Лекция 12. Введення до ймовірнісного програмування.

Что такое вероятностное программирование?

Вероятностное программирование - это способ создания систем, помогающих принимать реше1шя в условиях неопределенности. Многие решения, принимае мые нами каждодневно, подразумевают учет релевантных факторов, которые мы не наблюдаем непосредственно. Исторически одним из способов принять реше ние в таких усло виях неопределенности стали системы вероятностного рассужде- ния. *Вероятностное рассуждение* позволяет объединить наши знат-гия о ситуации с вероятностными законами и таки м образом учесть не наблюдаемые факторы , важные для принятия решения. До недавнего времени область применения си стем вероятностного рассуждения была ограничена, и ко многим возникающим на практике задачам их удавалось применить с большим трудом. Вероятностное программирование - это новый подход, позволивший упростить построеиие си стем вероятностного рассуждения и расширить их применимость.
Чтобы понять, в чем смысл вероятностного программирования, начнем с рас смотрения того, как при н имается решение в условиях нео пределеююсти и какую роль в этом и грают субъективные суждения. Затем посмотрим, как может помочь система вероятностного рассуждения. Мы познакомимся с тремя видами рассуж дений, присущих таким системам. После этого станет понятно, что такое вероят ностное программирование и как оно позволяет использовать всю мощь языков программирования для построения систем вероятностного рассуждения.

Как мы высказываем субъективное суждение?

В реальном мире интересующие нас вопросы редко допускают недвусмысленный ответ: да или нет. Например, при выводе н а ры нок нового продукта хотелось бы знать, хорошо л и он будет продаваться. Вы полагаете, что его ждет успех, потому что продукт хорошо спроектирован и изучение рынка показало , что в 1-гем есть потребность, но полной уверенности нет. Быть может, ваш конкурент представит лучший продукт, а, быть может, в вашем продукте обнаружится фатальный изъян, из-за которого рынок от н его отвер н ется. Или эко н омическая ситуация повер н ет ся к худшему. Если вы хотите стопроцентн ой уверенности , то никогда не сможете принять решение о начале продаж (см. Рис. 1.1).
Для принятия решений такого рода помогает я зык вероятностей. Запуская про дукт, мы можем воспользоваться прошлым о пытом продажи похожих продуктов для о це н ки вероятн ости того, ч то этот окажется ус пешным . А зная эту вероятность, мы сможем решить, выводить продукт на рынок или нет. Возможно , нас интересу ет не тол ько са м факт успеш н ости продукта, н о и размер ожидаемого дохода или, напротив, размер убытков, которые мы понесем в случае неудачи. Знание веро ятностей различных исходов помогает принимать более обоснованные решения.
Ну хорошо, вероятноспгое м ышление может помочь в принятии трудных ре шений с элементами субъекти вности . Но как это выглядит на практ ике? Общий принцип выражен в виде следующего факта.

Факт. Субъективное суждение основано на знании+ логике.

Вы располагаете некоторыми знаниями об интересующей вас проблеме. Напри мер, вы много знаете о своем продукте и, возможного, провели исследование рын ка, чтобы понять, кто будет его покупать. Возможн о, вы также кое-что знаете о конкурентах и знакомы с прогнозом развития экономики. Логика же позволяет

получ ать ответ ы на вопросы с использо ванием имеющихся зна ни й.

Рис. 1. 1. В прошлом году мой продукт всем нравился , но как будет в следующем?

Нам необходим способ описания своих зна ний и логи ка получения ответов н а вопросы с ис п ользованием этих знан и й. Вероятностное п рограммирование даст то и другое.

Системы вероятностных рассуждений помогают принимать решения

Вероятностные рассуждения - это подход, в котором модель предметной обла сти используется для принятия решени й в условиях неопределенности. Возьмем пример из области футбола. Предположим , что по статистике 9 % угловых завер шаются голом. Требуется предсказать исход конкретного углового удара. Рост центрального напада ющего атакующей коман ды равен 193 см, и известно, что он отлич но играет головой. Основного вратаря защищающейся команд ы только что унесли на носилках, и на замену ем у вы ш ел вратарь, который проводит свой п ер- вый матч. Кроме того, дует сильный ветер, который затрудняет контроль над по летом мяча. И как при таких услов ия х вычислить вероятность?

Рис .1.2.Как система вероятн остн ых рассуждений предсказывает ИСХОД углово го удара

На рис. 1.2 показано, как получать ответ с помощью системы вероятностиых рассуждений. Все с во и з н ания об углов ых ударах и релевантных факторах мы кодируем в виде модели углового. Затем мы предоставляем факты о конкретном угловом удар е: что цен траль ны й нападающий высоки й , что вратарь н еопы тный и что дует сильный ветер. Мы говорим системе, что хотели бы узнать, будет ли забит гол. Алгоритм вывода возвращает ответ: гол будет забит с вероятностью 20 %.

Основные определения

Общие знания – что известно о предметной области в общем, без деталей, описывающих конкретную ситуацию.

Вероятностная модель – представление общих знаний о предметной области в количественном виде, в терминах вероятностей.

Факты – имеющаяся информация о конкретной ситуации.

Запрос – свойство ситуации, о котором мы хотели бы узнать.

Вывод – процесс использования вероятностной модели для получения ответа на запрос при известных фактах.

В си стеме вероятностных рассужде н ий мы создаем модель, в которой отражены все релевантные общие знания о предметной области в виде вероятностей. В на шем примере это может быть о писание ситуации у глового удара и всех релевантны х характеристик игроков и окружающих условий, которые могут повлиять н а исход. Затем модель п риме няется к конкретной ситуации, для которой мы хотим получить заключе11ие. Для этого мы передаем модели имеющуюся информацию, которая н аз ывается фактами. В да нн ом случ ае есть три факта: це нтральн ы й на падающий высокий, вратарь неопытный и дует сильный ветер. Полу ченное от мо дели заключе~1ие поможет принять р е шен ие

-

наприм ер, что на следую щу ю игру
нужно поставить другого вратаря. Сами заключения пр едстают в виде вероятно стей, 1-гапример , вероятностей других игровых навыков вратаря.
Соот ношение между моделью, предоставляемой вами информацией и ответа ми на за просы математически стро го о п ределе но з аконами теор и и вероятностей. Процесс использования модели для получения ответов н а запросы при известных фактах называется вероятностиым выводом, или просто выводом. По счастью, раз работаны алгоритмы , которые вы п олняют н еобходим ые вычисления, скрывая от

вас всю математику. Они называются алгоритмами вывода.
На рис. 1.3 схем ати чес ки изображены описан ные вы ше понятия.

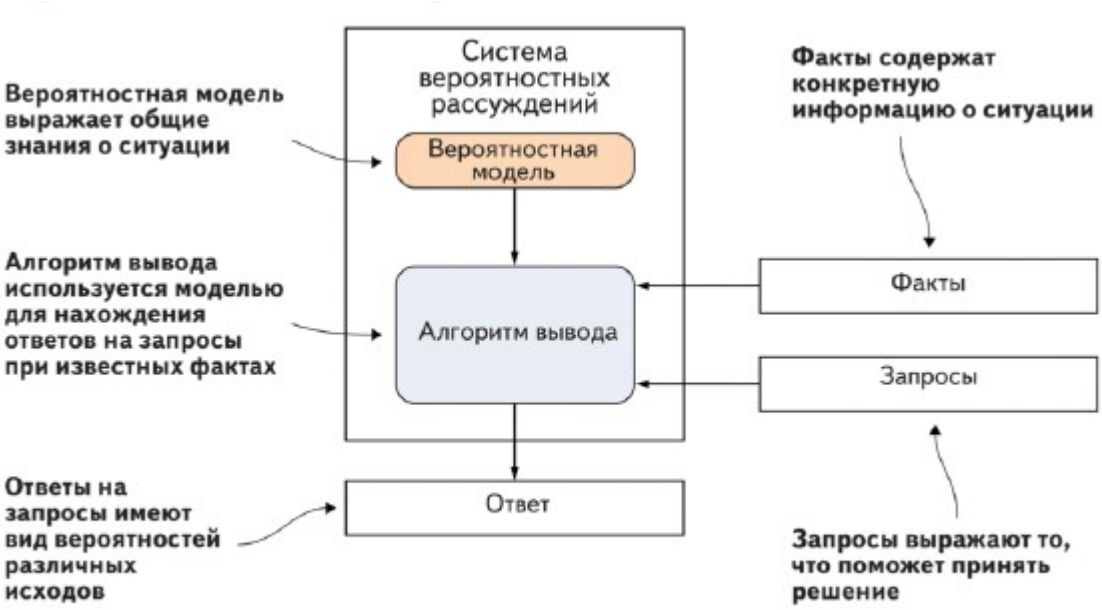

Рис. 1.3. Основные компоненты системы вероятностных рассуждений

Итак, мы вкратце описали основн ы е ком поненты систем ы вероятностных рас суждеиий и способ взаимодействия с ней. Но что можно делат ь с такой системой? Как она помогает принимат ь решен ия? В следующем разделе опи сан ы три вида рассуждений , котор ы е может вып олнять такая система.

Системы вероятностных рассуждений обладают гибкостью. Они могут отвечать на запросы о любом аспекте ситуации, если задаиы факты, отиосящиеся к любым други м ас пектам. На практике система вы полняет рассужде н ия трех видов.

- *Предсказание будущих событий.* Мы уже видели это на рис. 1.2, где система предсказывает, будет ли в данной ситуации забит гол. Факты обычно включают информацию о текущей ситуации, например рост центрального нападающего, опытность вратаря и силу ветра.
- *Вывод причины событий.* Прокрутим пленку вперед на 10 секунд. Высокий центральный нападающий только что забил гол головой, протолкнув мяч под корпусом вратаря. Что можно сказать об этом вратаре-новобранце при таких фактах? Можно ли заключить, что ему не хватает умения? На рис. 1.4 показано, как использовать систему вероятностных рассуждений для ответа на этот вопрос. Берется та же модель углового удара, что и для прогноза гола. (Это полезное свойство вероятностных рассуждений: одну и ту же модель можно использовать как для предсказания будущего результата, так и для объяснения причин, приведших к известному результату.) Факты те же, что и прежде, плюс тот факт, что гол забит. Запрос касается квалификации вратаря, а ответ дает вероятности различных уровней квалификации.

Рис. 1.4. Если изменить запрос и факты, то система сможет сделать вывод о том, почему был забит гол

Немного поразмыслив, вы поймете, что первый способ - рассуждения с прямым ходом времени, т. е. предсказание будущих событий на ос1юве того, что известно о текущей ситуации, а второй - рассужден ия с обратным ходом времени, когда требуется вывести прошлые условия из известных результа тов. Типичная вероятностная модель следует естественному ходу времени.

Игрок подает угловой, затем ветер воздействует на летящий мяч, затем цен-тральн ый на пада ющий прыгает, п ытая с ь достать мя ч головой, и , наконец, вратарь пытается спасти ворота. Но процесс рассуждения может происхо дить как в прямом , так и в обратном порядке. Это клю 0гевая особенн ость ве роятностного рассуждения, которую я не раз буду подчеркивать: направле ииe рассуждения иеобязательио совпадает с иаправлеиием модели.

Обучение на прошлых событиях, чтобы лучше предсказывать будущие. Прокрутим пленку еще на 10 минут вперед. Та же команда заработала еще один угловой. Все так же, как и раньше – высокий центральный нападающий, неопытный вратарь – только ветер теперь стих. Вероятностное рассуждение позволяет использовать информацию о том, что случилось при подаче предыдущего углового, чтобы лучше предсказать исход следующего. На рис. 1.5 показано, как это делается. В состав фактов входит все то, что и в прошлый раз (с пометкой, что это было в прошлый раз), а также новая информация о текущей ситуации. Отвечая на вопрос, будет ли забит гол на этот раз, алгоритм вывода сначала определяет свойства ситуации, которые привели к голу в первый раз, например, квалификацию нападающего и вратаря. А затем эти свойства используются для предсказания исхода в новой ситуации.

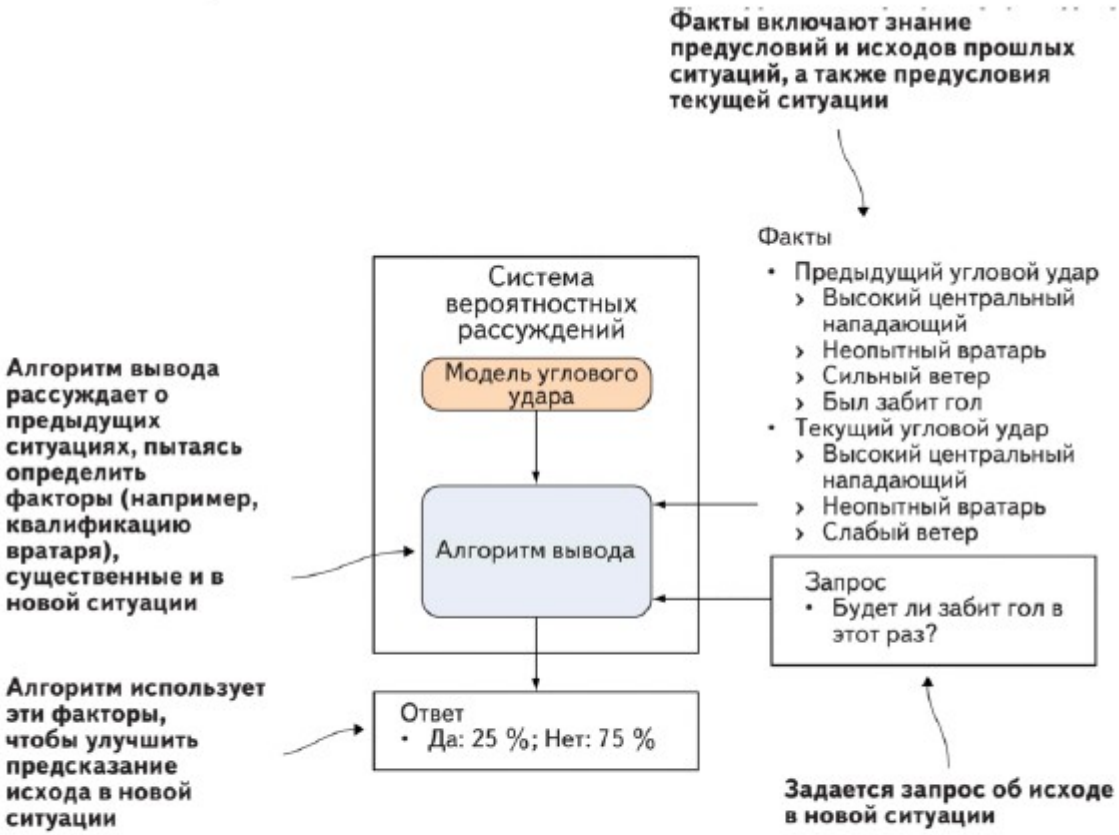

Рис. 1.5. Принимая во внимание факты, относящиеся к результату прошлого углового, система вероятностного рассуждения может лучше предсказать результат следующего

Запросы перечисленных видов могут помочь в принятии различных решений.

- Зная вероятность гола при наличии или отсутствии дополнительного защитника, можно решить, не стоит ли заменить атакующего защитником.
- На основе оценке квалификации вратаря можно решить, какую зарплату предложить ему в следующем контракте.
- Получив информацию о вратаре, можно решить, стоит ли ставить его на следующую игру.

Обучение улучшенной модели

Выше были описаны способы рассуждения о конкретных ситуациях при известных фактах. Но система вероятностных рассуждений позволяет сделать еще одну вещь: обучаясь на прошлом опыте, улучшить общие знания. Третий способ рассуждений – это использование прошлых результатов для лучшего предсказания исхода в конкретной новой ситуации. А сейчас мы говорим об улучшении самой модели. Если имеется обширный прошлый опыт, т. е. информация о многих угловых ударах, то почему бы не обучить новую модель, представляющую общие знания о том, что обычно происходит при подаче углового? На рис. 1.6 показано, что это достигается с помощью алгоритма обучения. В отличие от алгоритма вывода, его задача состоит не в том, чтобы отвечать на вопросы, а в том, чтобы породить новую модель. На вход алгоритма обучения подается исходная модель, а он обновляет ее на основе опыта. Затем новую модель можно использовать для ответа на будущие запросы. Предположительно ответы новой модели будут более обоснованы, чем ответы исходной.

Системы вероятностных рассуждений и точные предсказания

Как и в любой системе машинного обучения, чем больше данных у системы вероятностных рассуждений, тем точнее ее ответы. Качество предсказания зависит от двух факторов: верности отражения реального мира в исходной модели и объема предоставленных данных. Вообще говоря, чем больше данных, тем менее существенно качество исходной модели. Причина в том, что новая модель – комбинация исходной и информации, содержащейся в данных. Если данных очень мало, то доминирует исходная модель, поэтому чем она точнее, тем лучше. Если же данных много, то доминируют именно они, так что в новой модели остается очень мало от исходной, поэтому ее точность не слишком важна. Например, если мы обучаем модель по данным за весь футбольный сезон, то факторы, от которых зависит результативность углового, будут учтены весьма точно. Если же имеются данные только об одном матче, то придется изначально задать хорошие значения факторов, иначе на точность предсказаний можно не рассчитывать. Системы вероятностного программирования умеют пользоваться моделью и имеющимися данными так, что предсказания получаются максимально точными.

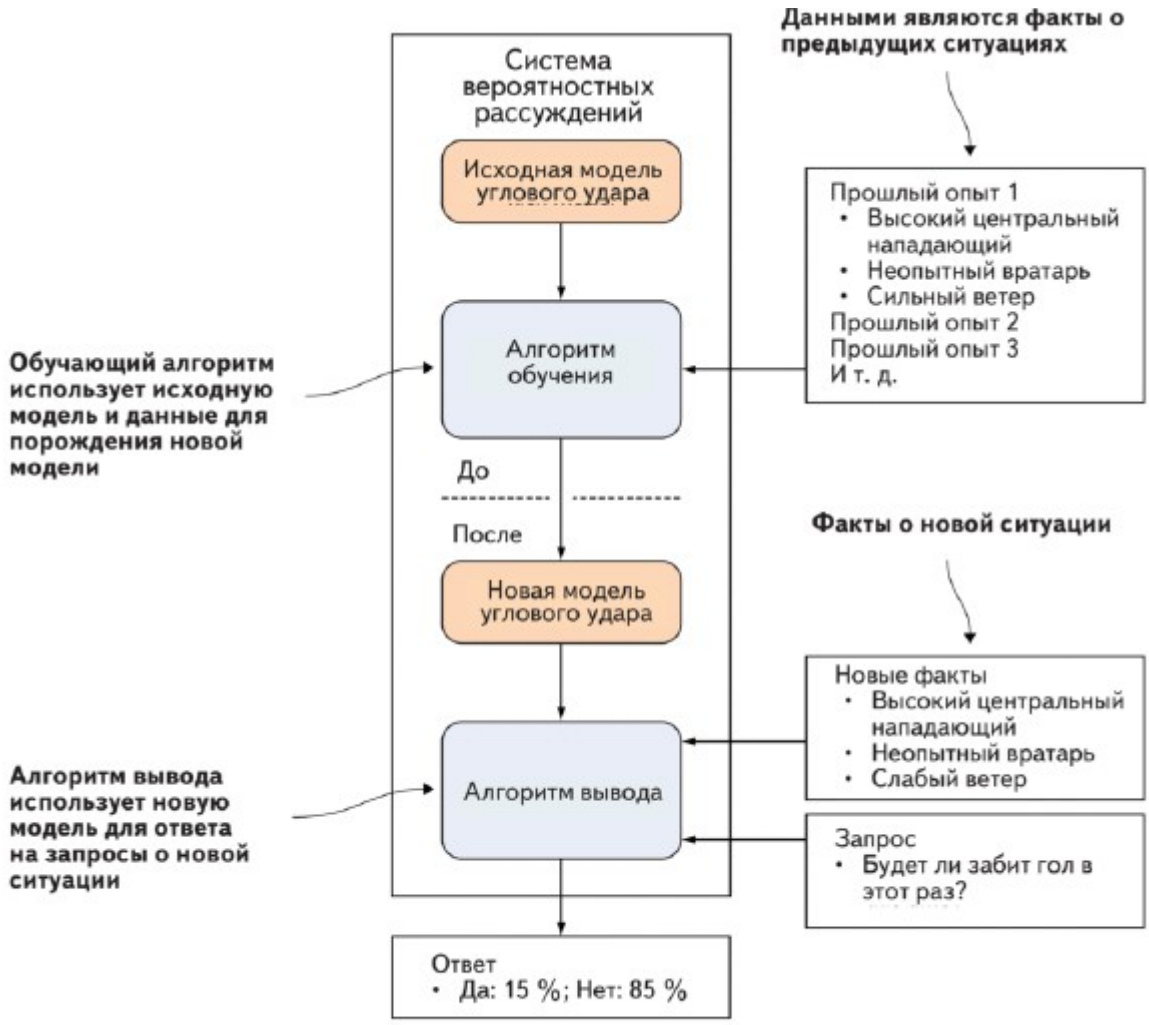

Рис. 1.6. С помощью алгоритма обучения можно обучить новую модель на основе прошлого опыта, а затем использовать ее для будущих выводов

Теперь вы знаете, что такое вероятностно е рассуждение. Ну а что же тогда такое вероятностное программирование?

представимых языком, называется *выразительной силой* языка. Для разработки приложений требуется, чтобы выразительная сила была как можно больше.

Система *вероятностного программирования* – это просто система вероятностных рассуждений, для которой языком представления является язык программирования. Говоря *«язык программирования»*, я имею в виду, что он обладает всеми возможностями, которые принято ожидать от типичного языка программирования: переменными, развитой системой типов данных, средствами управления потоком выполнения, функциями и т. д. Как вы вскоре увидите, языки вероятностного программирования способны выразить широкий спектр вероятностных моделей, далеко выходящий за рамки большинства традиционных систем вероятностных рассуждений. Выразительная сила языков вероятностного программирования очень высока.

На рис. 1.7 иллюстрируется соотношение между системами вероятностного программирования и вероятностных рассуждений в общем случае. Сравните этот рисунок с рис. 1.3, чтобы стали яснее различия между двумя системами. Основное изменение состоит в том, что модели выражаются в виде программ на некотором языке программирования, а не в виде математических конструкций типа байесовской сети. Поэтому факты, запросы и ответы становятся переменными программы. Факты можно представить конкретными значениями переменных, запрос – это получение значения переменной, а ответ – вероятности того, что переменные принимают те или иные значения. Кроме того, система вероятностного программирования обычно включает набор алгоритмов вывода. Эти алгоритмы применяются к моделям, написанным на языке системы.

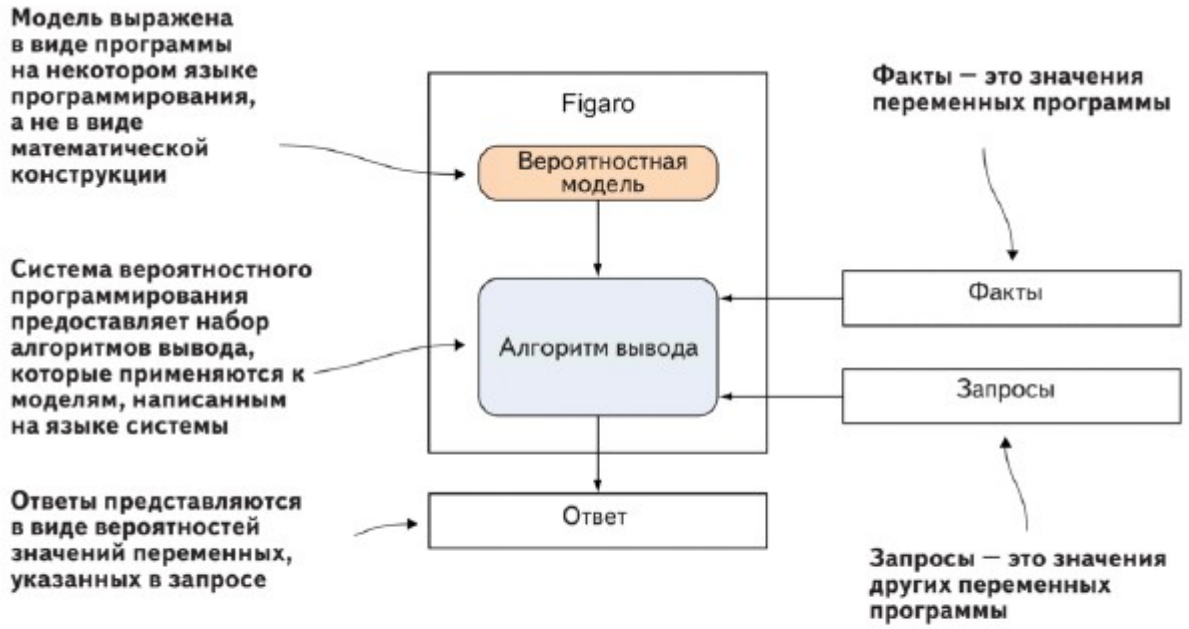

Рис. 1.7. Система вероятностного программирования – это система вероятностных рассуждений, в которой для представления вероятностных моделей применяется язык программирования

Существует много систем вероятностного программирования (см. обзор в приложении B), но в этой книге рассматриваются только функциональные, полные по Тьюрингу системы. *Функциональные* означает, что они основаны на функциональном языке программирования, но пусть это вас не пугает – чтобы пользоваться функциональной системой вероятностного программирования, не нужно знать, что такое лямбда-функция. Просто функциональное программирование составляет теоретическое основание, благодаря которому язык имеет возможность представлять вероятностные модели. А *полным по Тьюрингу* называется язык, на котором можно закодировать любое вычисление, которое способен выполнить цифровой компьютер. Если нечто вообще можно сделать с помощью компьютера, то это можно сделать на любом полном по Тьюрингу языке. Большинство языков программирования, с которыми вы знакомы, например C, Java и Python, являются полными по Тьюрингу. Поскольку системы вероятностного программирования основаны на языках программирования, полных по Тьюрингу, то они обладают выдающейся гибкостью в части типов моделей, которые можно построить с их помощью.

Основные определения

Язык представления – язык для кодирования знаний о предметной области в модели.

Выразительная сила – способность языка представления кодировать различные виды знаний.

Полный по Тьюрингу – язык, позволяющий выразить любое вычисление, которое можно выполнить на цифровом компьютере.

Язык вероятностного программирования – язык представления системы вероятностных рассуждений, в котором для представления знаний используется полный по Тьюрингу язык программирования.

В приложении B приведен обзор некоторых систем вероятностного программирования, помимо используемой в этой книге системы Figaro. В большинстве из них используются полные по Тьюрингу языки. В некоторых, например BUGS и Dimple, это не так, но они все равно полезны для тех приложений, на которые рассчитаны. В этой книге мы будем заниматься исключительно возможностями полных по Тьюрингу языков вероятностного программирования.

Представление вероятностных моделей в виде программ

Но каким образом язык программирования становится языком вероятностного моделирования? Как представить вероятностную модель в виде программы? Сейчас я дам очень краткий ответ на этот вопрос, а более глубокое обсуждение отложу на потом, когда мы станем лучше понимать, как выглядит вероятностная программа.

Главная идея любого языка программирования – *выполнение*. Мы выполняем программу, порождающую некоторый выход. Вероятностная программа в этом смысле аналогична, только она может иметь не один, а несколько путей выполнения, каждый из которых порождает свой выход. По какому пути следовать, определяется случайным выбором, совершаемым внутри программы. Каждый случайный выбор

имеет несколько возможных исходов, и в программе закодирована вероятность каждого исхода. Поэтому можно считать, что вероятностная программа – это программа, при выполнении которой случайным образом генерируются выходные данные.

Эта идея иллюстрируется на рис. 1.8. Здесь система вероятностного программирования содержит программу углового удара, которая описывает случайный процесс генерации исхода углового. Программа принимает ряд входных данных; в нашем примере это рост центрального нападающего, опытность вратаря и сила ветра. На основе этих данных программа случайным образом выполняется и генерирует выходные данные. Каждое случайное выполнение дает на выходе конкретный результат. Поскольку каждый случайный выбор может иметь несколько исходов, то существует много возможных путей выполнения, дающих различные результаты. Любой заданный результат, например взятие ворот, может быть сгенерирован на различных путях выполнения.

Рис. 1.8. Вероятностная программа определяет процесс случайной генерации выходных данных по известным входным

Рассмотрим, как эта программа определяет вероятностную модель. Любое конкретное выполнение программы является результатом последовательности случайных выборов. Каждый случайный выбор происходит с некоторой вероятностью. Если перемножить все эти вероятности, то получится вероятность пути выполнения. Таким образом, программа определяет вероятность каждого пути выполнения. Если представить себе, что программа прогоняется многократно, то доля прогонов, на которых генерируется заданный путь выполнения, равна его вероятности. Вероятность некоторого результата равна доле прогонов, на которых

был получен этот результат. На рис. 1.8 гол получается в 1/4 всех прогонов, поэтому вероятность гола равна 1/4.

Примечание. Возможно, вы недоумеваете, почему один из блоков на рис. 1.8 назван «Случайное выполнение», а не «Алгоритм вывода», как на других рисунках. На рис. 1.8 показан смысл вероятностной программы – определение процесса случайного выполнения, а не как использовать систему вероятностного программирования – выполнить алгоритм вывода для получения ответа на запрос при заданных фактах. Поэтому хотя структурно рисунки похожи, они иллюстрируют разные концепции. На самом деле, случайное выполнение лежит в основе и некоторых алгоритмов вывода, но есть много алгоритмов, которые не основаны на простом случайном выполнении.

Принятие решений с помощью вероятностного программирования

Легко видеть, как можно использовать вероятностную программу для предсказания будущего. Нужно просто выполнить ее случайное число раз, подавая на вход известные данные о настоящем, и посмотреть, сколько раз порождается каждый результат. В примере с угловым ударом на рис. 1.8 мы выполняли программу много раз, подавая на вход такие данные: высокий центральный нападающий, неопытный вратарь и сильный ветер. Поскольку в 1/4 прогонов получился гол, можно сказать, что при этих входных данных вероятность гола равна 25 %.

Однако прелесть вероятностного программирования заключается в том, что его можно с тем же успехом использовать для любых видов вероятностных рассуждений, описанных в разделе 1.3.1. Оно позволяет не только предсказывать будущее, но и выводить факты, приведшие к наблюдаемым результатам; можно «открутить» программу назад и посмотреть, какие причины породили данный исход.

Как все это работает? Вероятностное программирование стало применяться на практике, когда было осознано, что алгоритмы вывода, работающие для более простых языков представления, например байесовских сетей, можно обобщить и на программы. В части 3 описаны разнообразные алгоритмы вывода, благодаря которым это возможно. К счастью, в состав вероятностных систем программирования входит ряд встроенных алгоритмов вывода, которые применяются к программам автоматически. Вам нужно лишь предоставить знания о предметной области в виде вероятностной программы и описать факты, а система сама позаботится о выводе и обучении.

1.2. Зачем нужно вероятностное программирование?

Вероятностные рассуждения – одна из фундаментальных технологий машинного обучения. Такие системы используются компаниями Google, Amazon и Microsoft для извлечения смысла из имеющихся данных. Вероятностные рассуждения применялись в таких разных приложениях, как прогнозирование цены акций, рекомендование фильмов, диагностика компьютеров и обнаружение вторжений в вычислительные системы. Во многих из этих приложений используются методы, описанные в этой книге.

В предыдущем разделе следует выделить два важнейших положения:

- вероятностные рассуждения можно использовать для предсказания будущего, объяснения прошлого и обучения на прошлом опыте для лучшего предсказания будущего;
- вероятностное программирование – это вероятностное рассуждение, в котором для представления используется полный по Тьюрингу язык программирования.

В сочетании эти два положения позволяют отчеканить следующую формулу.

Факт. Вероятностное рассуждение + полнота по Тьюрингу = вероятностное программирование.

Обоснованием вероятностного программирования является тот факт, что из двух концепций, которые сами по себе являются достаточно мощными, составляется новая. В результате получается более простой и гибкий способ применения компьютеров для принятия решений в условиях неопределенности.

1.2.1. Улучшенные вероятностные рассуждения

У большинства современных вероятностных языков представления имеются ограничения на сложность представляемых систем. В таких относительно простых языках, как байесовские сети, предполагается фиксированный набор переменных, поэтому они не обладают достаточной гибкостью для моделирования предметных областей, в которых состав переменных может изменяться. В последние годы были разработаны более гибкие языки. В некоторых (например, BUGS) даже имеются возможности, характерные для языков программирования, напри-

мер итерация и массивы, хотя полными по Тьюрингу они все же не являются. Успех языков, подобных BUGS, ясно говорит о необходимости более развитых и структурированных представлений. Но движение в сторону полноценных полных по Тьюрингу языков открывает целый мир новых возможностей для вероятностных рассуждений. Теперь стало возможно моделировать длительные процессы с большим числом событий и взаимодействующих сущностей.

Снова рассмотрим пример на футбольную тематику, но теперь представьте, что вы занимаетесь спортивной аналитикой и хотите порекомендовать решения по подбору игроков в команду. Можно было бы воспользоваться для этой цели накопленной статистикой, но статистика не улавливает контекст, в котором была собрана. Можно провести более тонкий контекстный анализ, построив детальную модель футбольного сезона. Для этого придется моделировать много взаимосвязанных событий и взаимодействие игроков и команд. Трудно представить себе, как построить такую модель без структур данных и управляющих конструкций, имеющихся в полноценном языке программирования.

Вернемся к вопросу о выводе продукта на рынок и взглянем на процесс принятия решений по развитию бизнеса во всей его полноте. Запуск продукта – не изолированное событие, ему предшествуют этапы анализа рынка, исследований и разработок, и исход каждого этапа не предрешен. Результаты запуска продукта зависят от всех предшествующих этапов, а также от анализа того, что еще имеется на рынке. Полный анализ должен также учитывать, как отреагируют на наш продукт конкуренты и какие новые продукты они смогут предложить. Это трудная задача, т. к. приходится выдвигать гипотезы о конкурирующих продуктах. Возможно даже, что существуют конкуренты, о которых вы еще не знаете. В этом примере продукты оказываются структурами данных, порождаемыми сложными процессами. Поэтому наличие полноценного языка программирования было бы крайне полезно для создания модели.

У вероятностного программирования есть одна приятная особенность: если вы хотите использовать более простой каркас вероятностных рассуждений – пожалуйста. Системы вероятностного программирования позволяют представить широкий спектр существующих каркасов, а также такие системы, которые ни один из существующих каркасов представить не в состоянии. В этой книге вы узнаете о многих каркасах, в которых используется вероятностное программирование. Поэтому, изучая вероятностное программирование, вы заодно освоите многие современные каркасы вероятностных рассуждений.

1.2.2. Улучшенные языки имитационного моделирования

Полные по Тьюрингу языки вероятностного моделирования уже существуют. Обычно они называются *языками имитационного моделирования* (simulation languages). Мы знаем, что можно построить имитационную модель такого сложного процесса, как футбольный сезон, пользуясь языками программирования. В этом контексте я буду использовать термин *язык имитационного моделирова-*

ния для описания языка, который позволяет представить выполнение сложного процесса с элементами случайности. Как и вероятностные программы, такие имитационные модели выполняются случайным образом и порождают различные результаты. Имитационное моделирование широко используется как вероятностное рассуждение и применяется в самых разных областях, например: военном планировании, проектировании компонентов, здравоохранении и прогнозировании спортивных результатов. На самом деле, широкое распространение изощренных имитационных моделей говорит о потребности в развитых языках вероятностного моделирования.

Но вероятностная программа – нечто гораздо большее, чем имитационная модель. Имитационное моделирование позволяет реализовать лишь один аспект вероятностного программирования: предсказание будущего. Его нельзя использовать для объяснения причин наблюдаемых результатов. И хотя модель можно улучшить, дополнив ее новой актуальной информацией, трудно включить неизвестную информацию, которую нужно вывести. Поэтому способность к обучению на прошлом опыте с целью улучшения будущих прогнозов и анализа оказывается ограниченной. Использовать имитационные модели в машинном обучении невозможно.

Вероятностную программу можно уподобить имитационной модели, которая допускает не только выполнение, но и анализ. При разработке вероятностного программирования пришло осознание того, что многие алгоритмы вывода, используемые в более простых системах моделирования, можно использовать и для имитационного моделирования. Поэтому открывается возможность создать вероятностную модель, написав имитационную модель и применив к ней алгоритмы вывода.

И последнее. Системы вероятностных рассуждений уже существуют, например программы Hugin, Netica и BayesiaLab предлагают реализацию байесовских сетей. Но более выразительные языки представления, применяемые в вероятностном программировании, настолько новые, что мы лишь начинаем открывать для себя таящуюся в них мощь. Положа руку на сердце, не могу сказать, что вероятностное программирование уже используется во многих прикладных системах. Но некоторые нетривиальные приложения все же существуют. Microsoft с помощью вероятностного программирования научилась определять истинный уровень умения игроков в онлайновые игры. Стюарт Рассел из Калифорнийского университета в Беркли написал программу, которая следит за соблюдением Договора о всеобъемлющем запрещении ядерных испытаний, определяя сейсмическую активность, которая могла бы указывать на ядерный взрыв. Джош Тененбаум из Массачусетского технологического института и Ноа Гудман из Стэнфордского университета создали вероятностные программы для моделирования когнитивных способностей человека, они обладают очень неплохими объяснительными возможностями. В компании Charles River Analytics мы использовали вероятностное программирование для вывода заключений о компонентах вредоносного ПО и определения их эволюции. Но я полагаю, что все это – только вершина айсберга. Системы вероятностного программирования достигли такого уровня, что могут использоваться гораздо большим числом специалистов для принятия решений в своих предметных областях. Прочитав эту книгу, вы получите шанс присоединиться к новой технологии, пока не стало слишком поздно.

1.3. Введение в Figaro, язык вероятностного программирования

В этой книге мы будем работать с системой вероятностного программирования Figaro. (Я назвал ее в память персонажа оперы Моцарта «Женитьба Фигаро». Я люблю Моцарта и играл доктора Бартоло в Бостонской постановке этой оперы.) Главная цель книги – научить читателя принципам вероятностного программирования, так чтобы изученные методы можно было перенести на другие системы, некоторые из которых перечислены в приложении В. Но есть и другая цель – приобрести практический опыт создания вероятностных программ и предоставить инструменты, которыми можно воспользоваться немедленно. Поэтому многие примеры конкретизированы с помощью кода на Figaro.

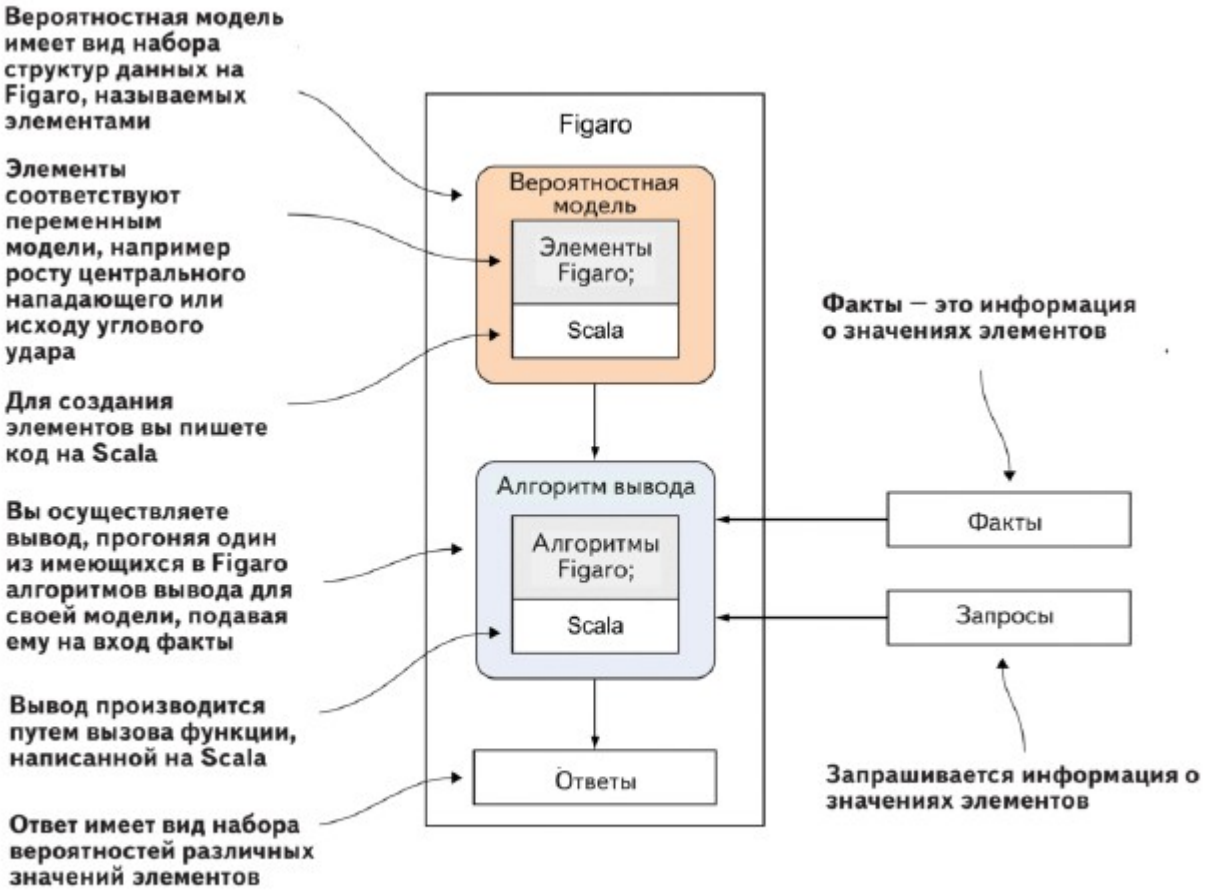

Рис. 1.9. Как в Figaro используется язык Scala для построения системы вероятностного программирования

Исходный код Figaro открыт, поддержка осуществляется с помощью GitHub, а сама система разрабатывается с 2009 года. Она реализована в виде библиотеки, написанной на Scala. На рис. 1.9 показано, как в Figaro используется Scala для реализации системы вероятностного программирования. Это уточнение рис. 1.7, где описаны основные компоненты такой системы. Начнем с вероятностной модели. В Figaro модель состоит из структур данных, называемых *элементами*. Каждый элемент представляет переменную, которая может принимать произвольное число значений. Структуры данных реализованы на Scala, и для создания модели, в которой они используются, вы пишете программу на Scala. Программе можно подать на вход факты, содержащие информацию о значениях элементов, а в запросе указать, какие выходные элементы вас интересуют. Вы выбираете один из встроенных в Figaro алгоритмов выбора и применяете его к своей модели, чтобы получить ответ на запрос при заданных фактах. Алгоритмы вывода реализованы на Scala, а применение алгоритм сводится к вызову функции Scala. Результатом вывода являются вероятности различных значений элементов, указанных в запросе.

Тот факт, что Figaro написана на Scala, дает ряд преимуществ. Некоторые из них связаны с использованием языка общего назначения вместо специализированного, другие – со спецификой именно Scala. Сначала остановимся на преимуществах вложения в язык общего назначения.

- Факты можно представлять, используя программу на объемлющем языке. Например, программа может прочитать данные из файла, каким-то образом обработать их и представить в виде фактов для модели Figaro. На специализированном независимом языке сделать это было бы гораздо труднее.
- Аналогично ответы, возвращенные Figaro, можно использовать в программе. Например, если менеджер футбольной команды работает с некоторой программой, то эта программа может запросить вероятность гола с тем, чтобы порекомендовать менеджеру варианты действий.
- Код на языке общего назначения можно включить в вероятностную программу. Рассмотрим, к примеру, физическую модель траектории мяча после удара головой. Эту модель можно было бы сделать частью элемента Figaro.
- Для построения модели Figaro можно использовать стандартные приемы программирования. Например, можно завести словарь, который содержит элементы Figaro, соответствующие всем игрокам команды, и выбирать из него элементы, исходя из ситуации, складывающейся на поле.

Теперь перечислим причины, по которым Scala особенно хорошо подходит на роль объемлющего языка для системы вероятностного программирования.

Поскольку Scala – функциональный язык программирования, Figaro также получает все преимущества функционального программирования, что позволяет естественно записывать многие модели. Я продемонстрирую это во второй части книги.

Scala – объектно-ориентированный язык, и то, что он сочетает в себе функциональные и объектно-ориентированные черты, является дополнительным преимуществом. Figaro также является объектно-ориентированным. Во второй части

я покажу, что это полезно для выражения некоторых паттернов проектирования в вероятностном программировании.

Наконец, ряд преимуществ Figaro не связан с вложением в Scala.

- На Figaro можно представить чрезвычайно широкий спектр вероятностных моделей. Элементы Figaro могут принимать значения любого типа, в том числе: булевы, целые, с двойной точностью, массивы, деревья, графы и т. д. Связи между элементами можно определить с помощью произвольной функции.
- Figaro предоставляет развитые средства задания фактов с помощью условий и ограничений.
- В Figaro имеется обширный набор алгоритмов вывода.
- Figaro позволяет представлять динамические модели ситуаций, изменяющихся во времени, и рассуждать о них.
- Figaro дает возможность включать в модель явные решения и поддерживает вывод оптимальных решений.

Использование Scala

Поскольку система Figaro реализована в виде библиотеки на Scala, для работы с ней необходимы практические навыки работы со Scala. Эта книга посвящена вероятностному программированию, поэтому я не ставил себе задачу научить Scala. Для этой цели есть немало замечательных ресурсов, например Школа Scala в Твиттере (http://twitter.github.io/scala_school). Но на случай, если вы пока не уверены в своих познаниях, я буду объяснять используемые средства Scala по ходу дела. Вы сможете читать книгу, даже если совсем не знаете Scala.

Чтобы пользоваться всеми преимуществами вероятностного программирования и Figaro, не нужно знать Scala в совершенстве. В этой книге я избегал продвинутых и малопонятных возможностей Scala. С другой стороны, чем лучше вы знаете Scala, тем больших успехов достигнете в работе с Figaro. Возможно даже, что после прочтения этой книги вы станете лучше владеть Scala.

Есть несколько причин, по которым Figaro – удобный язык для изучения вероятностного программирования.

- Будучи реализован в виде библиотеки на Scala, Figaro может быть использован в программах, написанных на Java и Scala, что упрощает его интеграцию с приложениями.
- По той же причине Figaro позволяет использовать для построения моделей все богатство объемлющего языка. Scala – современный передовой язык программирования, обладающий множеством полезных средств для организации программ, и все это вы автоматически получаете в свое распоряжение, работая с Figaro.
- Figaro располагает полным набором алгоритмов.

1.3.1. Figaro и Java: построение простой системы вероятностного программирования

Для иллюстрации достоинств вероятностного программирования и языка Figaro я покажу два способа написания простого приложения: сначала на языке Java, с которым вы, возможно, знакомы, а потом на Figaro. Хотя у Scala есть некоторые преимущества по сравнению с Java, не в этом состоит различие, которое я хочу подчеркнуть. Основная идея в том, что *Figaro содержит такие средства для представления вероятностных моделей и выполнения вывода из них, которые недоступны без вероятностного программирования.*

Наше скромное приложение будет играть роль примера «Hello World» для Figaro. Представьте, что человек просыпается утром, смотрит, ясно ли на улице, и произносит приветствие, зависящее от погоды. Так происходит два дня подряд. Кроме того, погода во второй день зависит от первого: если в первый день было ясно, то вероятность, что и на завтра будет солнечно, повышается. Эти предложения на естественном языке можно записать количественно, как показано в табл. 1.1.

Таблица 1.1. Количественные вероятности в примере «Hello World»

Погода сегодня		
Ясно	0.2	
Пасмурно	0.8	
Приветствие сегодня		
Если сегодня на улице ясно	«Здравствуй, мир!»	0.6
	«Здравствуй, вселенная!»	0.4
Если сегодня на улице пасмурно	«Здравствуй, мир!»	0.2
	«О нет, только не это»	0.8
Погода завтра		
Если сегодня на улице ясно	Ясно	0.8
	Пасмурно	0.2
Если сегодня на улице пасмурно	Ясно	0.05
	Пасмурно	0.95

Приветствие завтра		
Если завтра на улице ясно	«Здравствуй, мир!»	0.6
	«Здравствуй, вселенная!»	0.4
Если завтра на улице пасмурно	«Здравствуй, мир!»	0.2
	«О нет, только не это»	0.8

А пока достаточно интуитивного понимания того, что означает фраза «сегодня будет ясно с вероятностью 0.2». Аналогично, если завтра на улице будет ясно, то с вероятностью 0.6 будет произнесено приветствие «Здравствуй, мир!» и с вероятностью 0.4 – «Здравствуй, вселенная!».

Наметим три задачи, которые должна решать эта модель. В разделе 1.1.3 мы видели три типа рассуждений, доступных вероятностной модели: *предсказание* будущего, *вывод* прошлых событий, приведших к наблюдаемому результату, и обучение на прошлом опыте для лучшего предсказания будущего. Все это мы сделаем с помощью нашей простой модели. Точнее, задачи формулируются следующим образом.

1. Предсказать сегодняшнее приветствие.
2. Зная, что сегодня было произнесено приветствие «Здравствуй, мир!», сделать вывод о том, ясно ли на улице.
3. Зная, что сегодня было произнесено приветствие «Здравствуй, мир!», научиться предсказывать завтрашнее приветствие.

Посмотрим, как решить эти задачи на Java.

Листинг 1.1. Программа Hello World на Java

```
class HelloWorldJava {
   static String greeting1 = "Здравствуй, мир!";                    <-- (1)
   static String greeting2 = "Здравствуй, вселенная!";
   static String greeting3 = "О нет, только не это";

   static Double pSunnyToday = 0.2;                                 <-- (2)
   static Double pNotSunnyToday = 0.8;
   static Double pSunnyTomorrowIfSunnyToday = 0.8;
   static Double pNotSunnyTomorrowIfSunnyToday = 0.2;
   static Double pSunnyTomorrowIfNotSunnyToday = 0.05;
   static Double pNotSunnyTomorrowIfNotSunnyToday = 0.95;
   static Double pGreeting1TodayIfSunnyToday = 0.6;
   static Double pGreeting2TodayIfSunnyToday = 0.4;
   static Double pGreeting1TodayIfNotSunnyToday = 0.2;
   static Double pGreeting3IfNotSunnyToday = 0.8;
   static Double pGreeting1TomorrowIfSunnyTomorrow = 0.5;
   static Double pGreeting2TomorrowIfSunnyTomorrow = 0.5;
   static Double pGreeting1TomorrowIfNotSunnyTomorrow = 0.1;
   static Double pGreeting3TomorrowIfNotSunnyTomorrow = 0.95;
```

```
  static void predict() {
    Double pGreeting1Today =
           pSunnyToday * pGreeting1TodayIfSunnyToday +
           pNotSunnyToday * pGreeting1TodayIfNotSunnyToday;
    System.out.println("Сегодня будет приветствие " +
       greeting1 + " с вероятностью " + pGreeting1Today + ".");
  }

  static void infer() {
    Double pSunnyTodayAndGreeting1Today =
           pSunnyToday * pGreeting1TodayIfSunnyToday;
    Double pNotSunnyTodayAndGreeting1Today =
           pNotSunnyToday * pGreeting1TodayIfNotSunnyToday;
    Double pSunnyTodayGivenGreeting1Today =
           pSunnyTodayAndGreeting1Today /
           (pSunnyTodayAndGreeting1Today +
            pNotSunnyTodayAndGreeting1Today);
    System.out.println("Если сегодня произнесено приветствие " +
       greeting1 + ", то сегодня ясно с вероятностью  " +
       pSunnyTodayGivenGreeting1Today + ".");
  }

  static void learnAndPredict() {
    Double pSunnyTodayAndGreeting1Today =
           pSunnyToday * pGreeting1TodayIfSunnyToday;
    Double pNotSunnyTodayAndGreeting1Today =
           pNotSunnyToday * pGreeting1TodayIfNotSunnyToday;
    Double pSunnyTodayGivenGreeting1Today =
           pSunnyTodayAndGreeting1Today /
           (pSunnyTodayAndGreeting1Today +
            pNotSunnyTodayAndGreeting1Today);
    Double pNotSunnyTodayGivenGreeting1Today =
           1 - pSunnyTodayGivenGreeting1Today;
    Double pSunnyTomorrowGivenGreeting1Today =
           pSunnyTodayGivenGreeting1Today *
              pSunnyTomorrowIfSunnyToday +
           pNotSunnyTodayGivenGreeting1Today *
              pSunnyTomorrowIfNotSunnyToday;
    Double pNotSunnyTomorrowGivenGreeting1Today =
           1 - pSunnyTomorrowGivenGreeting1Today;
    Double pGreeting1TomorrowGivenGreeting1Today =
           pSunnyTomorrowGivenGreeting1Today *
              pGreeting1TomorrowIfSunnyTomorrow +
           pNotSunnyTomorrowGivenGreeting1Today *
              pGreeting1TomorrowIfNotSunnyTomorrow;
    System.out.println("Если сегодня произнесено приветствие " +
       greeting1 + ", то завтра будет сказано " + greeting1 +
       " с вероятностью " +
       pGreeting1TomorrowGivenGreeting1Today);
  }

  public static void main(String[] args) {
    predict();
    infer();
    learnAndPredict();
  }
}
```

❸

❹

❺

❻

❶ — Определяем приветствия
❷ — Задаем числовые параметры модели
❸ — Предсказываем сегодняшнее приветствие, применяя правила вероятностного вывода
❹ — Из того, что было произнесено приветствие «Здравствуй, мир!», выводим сегодняшнюю погоду, применяя правила вероятностного вывода
❺ — Из того, что было произнесено приветствие «Здравствуй, мир!», обучаемся предсказывать завтрашнее приветствие, применяя правила вероятностного вывода
❻ — Метод main, который выполняет все задачи

Не буду здесь описывать, как выполняются вычисления с применением правил вероятностного вывода. В коде используются три правила вывода: цепное правило, правило полной вероятности и правило Байеса. Все они будут объяснены в главе 9. А пока выделим две серьезные проблемы в этой программе.

- *Не существует способа определить структуру модели.*
  Определение модели содержится в списке имен переменных, принимающих значения типа double. Когда в начале этого раздела я описывал модель и приводил числа в табл. 1.1, структура модели была очевидна и более-менее понятна, пусть и на интуитивном уровне. У списка же переменных нет вообще никакой структуры. Назначение переменных закодировано только в их именах, что нельзя назвать хорошей идеей. Следовательно, написать модель таким образом трудно, и этот путь чреват ошибками. К тому же, написанный код трудно читать и сопровождать. Если потребуется модифицировать модель (например, сделать так, чтобы приветствие зависело еще и от того, хорошо ли вы спали), то, скорее всего, придется переписать большие куски кода.
- *Самостоятельно кодировать правила вывода трудно и чревато ошибками.*
  Вторая серьезная проблема связана с кодом, в котором для ответа на запросы используются правила вероятностного вывода. Чтобы написать такой код, необходимо хорошо знать правила вывода. Даже если такие знания имеются, написать код правильно все равно тяжело. И проверить, получен ли правильный ответ, тоже трудно. А ведь это очень простой пример. В сложном приложении написать код рассуждений подобным способом вообще нереально.

Листинг 1.2. Программа Hello World на Figaro

```
import com.cra.figaro.language.{Flip, Select}                    ◁—❶
import com.cra.figaro.library.compound.If
import com.cra.figaro.algorithm.factored.VariableElimination

object HelloWorld {
   val sunnyToday = Flip(0.2)                                    ❷—▷
   val greetingToday = If(sunnyToday,
        Select(0.6 -> "Здравствуй, мир!", 0.4 -> "Здравствуй, вселенная!"),
        Select(0.2 -> "Здравствуй, мир!", 0.8 -> "О нет, только не это"))
   val sunnyTomorrow = If(sunnyToday, Flip(0.8), Flip(0.05))
   val greetingTomorrow = If(sunnyTomorrow,
        Select(0.6 -> "Hello, world!", 0.4 -> "Howdy, universe!"),
        Select(0.2 -> "Hello, world!", 0.8 -> "Oh no, not again"))  ❷—▷

   def predict() {                                               ◁—❸
     val result = VariableElimination.probability(greetingToday,
                  "Здравствуй, мир!")
     println("Сегодня будет приветствие \"Здравствуй, мир!\" " +
             "с вероятностью " + result + ".")
   }

   def infer() {                                                 ❹—▷
     greetingToday.observe("Здравствуй, мир!")
     val result = VariableElimination.probability(sunnyToday, true)
     println("Если сегодня произнесено приветствие  \"Здравствуй, мир\", " +
             "то будет солнечно с вероятностью " + result + ".")
   }

   def learnAndPredict() {                                       ❺—▷
     greetingToday.observe("Здравствуй, мир!")
     val result = VariableElimination.probability(greetingTomorrow,
                  "Hello, world!")
     println("Если сегодня произнесено приветствие  \"Здравствуй, мир!\", " +
             "то завтра будет сказано \"Здравствуй, мир!\" " +
             "с вероятностью " + result + ".")
   }

   def main(args: Array[String]) {                               ◁—❻
     predict()
     infer()
     learnAndPredict()
   }
}
```

❶ — Импортируем конструкции Figaro
❷ — Определяем модель
❸ — Предсказываем сегодняшнее приветствие, применяя алгоритм вывода
❹ — Применяем алгоритм вывода, чтобы вывести сегодняшнюю погоду из того, что сегодня было произнесено приветствие «Здравствуй, мир!»
❺ — Применяя алгоритм вывода, обучаемся предсказывать завтрашнее приветствие, зная, что сегодня было произнесено «Здравствуй, мир!»
❻ — Метод main, который выполняет все задачи