)$ не учитываются, то границы областей будут получаться изрезанными: любой локальный выброс, произошедший из-за шума, может быть отнесен к неверной области лишь на основе значений зависимых переменных в данной точке, невзирая на ее пространственное положение. В то же время, чем проще граница области, тем меньше ее длина описания $L(\delta G_k)$. Значит, учет этих величин должен приводить к получению более гладких границ областей и более адекватного результата сегментации. Если этот учет корректен, то также не будут получаться и слишком сглаженные границы, поскольку это приведет к понижению правдоподобия данных в рамках регрессионных моделей (однородности содержания отдельных областей).

Таким образом, задачи, которые можно было бы решать отдельно (собственно сегментацию исходных данных и описание границ областей), оказываются связанными общим критерием качества. Первая сумма в уравнении (118) соответствует качеству регрессионных моделей, а вторая сумма может быть проинтерпретирована, как критерий качества группирования. Как и при совместном решении проблем выбора признаков и кластеризации, в данном случае принцип минимальной длины описания позволил связать две отдельные на первый взгляд задачи.

Вопросы и упражнения

1. В чем заключается задача регрессии? В чем сходство и отличие этот задачи и задачи распознавания образов?

2. В чем заключается проблема выбора факторов? Какая связь между этой проблемой и проблемой выбора признаков в методе обобщенных решающих функции в распознавании образов?
3. Как проблема переобучения проявляется в задаче регрессии?
4. Как принцип минимальной длины описания позволяет решить проблему выбора факторов?
5. Какие две задачи совмещает в себе задача сегментации?
6. Какие наиболее значимые приложения имеют методы сегментации?
7. Из каких слагаемых складывается сложность модели сегментации?

СРС.5. Приложения методов восстановления формальных грамматик к задачам анализа естественного языка

На основе единственной формальной грамматики можно попытаться описать всю структуру некоторого естественного языка. При этом формальные грамматики позволяют описывать «грамматические классы», включающие символы разных уровней (фонемы или буквы, морфемы, слова и т.д.), а также цепочки, состоящие из терминальных и нетерминальных символов различных уровней. Однако более практичным оказывается разделение задачи описания структуры языка на ряд подзадач, связанных с анализом элементов его синтаксиса и морфологии. Это позволяет существенно упростить задачу обучения и использовать в каждой из подзадач грамматики частного вида. Однако для систем машинного обучения общего назначения остается вопрос: нужно ли априорно вводить в формальные грамматики такое достаточно жесткое разделение на уровни, приписываемые естественному языку, или же оно может быть получено автоматически при восстановлении грамматики более общего вида, допускающего смешанные классы и цепочки. Здесь же мы рассмотрим несколько частных применений формальных грамматик для анализа отдельных элементов структуры языка.

Обучение фразам

Рассмотрим сначала задачу обучения фразам, имеющую непосредственное отношение к проблеме восстановления синтагматической структуры речи. Эта задача может быть сформулирована как задача восстановления стохастических грамматик следующего вида.

$$V_N = \{S\}, P_S = \left\{ S \xrightarrow{P(a_1 a_2 ... a_n | S)} a_1 a_2 ... a_n S \mid a_i \in V_T, n = 1,..., N \right\}, \quad (119)$$

где элементы $a_i \in V_T$ – это слова некоторого языка над V_T.

Воспользуемся рассмотренной в лекциях схемой: сначала сконструируем некоторую «беспорядочную» или *ad hoc* грамматику, а затем будем ее улучшать (в смысле длины описания) путем последовательных преобразований множества правил вывода.

Поскольку в грамматиках вида (119) отсутствуют дополнительные нетерминальные символы, исходная грамматика должна быть соответствующего вида. Для образца языка $S_t = \{\alpha_i\}_{i=1}^{t}$ в качестве исходной можно выбрать *ad hoc* грамматику, содержащую правила $P = \{S \to \alpha_i\}_{i=1}^{t}$, а далее производить разбиение этих правил на правила вида $S \to \alpha_i' S, S \to \alpha_i'' : \alpha_i' \alpha'' = \alpha_i$. Другой способ восстановления грамматики заключается в том, чтобы начинать с «беспорядочной» грамматики, содержащей правила $P = \{S \to aS \mid a \in V_T\}$, и выполнять операцию конструирования: из двух правил $S \to \alpha S$ и $S \to \beta S$ формировать правило $S \to \alpha\beta S$. Операция конструирования может быть выполнена для любых двух правил, однако она будет иметь смысл только тогда, когда результирующее правило $S \to \alpha\beta S$ может быть использовано много раз при порождении данного текстового корпуса (иными словами, когда сочетание $\alpha\beta$ встречается достаточно часто в тексте). Поскольку не следует ожидать наличия длинных стандартных фраз, начинать восстановление грамматики с «беспорядочной» грамматики эффективнее.

Таким образом, восстановление следует начинать с грамматики вида $P = \{S \to aS \mid a \in V_T\}$. Эта грамматика будет содержать столько правил, сколько встречается различных слов в образце языка. Вероятность применения правила будет равна вероятности появления соответствующего слова. После формирования на основе двух правил $S \to \alpha S$ и $S \to \beta S$ дополнительного правила $S \to \alpha\beta S$ вероятности использования первых двух правил понизятся (и могут стать равными нулю, если данные слова встречаются только вместе); при этом уменьшится и длина цепочки правил, приводящих к генерации образца языка, однако сложность грамматики может возрасти.

Посмотрим на конкретном примере, какие словосочетания выделяются в рамках грамматики вида (119) при ее восстановлении индукцией на основе критерия МДО. Для этого используем небольшой тестовый корпус, включающий несколько классических художественных произведений (общее количество символов – 4,5 млн.). При этом в качестве терминальных символов используем словоформы (разные формы одного и того же слова будем считать различными), не исключая из рассмотрения частицы, союзы, предлоги и т.д., чтобы использовать минимальную информацию о языке.

Ниже приведено несколько цепочек слов, полученных на некотором корпусе в результате операций конструирования, которые были выполнены первыми и принесли наибольший выигрыш в длине описания:

«может быть» (n_1=994, n_2=1326, n_{12}=522)
«как будто» (n_1=6171, n_2=615, n_{12}=546)
«о том» (n_1=2339, n_2=868, n_{12}=454)
«к нему» (n_1= 4110, n_2=371, n_{12}=369)
«потому что» (n_1= 909, n_2=12577, n_{12}=588)

«тотчас же» (n_1=492, n_2=3555, n_{12}=347)
«то что» (n_1= 2879, n_2=11989, n_{12}=729)
«у него» (n_1= 2571, n_2=1410, n_{12}=366)
«об этом» (n_1= 558, n_2=684, n_{12}=227)
«с ним» (n_1= 7791, n_2=778, n_{12}=386)

Здесь n_1 и n_2 – число вхождений в текстовый корпус первого и второго слова соответственно, а n_{12} – число вхождений цепочки этих слов.

Также на первых шагах работы алгоритма были получены такие цепочки, как «несмотря на», «как бы», «если бы» и т.д. Помимо таких шаблонов в качестве устойчивых сочетаний слов были выделены имена героев взятых литературных произведений, например, «Сергей Иванович», «Катерина Ивановна».

Интересен большой выигрыш в длине описания от введения таких сочетаний слов, как «к нему», «у него», «с ним». Выгодность введения правила подстановки, порождающего форму некоторого слова (или словосочетание) вместе с примыкающим предлогом, еще более явно видна на следующих примерах:

«в тайне» (n_2=4, n_{12}=4)
«в частности» (n_2=4, n_{12}=4)
«в [здравом рассудке]» (n_2=4, n_{12}=4)
«в [трех шагах]» (n_2=4, n_{12}=4)
«в [первые минуты]» (n_2=4, n_{12}=4)
«на холме» (n_2=7, n_{12}=4)
«на западе» (n_2=3, n_{12}=3)
«на морозе» (n_2=3, n_{12}=3)
«на левом» (n_2=7, n_{12}=4)
«в левом» (n_2=7, n_{12}=3)
n(«в») =15593, n(«на»)=9675.

Здесь выигрыш от операции конструирования незначительный, но устойчиво положительный. Видно, что словоформы «западе» или «морозе» (а также многие другие) в нашем текстовом корпусе встречаются только после предлога «на», а словоформы «тайне», «частности» – только после предлога «в». Словоформа «левом» встретилась только после предлогов «на» и «в», в связи с чем на ее основе образовались два различных правила, порождающие сразу цепочки «на левом» и «в левом», а исходный символ «левом» исчез из текстового корпуса как самостоятельный элемент. Если бы рассматривался слитный текст, не разделенный на слова, то было бы сложно отличить приставки от предлогов (между ними есть определенная разница, в частности, предлоги связаны с падежами слов, в то время как приставки могут быть присоединены к различным формам одного и того же слова). Это говорит о том, что разделение на уровни: морфемы, слова, фразы – не вполне четкое.

Большой суммарный выигрыш в длине описания получился от введения правил порождения таких сочетаний слов, как

«по крайней мере» (n=111)
«в самом деле» (n=100)
«несмотря на то что» (n=72)
«то же время» (n=70)

«для того чтобы» (*n*=63)
«во всяком случае» (*n*=30)
«точно так же» (*n*=42)
«с тех пор» (*n*=58)
«в то время как» (*n*=49)

Они действительно могут быть охарактеризованы как неделимые фразы. Вместе с тем, среди них оказались такие сочетания слов, как, например,

«я не могу» (*n*=130)	«я могу» (*n*=45)	«я могу быть»(*n*=5)	«что же я могу» (*n*=3)
«я не понимаю (*n*=40)	«я понимаю» (*n*=47)	«как я понимаю» (*n*=3)	«я понимаю что» (*n*=3)
«я не хочу» (*n*=37)	«я хочу» (*n*=60)	«я хочу знать» (*n*=4)	«я хочу быть» (*n*=3)

Это частично обусловлено жанром художественных произведений, вошедших в корпус (к примеру, если бы корпус был составлен из научных статей, то эти сочетания слов не были бы выделены). Тем не менее, высокая частота встречаемости таких фраз не случайна. При изучении языка подобные шаблонные фразы обычно усваиваются до усвоения грамматических правил языка, позволяющих правильно конструировать любые предложения.

Приведем также некоторые наиболее длинные сочетания слов, введение для которых собственного нетерминального символа позволяет уменьшить суммарную длину описания:

«во что бы то ни стало» (*n*=13)
«как бы то ни было» (*n*=9)
«в то же время» (*n*=32)
«но в то же время» (*n*=11)
«как ни в чем не бывало» (*n*=6)
«на одном и том же месте» (*n*=3)
«до тех пор пока не» (*n*=8)
«как раз в то время когда» (*n*=3)
«в одно и то же время» (*n*=6)

Но и среди наиболее длинных сочетаний слов встретились такие, как

«не спуская с него глаз» (*n*=5)
«ты не можешь себе представить» (*n*=4)
«я до сих пор не могу» (*n*=3)
«я только хочу сказать что» (*n*=3)

Как и сочетания слов «я не хочу», «я не могу» и т.д., эти сочетания слов, безусловно, являются стереотипными и составляют элемент синтагматической структуры речи. Однако видно и заметное различие между фразами «во что бы то ни стало» и «ты не можешь себе представить»: первой фразе, в отличие от второй, соответствует один концепт (это проявляется, в частности, в том, что она переводится на иностранный язык как единое целое, а не по частям).

Таким образом, в результате восстановления грамматики (119) определяются идиоматические выражения, часто используемые фразы, словоформы с примыкающими к ним служебными словами. Это, видимо, свидетельствует об ограниченности вида выбранной грамматики. Во-первых, отсутствует учет морфологии слов. Во-вторых, в грамматике отсутствует возможность введения классов слов.

На практике для решения задачи обучения фразам формальные грамматики используются редко; вместо этого понятие фразы вводится априорно в явном виде. Тем не менее, рассмотрение этой задачи как задачи восстановления грамматик позволяет определить те ограничения, которые используются здесь для представления информации. На основе вида правил грамматики (119) можно заключить, что при решении задачи обучения фразам абсолютно не учитывается контекст фразы, в также категории слов, составляющих фразу. Может быть поставлена задача выявления структуры фраз, для чего слова в текстовом корпусе заменяются символами, обозначающими соответствующие словам части речи (соотнесение слов и соответствующих им частей речи обычно выполняется вручную). В остальном задача выявления структуры фраз и предложений ничем не отличается от задачи обучения фразам.

Часто, чтобы исследовать именно проблему обучения фразам, отодвигаются еще дальше от предыдущего уровня, в частности, рассматривают только слова, наделенные самостоятельным смыслом, не делают различий между формами одного слова и т.д. Это, однако, требует введения информации о языке, которая сама может быть предметом автоматического анализа.

Разделение морфов на классы

Для объединения отдельных букв в части слова может использоваться грамматика вида (119) и примерно такой же алгоритм, что и для поиска устойчивых словосочетаний. Если же словарь морфов уже выделен, то может возникнуть необходимость их разделения на классы (приставки, корни, суффиксы, окончания). При этом интересной является задача, в которой классы морфов не заданы априори. В целях решения задачи автоматического обучения морфологии языка применим восстановление автоматных грамматик.

Алгоритм восстановления автоматных грамматик описывался в лекциях. Этот алгоритм начинает работу с *ad hoc* грамматики и производит последовательное объединение нетерминальных символов, приводящее к максимальному уменьшению длины описания.

В качестве терминальных символов здесь выступают морфы. Нетерминальные символы соответствуют классам морфов. Посмотрим, насколько эти классы, полученные при автоматическом восстановлении, будут совпадать с классами, выделенными в морфологии.

Для этого возьмем весьма небольшую обучающую выборку словоформ, разделенных на морфы:

у-сматр-ива-ет	у-сматр-ива-ют	при-сматр-ива-ют-ся	у-смотр-им	при-смотр-им
при-лет-а-ет	с-лет-а-ет-ся	с-лет-а-ют-ся	с-лет-у	при-лет-у
с-лет-е	при-лет-е	при-лет-а-ют	у-лет-а-ют	с-лет-им-ся
у-лет-им	при-ют-им	при-ют-им-ся	у-ют-е	у-ют-у
при-ют-е	при-ют-у			

Начальная грамматика будет включать правила:

$$S \to \text{у}\, A_1;\ A_1 \to \text{сматр}\, A_2;\ A_2 \to \text{ива}\, A_3;\ A_3 \to \text{ет};$$

$$S \to \text{у}\, A_4;\ A_4 \to \text{сматр}\, A_5; A_5 \to \text{ива}\, A_6; A_6 \to \text{ют};$$

и т.д.

Мы здесь не будем объединять правила при формировании первоначальной грамматики, поэтому словоформы у-сматр-ива-ет и у-сматр-ива-ют будут давать два правила, в левой части которых стоит начальный символ: $S \to \text{у}\, A_1$ и $S \to \text{у}\, A_4$. Неправомерность объединения нетерминальных символов на этапе формирования первоначальной грамматики хорошо понятна на примере двух словоформ: у-лет-а-ют и у-лет-им. Если для них сразу же производить объединение нетерминальных символов, то в результате получатся правила: $S \to \text{у}\, A$, $A \to \text{лет}\, B$, $B \to \text{а}\, C$, $B \to$ им, $C \to$ ют. То есть суффиксальный морф «а» будет объединен в один класс с флексийным морфом «им».

В нашем эксперименте грамматический вывод представлял собой последовательность итераций, на каждой из которых перебирались все пары нетерминальных символов, и выбирались такие два символа, объединение которых давало максимальное уменьшение длины описания (подобный перебор для больших корпусов, разумеется, будет неприемлемым, и потребуется вводить дополнительные эвристики).

Сначала производились объединения правил вида $A_i \to o$ для разных i, где o – некоторое окончание. Такие объединения не увеличивали длину описания текстового корпуса посредством грамматики, но сокращали длину описания самой грамматики за счет уменьшения числа правил. При этом в системе грамматических правил появлялись правила с идентичной правой частью, но разными левыми частями, например, $A_i \to \text{лет}\, B$, где $B \to$ у. Далее такие нетерминалы A_i объединялись между собой, давая один класс.

Видно, что на этом этапе грамматического вывода каждый нетерминальный символ соответствует классу, состоящему лишь из одного терминального символа. Тогда возникает вопрос, что представляли собой исходные классы до объединения? И если при объединении нетерминальных символов происходит обобщение, то в чем именно оно заключается? Заметим, что исходное число нетерминальных символов соответствует не числу терминальных символов, а числу их вхождений в текстовый корпус. Иными словами, каждое вхождение некоторого терминального символа считается уникальным. Эта уникальность обеспечивается контекстом, в котором этот символ появляется. То есть некоторый нетерминальный символ описывает некоторый терминальный символ в некотором контексте. При первоначальном объединении нетерминальных символов происходило абстрагирование от того, в каком контексте встречается тот или иной символ (морф). Отметим, что это абстрагирование является неполным – определенная зависимость от контекста остается. В противном случае, такая процедура была бы лишена смысла, и исходную грамматику следовало бы формировать так, чтобы число нетерминальных символов соответствовало числу терминальных

символов (если не считать того, что первые символы в словах генерируются из начального символа *S*).

В результате первоначального абстрагирования от контекста, такие слова, как при-ют-у и при-лет-у, оказываются порождаемыми правилами вида: $S \to$ при A_1, $S \to$ при A_2, $A_1 \to$ ют B, $A_2 \to$ лет B, $B \to$ у. Теперь уже объединение таких нетерминальных символов, как A_1 и A_2, соответствует не просто объединению в класс одного и того же терминального символа, но встречающегося в разных контекстах, а объединению различных терминальных символов. В результате все корневые морфы объединяются в один класс, после чего происходит постепенное объединение остальных морфов.

Приведем конечную грамматику, полученную в результате грамматического вывода на основе нашей обучающей выборки:

$S \to$ у A | при A | с A
$A \to$ сматр B | лет D | смотр D | ют D | лет B
$B \to$ ива C | а C | ся
$C \to$ ет | ют | ют B | ет B
$D \to$ им | им B | е | у

Здесь каждая строка описывает совокупность правил с одинаковой левой частью, но разными правыми частями.

Полученные классы в достаточной степени соответствуют классам морфов, известным из морфологии. Одно небольшое отклонение заключается в том, что флексийные морфы разделены на две группы «ют», «ет» и «им», «е», «у». Это разбиение полностью обусловлено малым текстовым корпусом: «ют» и «ет» в нем встречаются только после суффиксов «а» и «ива», в то время как остальные окончания встречаются только сразу после корней. Менее обоснованным выглядит объединение суффиксальных морфов «ива» и «а» с постфиксальным морфом «ся». Вероятно, причиной этого является то, что алгоритм редукции грамматики является «жадным» алгоритмом (в связи с чем дает неоптимальное решение).

Отметим, что в исходной грамматике приставки «у», «при», «с» сразу же были объединены в один класс, так как порождались из начального символа *S*. Если бы в корпусе присутствовали словоформы без приставок, то входящие в них корневые морфы оказались бы также в этом классе. В связи с этим при формировании исходной грамматики следует вводить несколько начальных символов, из каждого из которых будет порождаться собственное слово.

Отметим также, что окончание «ют» и корень «ют» оказались в разных классах. При разборе некоторого слова в рамках данной грамматики встреченное «ют» будет распознано правильно в зависимости от своего положения в слове. Таким образом, начальное разделение символов на классы по контексту позволяет автоматически решить проблему их омонимии. Тезис о неоднозначности естественного языка

является общепринятым. Однако эта неоднозначность по большей части возникает из-за рассмотрения фрагмента текста вне всего контекста. Так, морф «ют» имеет неоднозначное толкование, но только до тех пор, пока не указано слово, в которое он входит. Слово «коса» имеет несколько значений, но эта неоднозначность разрешается, когда это слово используется в конкретном предложении. В предложении «спрос рождает предложение» выбор между двумя кандидатами на роль подлежащего, скорее всего, мог бы быть разрешен, если бы это предложение использовалось в определенном контексте. Однако уже при выделении классов слов возникают определенные проблемы с использованным нами подходом: слов очень много, а редуцировать грамматику, в которой под каждое вхождение в текстовый корпус каждого слова выделен собственный нетерминальный символ, крайне ресурсоемко. Применение же общих процедур грамматического вывода для выделения структуры целого текста (в терминах некоторой формальной грамматики), состоящего из многих предложений, практически неосуществимо.

Задача о выделении классов морфов тесно связана с задачей выделения классов слов, поскольку многие морфы являются характерными для определенных грамматических категорий, но мы для простоты изложения рассмотрим эти задачи раздельно.

Построение классов слов

Тексты на естественном языке являются, как правило, не только грамматически правильными, но и семантически корректными. К примеру, в малом числе текстов может встретиться фраза вида: «оранжевые мысли яростно молчат». В связи с этим можно попытаться извлечь из текстов и некоторую семантическую информацию, например, объединение близких по смыслу слов в группы. Считается, что слова близки по смыслу, если они встречаются в одинаковых контекстах.

Задание отношения эквивалентности на множестве слов на основании совпадения контекстов, в которых эти слова допустимы, является давно принятым в математической лингвистике. Вводимые через подобное отношение эквивалентности дистрибутивные классы (или семейства) слов являются основой для формального определения различных грамматических категорий. В математической лингвистике понятие допустимости слова в данном контексте подразумевает только грамматическую правильность построения соответствующего предложения без учета его осмысленности. Но, как уже указывалось, если определение классов слов осуществляется на практике по текстовому корпусу, то вовлеченные в этот процесс предложения будут корректными и в семантическом смысле, а значит, полученные классы слов будут захватывать как синтаксические, так и семантические категории.

Рассмотрим задачу построения классов слов на основе текстового корпуса как задачу грамматического вывода. Отдельные слова здесь – терминальные символы некоторой формальной грамматики. Классам слов,

очевидно, должны соответствовать нетерминальные символы. Контекст может моделироваться по-разному в зависимости от выбранного вида правил вывода. Мы рассмотрим восстановление регулярных грамматик с правилами вида $A \to aB$ и $A \to b$. Правило $A \to aB$ может быть проинтерпретировано таким образом: слово «a» относится к классу слов A и встречается слева от некоторого слова из класса B.

Эта задача похожа на задачу разделения морфов на классы. Однако здесь есть и некоторые особенности.

Во-первых, в исходно формируемой грамматике нетерминальные символы могут быть выделены под каждый терминальный символ (слово) или под каждое вхождение терминальных символов в текстовый корпус. В первом случае все возможные значения некоторого слова, которые могут быть различены по контексту, сливаются в один класс. Однако при построении классов слов проблемой омонимии часто пренебрегают. Как отмечалось выше, это позволяется существенно уменьшить число нетерминальных символов в начальной грамматике (или начальное количество классов слов), что приводит к существенному повышению вычислительной эффективности. Используя полученные таким образом классы слов, можно разделять омонимы для каждого слова по отдельности.

Во-вторых, нужно решить, следует ли использовать в качестве терминальных символов слова или нужно различать их разные формы. При использовании словоформ, очевидно, будет наблюдаться тенденция к формированию классов на основе падежей (в связи с появлением определенных форм слов рядом с соответствующими предлогами), а также на основе числа и рода (например, рядом со словоформой «цветом» может появиться «красным» или «зеленым», а рядом со словоформой «цвету» – «красному» или «зеленому», поэтому словоформы «красным» и «красному» не будут объединены в один класс, а «красному» и «зеленому» – будут). Как было установлено выше, предлоги со словами, находящимися в соответствующих падежах, образуют устойчивые сочетания, которые выделяются в грамматиках вида (119). Именно поэтому при использовании слов в качестве терминальных символов грамматики необходимо не только каждую словоформу в текстовом корпусе привести к исходной форме, но также исключить из рассмотрения предлоги и частицы. Объединение словоформ в парадигмы – задача морфологического анализа. Выделение цепочек слов тоже является отдельной задачей, рассмотренной выше. Неясности, возникающие с постановкой задачи выделения классов слов, связаны с вопросом, где проводить границу между этой задачей и другими задачами восстановления структуры языка. Разделение на подзадачи необходимо для создания практически применимых алгоритмов. В то же время, между этими подзадачами остаются взаимосвязи, которые нельзя игнорировать.

Часто для простоты пользуются следующим приемом: при сравнении слов, длиннее пяти символов, игнорируются различия в последних трех символах, а также исключаются из рассмотрения слова в три и менее

символа. Это грубый, но простой способ в русском языке перейти от словоформ к словам.

В качестве текстового корпуса используем текст некоторых лекций по искусственному интеллекту. Пусть при генерации первоначальной грамматики на основе каждого слова a формируется один нетерминальный символ A и набор правил подстановки $A \to aB_i$, где символу B_i соответствует слово b_i, встретившееся справа от слова a. Далее будет осуществляться редукция этой грамматики под управлением принципа МДО.

В результате могут быть получены классы, включающие разделение как по грамматическим, так и по семантическим признакам. Из-за использования малого по объему текстового корпуса (что необходимо при применении достаточно общих процедур грамматического вывода) многие выделенные классы содержат и посторонние элементы.

К примеру, были выделены такие классы:

A: иметься; использоваться; оказываться; являться.

B: следующий; неоднозначный; некоторый; любой.

C: малость; возрастание; убывание; перечисление.

D: напротив; во-первых; во-вторых; в-третьих; известно; полагая.

Видна не только связь между словами в этих классах по принадлежности к близким грамматическим категориям, но и определенная связь по смыслу. Если в классе *B* три из четырех слов – местоименные прилагательные (имеющие специфичную функцию указательных слов), то приведенный ниже класс содержит прилагательные с более абстрактными значениями:

E: сенсорный; универсальный; конкретный; синонимичный.

Приведем еще несколько выделенных классов:

F: поиск; редукция; выбирать; терять; ожидать; пригодиться.

G: термин; правило; ошибка; неправильный; новый; допустимый.

H: предпочтительный; затруднительный; неукорачивающий; NP-полный; конструирование; манипуляция.

I: исправленный; нестрогий; независимый; собственный; кодировать; трактовать.

J: формализм; трудность; сторона; приложение; целевой; ненадежный.

Далеко не все выделенные классы объединяют слова, имеющие четкое сходство по грамматическим категориям или смыслу, однако определенная взаимосвязь между ними прослеживается. К примеру, в классе *G* присутствуют существительные и прилагательные, так или иначе относящиеся к описанию данных посредством моделей (связь между словами «правило», «ошибка», «неправильный» достаточно очевидна). В классе *F* присутствуют слова «поиск» и «редукция», которые в достаточной степени ассоциируются со словами «выбирать» и «терять».

Недостаточное разделение слов по грамматическим категориям вызвано использованием в данном тексте предложений с достаточно

сложной структурой, а также с нестрогостью правил построения предложений при использовании небольшой обучающей выборки. Разделение слов на грамматические категории может также вестись путем анализа их морфологии. Корректное использование двух видов информации: синтаксической и морфологической, – необходимо для более качественного решения задачи построения классов слов.

Вместо задачи построения классов слов часто ставится задача построения тезауруса – иерархически сгруппированных классов слов. В представлении в форме тезауруса фиксируется степень близости различных слов внутри классов, что является более информативным, чем представление множества слов в виде планарных классов. При этом получаемые в результате классы в тезаурусе также подвержены совместному влиянию как семантических, так и грамматических категорий. При построении тезауруса по текстовым корпусам английского языка часто грамматические категории превалируют (смысловое объединение слов различных грамматических категорий отсутствует, что может быть следствием более жесткого, чем в русском языке, синтаксиса). Внутренняя структура отдельных классов оказывается не слишком показательной, однако определенную дополнительную информацию несет. На рис. 47 приведен пример фрагмента тезауруса.

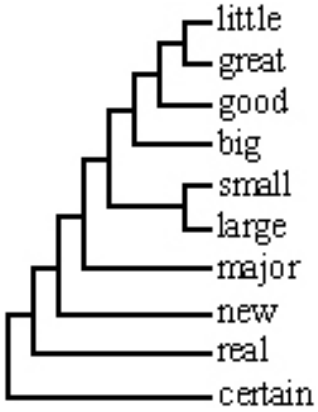


Рис. 47. Элемент автоматически построенного тезауруса: один из найденных классов слов, иерархически разбитый на подклассы

Лингвистическая информация о грамматических категориях является более доступной, чем информация о семантических категориях слов. В связи с этим на практике бывает полезным устранить влияние грамматики на процесс построения тезауруса, воспользовавшись априорной информацией о ней. К примеру, могут параллельно строиться классы для существительных и глаголов: существительные объединяются в классы на основе частот их появления рядом с глаголами (или классами глаголов), а для глаголов для той же цели в качестве опорных используются классы существительных (при этом формальные грамматики обычно не используются). Такая модификация задачи построения тезауруса может быть предпочтительней с практической точки зрения, но для системы

машинного обучения она подразумевает наличие априорного умения распознавания общих грамматических классов – подлежащих и сказуемых.

Обсуждение

Таким образом, формальные грамматики в рамках единого представления позволяют описывать разные структурные особенности языка: морфы и их классы, слова, их классы и устойчивые словосочетания и т.д. Однако автоматическое восстановление формальной грамматики, которая бы захватывала все эти структурные особенности, является крайне ресурсоемким особенно на больших текстовых корпусах. Как правило, на практике задачи по обнаружению морфов, формированию их классов, построению тезауруса, выделению как отдельных фраз, так и структуры целых предложения решаются раздельно. Существование возможности разделения (хотя и неполного) этих задач представляет несомненный интерес. В данном случае это разделение делалось человеком на основании своего опыта. Перечисленные частные задачи получаются из общей задачи восстановления грамматики путем введения ограничений на вид правил подстановки и выбором надлежащего алфавита терминальных символов. Формальные грамматики дают методологическое основание для синтеза из решения частных задач решение общей задачи в результате снятия этих ограничений. Однако нерешенным остается фундаментальный вопрос о том, как такое разделение и последующий синтез выполнять автоматически.

В частности, если система машинного обучения начинает свою работу с представления, в котором в качестве терминальных символов выступают буквы или фонемы, то как провести четкую границу между уровнями букв, морфем, слов, словосочетаний? Иными словами, как, к примеру, из исходного текста как совокупности букв вывести понятие (элемент информационного представления) слова, если оно не заложено в систему априори? Подчеркнем здесь еще раз, что в рамках такого общего представления как формальные грамматики, потенциально допустимы объекты со структурой, состоящей из смеси любых терминальных и нетерминальных символов. Понятие слова в некоторой формальной грамматике можно считать выделенным только тогда, когда в ней образован нетерминальный символ, порождающий все прочие символы, соответствующие конкретным словам. Однако при практическом обучении по текстовым корпусам на формирование такого нетерминального символа надеяться не приходится: выигрыш от введения одного класса типа «слово» может быть слишком незначительным по сравнению с использованием нескольких столь же высокоуровневых классов. Более того, при последовательном слиянии нетерминальных символов слишком велика вероятность на более ранних этапах принять локальное решение об объединении в один класс объектов разных уровней.

При независимом решении таких задач, как выделение классов слов или морфем, формирование словосочетаний и т.д. в качестве

терминальных символов используются нетерминальные символы грамматик, полученных при решении более низкоуровневых задач (например, формирования алфавита морфем). Хотя те же языковые объекты и зависимости между ними могут быть описаны в рамках одной формальной грамматики общего вида, это представление будет отличаться от представления в виде иерархии грамматик частного вида. Отличие между этими представлениями заключается не столько в выразительной силе представлений, сколько в вычислительной сложности задач, решаемых с их помощью. К примеру, в случае использования системы грамматик при работе со словами, как неделимыми объектами, можно абстрагироваться от структуры слов и игнорировать все правила подстановки, относящиеся к грамматикам других уровней. В случае же использования одной грамматики общего вида правила подстановки в ней не организованы в какую-либо структуру.

Стратегии поиска, исследуемые в рамках эвристического программирования, имеют большое значение для процесса грамматического вывода. Здесь мы использовали метод последовательного улучшения грамматики в результате ее локального изменения: выполнения операций слияния или конструирования для пар нетерминальных символов. В то же время могут также использоваться генетические алгоритмы, имитация отжига и другие стохастические методы оптимизации. Стратегии поиска могут варьироваться от «жадных» алгоритмов до полного перебора. Эти две крайности имеют много общего с двумя крайними способами описания данных – *ad hoc* и «беспорядочной» моделями. «Жадный» алгоритм является одним из наиболее простых (в смысле вычислительной сложности, а не длины описания) алгоритмов поиска, но он может давать весьма неточное решение. Полный перебор дает точное решение оптимизационной задачи, но сам обладает большой вычислительной сложностью. Очевидно, оптимальный алгоритм поиска должен быть компромиссом между сложностью поиска и его точностью. Также очевидно, что положение этого оптимума зависит от исходных данных. Так, полный перебор может оказаться лучшей альтернативой при решении оптимизационной задачи на малом объеме исходных данных, но при увеличении последнего он окажется неприемлемым.

Говоря об оптимизации процедуры грамматического вывода, нельзя не упомянуть об еще одной возможности ее осуществления, связанной с инкрементным обучением. На данный момент наиболее популярным является использование больших текстовых корпусов. Вместо этого можно начинать с малых текстовых корпусов с ограниченным лексиконом и упрощенным строением предложений и производить постепенное усложнение текстов. Грамматический вывод, производимый по новым тестам, будет базироваться на выделенных на предыдущих этапах обучения лингвистических единицах и грамматических правилах, что существенно упростит процедуру вывода и сделает ее более надежной. К

сожалению, формальные грамматики и методы их восстановления, допускающие инкрементное обучение, исследованы мало, а сам этот подход требует специально составленных обучающих выборок (здесь могут помочь книги для детей разного возраста).

Вопросы и упражнения

1. Какие подзадачи могут быть выделены в общей проблеме анализа структуры естественного языка?
2. В чем смысл выделения подзадач при анализе естественных языков?
3. Какие типы формальных грамматик обычно используются в задачах анализа естественного языка?
4. Правила какого вида используются для объединения элементов языка в группы, а какого – для описания закономерностей в цепочках его элементов?
5. На основе чего определяется отношение эквивалентности на множестве слов?
6. В чем отличие построения тезауруса от разделения слов на классы?
7. Какие упрощения вводятся в задаче разделения слов на классы?
8. В чем заключается возможность инкрементного обучения в задачах анализа естественного языка?
9. Какого типа алгоритмы поиска используются при восстановлении грамматик индукцией, и как можно применять методы эвристического программирования для повышения качества анализа?

СРС.6. Алгоритм *ID*3 автоматического построения деревьев решений

Кратко сформулируем задачу построения дерева решений. В этой задаче дано конечное множество атрибутов $\mathrm{X} = X_1 \times X_2 \times ... \times X_N$, где N – число атрибутов, и конечное множество классов $\mathrm{A} = \{a_1, a_2 ... a_d\}$, где d – число классов. По обучающей выборке $D = ((x_1, c_1), (x_2, c_2), ..., (x_M, c_M))$, где $x_i \in \mathrm{X}, c_i \in \mathrm{A}$, требуется восстановить отображение (алгоритм), действующее из X в A, задаваемое в виде дерева, в каждом узле которого осуществляется проверка одного из атрибутов, и в зависимости от значения этого атрибута происходит переход к соответствующему дочернему узлу. В листьях дерева решений располагаются номера классов, которые и выступают в качестве результата классификации. В стохастическом дереве решений в каждом листе может находиться несколько выходных классов с указанием вероятности для каждого из них.

Мы рассмотрим эвристический метод построения деревьев решений. Отметим, однако, что деревья решений также предоставляют возможность непосредственного применения генетических алгоритмов и

эволюционного программирования. Для применения генетических алгоритмов достаточно лишь немного модифицировать приведенную выше схему кодирования дерева решений в форме символьной строки, а для применения методов эволюционного программирования вполне удобно исходное представление в форме дерева решений.

Легко заметить, что полный перебор деревьев нереализуем из-за своей вычислительной сложности: дерево, включающее проверку всех атрибутов, содержит $\mathrm{card(X)} = \prod_{i=1}^{N} \mathrm{card}(X_i)$ листьев (card(X) – кардинальное число множества, или число элементов в нем), в каждом из которых может быть любое из d значений класса. Всего таких деревьев $d^{\mathrm{card(X)}}$. Общее же количество деревьев еще больше. Уже для сравнительно небольшого числа атрибутов полный перебор всех деревьев невозможен.

Существуют весьма простые с вычислительной точки зрения алгоритмы построения деревьев решений, дающие удовлетворительный результат. Сначала мы кратко опишем основную идею классического алгоритма *ID*3. Затем мы продемонстрируем, что этот алгоритм подпадает под общую схему минимизации длины описания посредством жадных алгоритмов, и приведем другую возможную модификацию этой схемы.

В алгоритме *ID*3 дерево решений строится рекурсивно, начиная с корня. Сначала выбирается некоторый атрибут. На основе значений атрибута множество обучающих примеров разделяется на непересекающиеся подмножества, в каждом из которых значение этого атрибута постоянно. Выбранный атрибут помещается в текущий узел дерева решений, где проверяется его значение, и в зависимости от этого значения происходит перемещение по одной из исходящих ветвей в следующий узел. Для каждого узла следующего уровня производится та же процедура, но уже для соответствующего подмножества примеров обучающей выборки. Проиллюстрируем это на примере.

Пусть имеется три атрибута X_1={0,1}, X_2={0,1}, X_3={0,1,2} и три класса a_1, a_2, a_3, и пусть дана обучающая выборка, состоящая из M=11 примеров:

x_1=(0,0,0), c_1=a_2 x_2=(1,0,0), c_2=a_3 x_3=(0,1,0), c_3=a_1
x_4=(1,1,0), c_4=a_1 x_5=(1,1,1), c_5=a_2 x_6=(1,0,1), c_6=a_2
x_7=(0,1,1), c_7=a_2 x_8=(0,0,2), c_8=a_1 x_9=(0,1,2), c_9=a_1
x_{10}=(1,0,2), c_{10}=a_3 x_{11}=(1,1,2), c_{11}=a_3

На рис. 48 представлен процесс построения дерева решений, при котором атрибуты тестируются в порядке возрастания их номеров, начиная с атрибута X_1. Для получения результирующего дерева на рис. 48 достаточно лишь исключить информацию о подмножествах, на которые разбиваются примеры обучающей выборки. Отметим, что подобное дерево решений может содержать узлы, которым не соответствует ни один пример обучающей выборки, а значит, отсутствует информация о том,

какие классы должны содержаться в листьях, являющихся потомками данного узла. Это приводит к необходимости введения пропусков как некоторого дополнительного класса «?».

Такое рекурсивное построение дерева решений – это просто вполне очевидный способ формирования *ad hoc* дерева. Однако в алгоритме *ID*3 есть принципиальный прием, которым мы пока не воспользовались, и который делает алгоритм интересным. Это выбор порядка проверки атрибутов по степени их информативности. Рассмотрим этот прием, для чего нужно сформулировать понятие информативности признака.

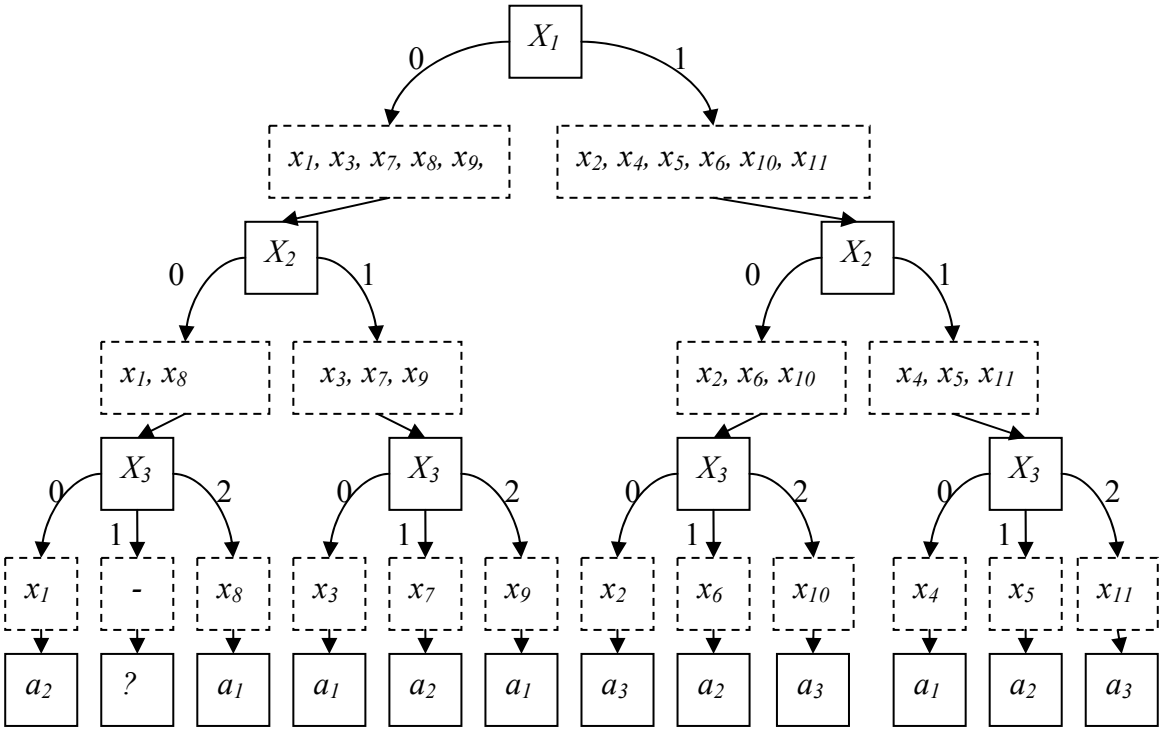


Рис. 48. Пояснение к рекурсивному построению дерева решений с указанием подмножеств, на которые разделяется обучающая выборка в результате проверки атрибутов. Порядок проверки атрибутов не изменяется

Исходно присутствует неопределенность в номере класса, к которому относится каждый объект. При выполнении классификации эта неопределенность устраняется за счет информации, содержащейся в атрибутах. Если для выполнения классификации требуется, скажем, один атрибут, принимающий с равной вероятностью одно из двух значений, то для выполнения классификации нужен один бит информации. Если же требуется три таких атрибута, то неопределенность в номере класса будет равна восьми битам.

Информативность одного атрибута можно определить как разницу в количестве информации, требуемой для выполнения классификации до использования этого атрибута и после его использования. Здесь полезно рассмотреть «беспорядочную» гипотезу, суть которой заключается в том, что ею допускается на выходе любой класс с некоторой вероятностью.

Пусть на основе обучающей выборки построено распределение вероятностей по классам $P(a)$. Тогда энтропия этого распределения, умноженная на размер обучающей выборки, будет определять длину описания «беспорядочной» гипотезы и будет ограничивать сверху количество информации, необходимой для завершения классификации (при описании алгоритма *ID*3 эта величина обычно называется количеством информации, необходимым для завершения построения дерева решений):

$$MH(\mathrm{A}) = -M\sum_{i=1}^{d} P(a_i)\log_2 P(a_i). \qquad (120)$$

После использования какого-либо атрибута выборка разделяется на подмножества, в каждом из которых имеется собственное распределение конечных классов. Наиболее предпочтителен тот атрибут, после проверки которого происходит наиболее четкое разделение гистограммы классов в подмножествах. Для индифферентного (по отношению к изучаемому концепту) атрибута гистограммы классов в подмножествах будут совпадать с гистограммой классов исходного множества, их энтропии будут равны, и для завершения классификации примеров обучающей выборки потребуется столько же информации, сколько требовалось до проверки атрибута. То есть информативность этого атрибута будет равна нулю. Получим конкретные математические выражения.

Пусть в зависимости от значения атрибута выборка из M элементов разделяется на K подвыборок, причем в j-й подвыборке M_j элементов, дающих распределение вероятностей по классам $P_j(a)$. Тогда информативность атрибута может быть оценена как

$$\begin{aligned} I &= MH(\mathrm{A}) - \sum_{j=1}^{K} M_j H_j(\mathrm{A}) = \\ &- M\sum_{i=1}^{d} P(a_i)\log_2 P(a_i) + \sum_{j=1}^{K} M_j \sum_{i=1}^{d} P_j(a_i)\log_2 P_j(a_i) \end{aligned} \qquad (121)$$

В нашем примере исходное распределение вероятностей классов было

$$P(a_1) = 4/11; P(a_2) = 4/11; P(a_3) = 3/11 \text{ и } M=11,$$

то есть исходная неопределенность номера классов для всех примеров составляет примерно 17,3 бита.

Атрибуты производят разделение на подмножества со следующими распределениями:

X_1: $P_1(a_1) = 3/5; P_1(a_2) = 2/5; P_1(a_3) = 0/5$ (M_1=5),
$P_2(a_1) = 1/6; P_2(a_2) = 2/6; P_2(a_3) = 3/6$ (M_2=6).
$M_1H_1(\mathrm{A}) + M_2H_2(\mathrm{A}) = 4.85 + 8.75 = 13.6$ бит.

X_2: $P_1(a_1) = 1/5; P_1(a_2) = 2/5; P_1(a_3) = 2/5$ (M_1=5),
$P_2(a_1) = 3/6; P_2(a_2) = 2/6; P_2(a_3) = 1/6$ (M_2=6).

$$M_1 H_1(\text{A}) + M_2 H_2(\text{A}) = 7.6 + 8.75 = 16.35 \text{ бит.}$$

X_3 : $P_1(a_1) = 2/4; P_1(a_2) = 1/4; P_1(a_3) = 1/4$ (M_1=4),

$P_2(a_1) = 0/3; P_2(a_2) = 3/3; P_2(a_3) = 0/3$ (M_2=3),

$P_3(a_1) = 2/4; P_3(a_2) = 0/4; P_3(a_3) = 2/4$ (M_3=4).

$$M_1 H_1(\text{A}) + M_2 H_2(\text{A}) + M_3 H_3(\text{A}) = 6 + 0 + 4 = 10 \text{ бит.}$$

Наибольшей информативностью обладает третий атрибут. Таким образом, он должен быть помещен в корень дерева. Выбор следующего атрибута должен осуществляться заново, причем отдельно для каждой ветви. Причина, по которой необходимо пересчитывать информативность атрибутов, вполне очевидна: проверенный атрибут может обладать разным количеством взаимной информации с различными атрибутами, то есть требуется не просто смотреть на количество информации, несомое атрибутами о классах, но количество новой информации.

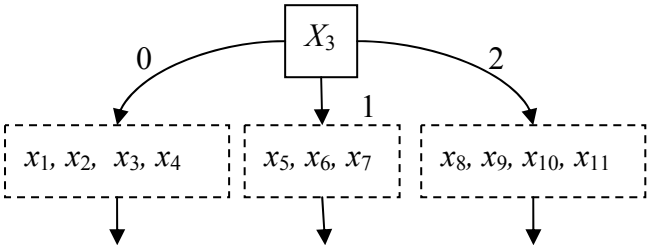


Рис. 49. Первый шаг выращивания дерева решений: помещение в корень проверки атрибута X_3, который разделяет обучающую выборку на три подмножества

На рис. 49 представлен первый шаг построения дерева решений, начиная с наиболее информативного атрибута X_3. Этот атрибут разбивает исходную выборку на три подмножества. Видно, что все примеры x_5, x_6, x_7 относятся к классу a_2, то есть дальнейшую проверку атрибутов в этой ветви вести не нужно. Несложно определить, что для левой ветви далее должен тестироваться атрибут X_2, а для правой – X_1. В частности, для правой ветви атрибут X_1 разбивает множество примеров на два подмножества с распределениями вероятностей по классам $P_1(a_1)$=1 и $P_2(a_2)$=1, то есть этот атрибут содержит информацию, достаточную для окончательной классификации примеров. Таким образом, на одном уровне дерева в разных его ветвях могут проверяться разные атрибуты.

Если завершить построение дерева решений этим алгоритмом, то получим дерево, представленное на рис. 50. Это дерево гораздо компактнее дерева, представленного на рис. 48, при построении которого порядок проверки атрибутов не выбирался. При этом, очевидно, произошло и обобщение. В частности, вторым деревом (рис. 50) пример (0, 0, 1) классифицируется как a_2, а первым деревом он считается пропуском. При малом размере выборки (или большом числе атрибутов) или зашумленных данных это обобщение было бы еще более принципиальным.

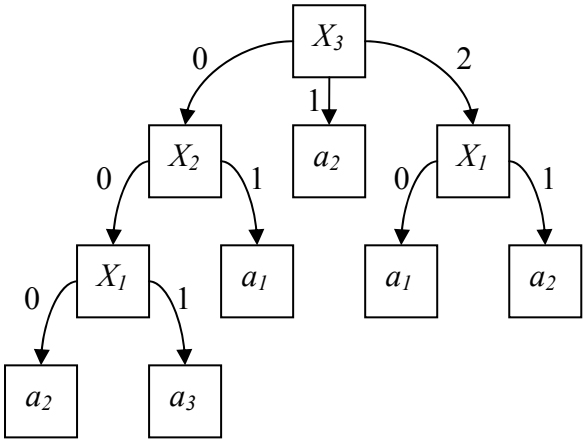


Рис. 50. Дерево решений, строящееся алгоритмом ID3 при выборе наиболее информативных признаков

Подобный выбор порядка атрибутов можно рассматривать как итеративное улучшение на основе критерия длины описания «беспорядочного» стохастического дерева решений посредством жадного алгоритма. Напомним, что длина описания дерева решений включает два слагаемых:

$$DL = DL_{tree} + DL_{exceptions}, \qquad (122)$$

где DL_{tree} – длина описания самого дерева решений, а $DL_{exceptions}$ – длина описания исключений. При этом длина описания исключений вычисляется по формуле

$$DL_{exceptions} = \sum_i \left(L(t_i) + m_i H(e_i)\right), \qquad (123)$$

где $L(t_i)$ – длина описания таблицы перекодировки для i-го узла; m_i – число примеров обучающей выборки, классификация которых заканчивается в i-м узле, а $H(e_i)$ – энтропия значений классов этих примеров.

Начальное «беспорядочное» стохастическое дерево решений с единственным узлом для нашего примера будет содержать единственный узел с уже приведенным распределением вероятностей $P(a)$. Это дерево представлено на рис. 51.

a_1 (P=4/11); a_2 (P=4/11); a_3 (P=3/11)

Рис. 51. «Беспорядочное» дерево решений, состоящее из единственного корневого узла, являющегося также и листом и содержащего в себе все конечные классы с указанными вероятностями

После замены этого узла узлом, в котором выполняется проверка атрибута X_3, получаем дерево решений, содержащее на 3 узла больше (что обычно обозначается как *выращивание* дерева). Вместо того чтобы считать объект, представленный на рис. 51, некоторой промежуточной конструкцией, его вполне можно трактовать как вполне корректное стохастическое дерево решений (рис. 52).

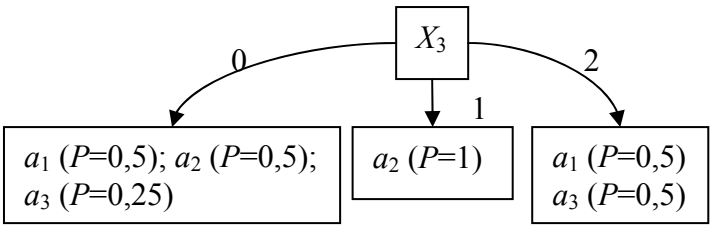


Рис. 52. Пример стохастического дерева решений, получившегося после первого шага процедуры выращивания

Тогда информативность признака, выражаемая формулой (121), представляет собой ни что иное, как изменение длины описания (122) в результате добавления в дерево решений проверки дополнительного атрибута. Алгоритм *ID*3 является «жадным», поскольку на каждой его итерации дерево выращивается так, чтобы локальное уменьшение длины описания было максимальным.

Отсюда также видно, что формула (121) не совсем точна, так как в ней учитывается только длина описания исключений (см. ур. (123)), при этом не учитывается изменение длины описания самого дерева решений. Помещение в текущий лист нового атрибута вызывает изменение размера дерева на K узлов. Для разных атрибутов эта величина может быть разной.

Несложно представить такую ситуацию, когда значение K столь велико, что разбивает обучающую выборку на подмножества, в каждом из которых содержится лишь по одному примеру. По формуле (121) такой атрибут будет идеальным. В действительности же, он ведет к построению *ad hoc* гипотезы и не несет информации. Если воспользоваться подсчетом изменения полной длины описания, введенной в предыдущем разделе, то окажется, что такой атрибут ничуть не лучше атрибута, принимающего лишь одно значение K=1. В процессе выращивания дерева решений такие атрибуты никогда не должны выбираться при наличии других атрибутов. Таким образом, мы видим, что эвристически введенная информативность атрибутов не является вполне адекватной, в то время как обращение к принципу МДО предохраняет от ошибок.

Таким образом, алгоритм *ID*3 вполне подпадает под общую схему итеративной минимизации длины описания. В наиболее общем случае в рамках этой схемы выполняются следующие шаги:

- конструируется начальное дерево решений («беспорядочное» или *ad hoc*);
- среди множества допустимых операций по преобразованию дерева решений выбирается та, которая позволяет в наибольшей степени уменьшить суммарную длину описания (122);
- выбор операции повторяется до тех пор, пока есть хоть одна операция, приводящая к уменьшению длины описания.

В алгоритме ID3, как уже отмечалось, итеративное улучшение начинается с «беспорядочного» дерева, а число операций преобразования весьма ограничено (есть лишь операция по замене листа узлом с проверкой одного из атрибутов).

Мы увидели, что выбор порядка проверки атрибутов в алгоритме *ID*3 ведет к уменьшению суммарной длины описания. Однако это делается посредством «жадного» алгоритма поиска, которым на некотором шаге может быть принято неоптимальное (в глобальном плане) решение. Принятые решения при дальнейшем выращивании дерева не пересматривается. Реализовать такой пересмотр можно, организовав перебор на глубину, большую единицы. Для этого можно использовать классические методы эвристического программирования.

Алгоритм построения дерева также может начинаться не с «беспорядочного», а с *ad hoc* дерева (например, подобного тому, какое представлено на рис. 48). Это дерево классифицирует все примеры обучающей выборки правильно, а все примеры, не вошедшие в обучающую выборку, как пропуски. Далее осуществляется последовательное упрощение дерева на основе каких-либо операций по преобразованию дерева. При этом, как и раньше, на каждом шаге выбирается та операция, которая приводит к максимальному уменьшению длины описания. Обычно рассматривается только один тип операций: удаление какого-то узла дерева со всеми его потомками. Как правило, этот тип операций называется *отсечением* (pruning).

При отсечении порядок проверки атрибутов не меняется, что делает отсечение довольно слабым средством по преобразованию деревьев решений, которое не может применяться самостоятельно, а лишь улучшает дерево, построенное некоторым другим способом. Например, отсечение дерева может осуществляться после его выращивания. Более сложные операции по преобразованию деревьев (перестановка родительского и дочернего узлов, исключение некоторого узла без удаления его потомков и т.д.) исследованы меньше. В частности, указанные операции имеют ясный смысл только тогда, когда во всех дочерних узлах модифицируемого узла тестируется один и тот же атрибут.

Таким образом, отсечение и выращивание деревьев решений укладываются в общую схему итеративной оптимизации длины описания. Операции по добавлению узлов и их удалению могут использоваться совместно, а также могут быть дополнены некоторыми другими операциями преобразования деревьев. Сам же оптимизационный алгоритм может быть абстрагирован от того, что именно подвержено оптимизации, коль скоро определена целевая функция и операции преобразования. Это полностью согласуется с методами эвристического программирования, поэтому для построения деревьев решений могут использоваться не только жадный алгоритм и нерассмотренные здесь генетические алгоритмы, но и другие методы поиска.

Вопросы и упражнения

1. В чем отличие задач восстановления деревьев решений и наборов правил?

2. Как в алгоритме *ID*3 происходит выбор порядка проверки атрибутов, и какое влияние имеет порядок выбора признаков на обобщающую способность строящихся при этом деревьев решений?
3. Какой тип поиска используется в алгоритме *ID*3?
4. Как можно обобщить операции по выращиванию дерева решений, используемые в алгоритме *ID*3?
5. Какие преимущества имеет принцип минимальной длины описания по сравнению с критерием сортировки атрибутов, применяемых в алгоритме *ID*3? В каких случаях этот критерий может давать неадекватный результат?
6. В чем отличие построения деревьев решений выращиванием и отсечением?
7. Что представляют собой *ad hoc* и беспорядочное деревья решений? С какого из них начинается построение дерева решений выращиванием и отсечением?

СПИСОК РЕКОМЕНДОВАННОЙ ЛИТЕРАТУРЫ

Основная литература по дисциплине

1. **Люгер, Д.Ф. Искусственный интеллект: стратегии и методы решения сложных проблем** / Д.Ф. Люгер. – 4-е изд.: Пер. с англ. – М.: Изд. дом “Вильямс”, 2003. – 865 с.
2. **Рассел, С. Искусственный интеллект: современный подход (AIMA)** / С. Рассел, П. Норвиг – 2-е изд.: Пер. с англ. – М.: Изд. дом “Вильямс”, 2005. – 1424 с.
3. **Хант, Э. Искусственный интеллект** / Э. Хант. – М.: Мир, 1978. – 558 с.
4. **Девятков, В.В. Системы искусственного интеллекта: учеб. пособие для вузов** / В.В. Девятков. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2001. – 352 с.
5. **Гаврилов, А.В. Системы искусственного интеллекта: учеб. пособие: в 2-х ч.** / А.В. Гаврилов. – Новосибирск: Изд-во НГТУ, 2001.
6. **Смолин, Д.В. Введение в искусственный интеллект: конспект лекций** / Д.В. Смолин – М.: ФИЗМАТЛИТ, 2004. – 208 с.
7. **Потапов, А.С. Распознавание образов и машинное восприятие: общий подход на основе принципа минимальной длины описания** / А.С. Потапов. – СПб.: Политехника, 2007. – 548 с.
8. **Назаров, А.В. Нейросетевые алгоритмы прогнозирования и оптимизации систем** / А.В. Назаров, А.И. Лоскутов. – СПб.: Наука и Техника, 2003. – 384 с.
9. **Ту, Дж. Принципы распознавания образов** / Дж. Ту, Р. Гонсалес – М.: Мир, 1978. – 412 с.
10. **Джексон, П. Введение в экспертные системы: учеб. пособие** / П. Джексон. – Пер. с англ. – М.: Изд. дом “Вильямс”, 2001. – 624 с.

11. **Слэйгл, Дж. Искусственный интеллект: подход на основе эвристического программирования** / Дж. Слэйгл. – Пер. с англ. – М.: Мир, 1973. – 319 с.

Дополнительная литература

12. **Турчин, В.Ф. Феномен науки: кибернетический подход к эволюции** / В.Ф. Турчин. – 2-е изд. – М.: ЭТС, 2000. – 368 с.
13. **Гаазе-Рапопорт, М.Г. От амебы до робота: модели поведения** / М.Г. Гаазе-Рапопорт, Д.А. Поспелов – М.: Наука, 1987. – 288 с.
14. **Фу, К. Структурные методы в распознавании образов** / К. Фу. – Пер. с англ. – М.: Мир, 1977. – 320 с.
15. **Горелик, А.Л. Методы распознавания** / А.Л. Горелик, В.А. Скрепкин – М.: Высш. школа, 1977. – 222 с.
16. **Патрик, Э. Основы теории распознавания образов** / Э. Патрик. – Пер. с англ. / Под ред. Б.Р. Левина. – М.: Сов. радио, 1980. – 408 с.
17. **Минский, М. Фреймы для представления знаний** / М. Минский. – Пер. с англ. – М.: Энергия, 1979. – 151 с.
18. **Будущее искусственного интеллекта** / Редакторы-составители: К.Е. Левитин, Д.А. Поспелов. – М.: Наука, 1991. – 302 с.
19. **Искусственный интеллект. Справочник в 3-х томах** / М.: Радио и связь, 1990.
20. **Павилёнис Р.И. Проблема смысла: современный логико-философский анализ языка** / Р.И. Павилёнис – М.: Мысль, 1983. – 286 с.

ОГЛАВЛЕНИЕ