7

Моделирование с помощью деревьев решений

Выше вы ознакомились с различными автоматическими классификаторами, а в этой главе мы продолжим эту тему и поговорим об очень полезном методе, который называется *обучением деревьев решений*. В отличие от других классификаторов, модели, порождаемые деревьями решений, легко поддаются интерпретации. Список чисел, которые выдает байесовский классификатор, говорит об относительной важности каждого слова, но для получения окончательного результата необходимо произвести вычисления. Интерпретировать результаты, вырабатываемые нейронной сетью, еще сложнее, поскольку вес связи между двумя нейронами сам по себе мало что значит. Для того же чтобы понять, как «рассуждало» дерево решения, достаточно просто взглянуть на него, а при желании можно даже представить весь процесс в виде последовательности предложений if–then (если–то).

В этой главе мы рассмотрим три примера использования деревьев решений. В первом из них мы покажем, как спрогнозировать количество пользователей сайта, которые готовы заплатить за премиальный доступ. Многие онлайновые сайты, взимающие плату за подписку или за каждое использование, предлагают пользователям сначала немного поработать с приложением, а только потом раскошеливаться. Если речь идет о подписке, то сайт обычно предлагает бесплатную пробную версию с ограниченным временем или же с урезанными возможностями.

Сайты, взимающие плату за каждое использование, могут предложить один бесплатный сеанс или еще что-то в этом роде.

В других примерах мы применим деревья решений к моделированию цен на недвижимость и к оценке степени привлекательности.

Прогнозирование количества регистраций

Когда сайт с большим числом посетителей развертывает новое приложение, предлагая бесплатный доступ и подписку, оно может привлечь тысячи новых пользователей. Многие из них просто движимы любопытством и на самом деле в этом приложении не заинтересованы, поэтому вероятность того, что они станут платными клиентами, крайне мала. Из-за этого трудно выделить потенциальных клиентов, на которых стоит акцентировать маркетинговые усилия, поэтому многие сайты прибегают к массовой рассылке писем всем открывшим учетную запись, вместо того чтобы действовать более целенаправленно.

Для решения этой проблемы было бы полезно уметь прогнозировать вероятность того, что некий пользователь станет платным клиентом. Вы уже знаете, что для этой цели можно воспользоваться байесовским классификатором или нейронной сетью. Однако в данном случае очень важна четкость – если вы знаете, какие факторы указывают на то, что пользователь может стать клиентом, то можете использовать эту информацию при выработке рекламной стратегии, для того чтобы сделать некоторые разделы сайты более легкодоступными или для придумывания других способов увеличения количества платных клиентов.

Мы будем рассматривать гипотетическое онлайновое приложение, которое предлагается бесплатно на пробный период. Пользователь регистрируется для пробной работы, на какое-то число дней получает доступ к сайту, а потом должен решить, хочет ли он перейти на базовое или премиальное обслуживание. После того как пользователь зарегистрировался, о нем собирается информация, а в конце пробного периода владельцы сайта узнают, какие пользователи решили стать платными клиентами.

Чтобы не раздражать пользователей и завершить регистрацию как можно скорее, сайт не просит заполнять длинную анкету, а собирает информацию из протоколов сервера, например: с какого сайта пользователь попал сюда, его географическое положение, сколько страниц он просмотрел, прежде чем зарегистрировался, и т. д. Если собрать все эти данные и свести их в таблицу, то получится что-то похожее на табл. 7.1.

Таблица 7.1. Поведение пользователя на сайте и окончательное решение об оплате

Откуда пришел	Местонахождение	Читал FAQ	Сколько просмотрел страниц	Выбранное обслуживание
Slashdot	США	Да	18	Нет
Google	Франция	Да	23	Премиальное
Digg	США	Да	24	Базовое
Kiwitobes	Франция	Да	23	Базовое
Google	Великобритания	Нет	21	Премиальное
(напрямую)	Новая Зеландия	Нет	12	Нет
(напрямую)	Великобритания	Нет	21	Базовое
Google	США	Нет	24	Премиальное
Slashdot	Франция	Да	19	Нет
Digg	США	Нет	18	Нет
Google	Великобритания	Нет	18	Нет
Kiwitobes	Великобритания	Нет	19	Нет
Digg	Новая Зеландия	Да	12	Базовое
Google	Великобритания	Да	18	Базовое
Kiwitobes	Франция	Да	19	Базовое

Организуйте данные в виде списка строк, каждая из которых представляет собой список столбцов. В последнем столбце указывается, оформил данный пользователь платный контракт или нет. Именно это значение мы и хотели бы уметь прогнозировать. Создайте новый файл treepredict.py, с которым вы будете работать в этой главе. Если вы хотите вводить данные вручную, включите в начало файла такие строки:

```
my_data=[['slashdot','USA','yes',18,'None'],
         ['google','France','yes',23,'Premium'],
         ['digg','USA','yes',24,'Basic'],
         ['kiwitobes','France','yes',23,'Basic'],
         ['google','UK','no',21,'Premium'],
         ['(direct)','New Zealand','no',12,'None'],
         ['(direct)','UK','no',21,'Basic'],
         ['google','USA','no',24,'Premium'],
         ['slashdot','France','yes',19,'None'],
         ['digg','USA','no',18,'None'],
         ['google','UK','no',18,'None'],
         ['kiwitobes','UK','no',19,'None'],
         ['digg','New Zealand','yes',12,'Basic'],
         ['slashdot','UK','no',21,'None'],
```

```
['google', 'UK', 'yes', 18, 'Basic'],
['kiwitobes', 'France', 'yes', 19, 'Basic']]
```

Можете вместо этого загрузить набор данных со страницы *http://kiwitobes.com/tree/decision_tree_example.txt*.

Для загрузки нужно включить в файл такую строку:

```
my_data=[line.split('\t') for line in file('decision_tree_example.txt')]
```

Теперь у нас есть информация о том, где пользователь находится, как он попал на сайт и сколько времени провел на сайте перед тем, как зарегистрироваться. Нужно лишь заполнить последний столбец, поместив в него обоснованную гипотезу.

Введение в теорию деревьев решений

Деревья решений – один из простейших методов машинного обучения. Это совершенно прозрачный способ классификации наблюдений, и после обучения они представляются в виде последовательности предложений if–then (если–то), организованных в виде дерева. На рис. 7.1 приведен пример дерева решений для классификации фруктов.

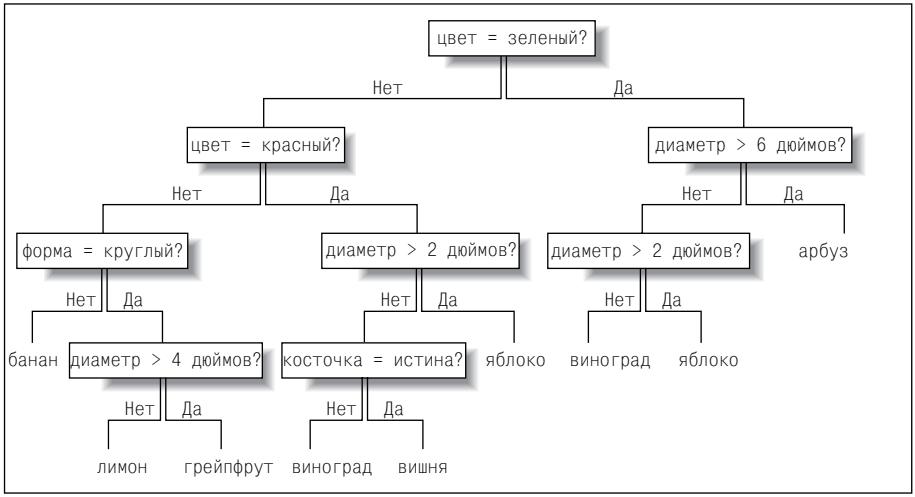

***Рис. 7.1.** Пример дерева решений*

Имея дерево решений, нетрудно понять, как оно принимает решения. Достаточно проследовать вниз по дереву, правильно отвечая на вопросы, – и в конечном итоге вы доберетесь до ответа. Обратная трассировка от узла, в котором вы остановились, до корня дает обоснование выработанной классификации.

В этой главе мы рассмотрим способ представления дерева решений, код построения дерева из реальных данных и код для классификации

новых наблюдений. Первым делом нужно создать представление дерева. Создайте новый класс `decisionnode`, с помощью которого мы будем представлять один узел дерева:

```
class decisionnode:
  def __init__(self,col=-1,value=None,results=None,tb=None,fb=None):
    self.col=col
    self.value=value
    self.results=results
    self.tb=tb
    self.fb=fb
```

В каждом узле имеется пять переменных экземпляра, и все они могут быть заданы в инициализаторе:

- `col` – индекс столбца проверяемого условия.
- `value` – значение, которому должно соответствовать значение в столбце, чтобы результат был равен `true`.
- `tb` и `fb` – экземпляры класса `decisionnodes`, в которые происходит переход в случае, если результаты – `true` или `false` соответственно.
- `results` – словарь результатов для этой ветви. Значение равно `None` для всех узлов, кроме листовых.

Функции, создающие дерево, возвращают корневой узел, от которого можно начать обход, следуя по ветвям `true` или `false`, пока не дойдем до узла, содержащего результаты.

Обучение дерева

В этой главе используется алгоритм *CART* (Classification and Regression Trees – деревья классификации и регрессии). Для построения дерева решений алгоритм сначала создает корневой узел. Рассмотрев все наблюдения в таблице, он выбирает наилучшую переменную, по которой можно разбивать данные на две части. Для этого нужно решить, какое условие (например, «Читал ли пользователь FAQ (Часто задаваемые вопросы)?») разобьет множество выходных данных так, чтобы было проще догадаться, что пользователь собирается сделать (на какое обслуживание он подпишется).

Функция `divideset` разбивает множество строк на два подмножества исходя из данных в одном столбце. На входе она принимает список строк, номер столбца и значение, по которому делить столбец. В случае прочтения FAQ возможные значения – «Да» или «Нет», а для столбца «Откуда пришел» есть несколько возможностей. Функция возвращает два списка строк: первый содержит те строки, для которых данные в указанной колонке соответствуют переданному значению, второй – остальные строки.

```
# Разбиение множества по указанному столбцу. Может обрабатывать как числовые,
# так и дискретные значения.
def divideset(rows,column,value):
```

```
# Создать функцию, которая сообщит, относится ли строка к первой группе
# (true) или ко второй (false)
split_function=None
if isinstance(value,int) or isinstance(value,float):
  split_function=lambda row:row[column]>=value
else:
  split_function=lambda row:row[column]==value

# Разбить множество строк на две части и вернуть их
set1=[row for row in rows if split_function(row)]
set2=[row for row in rows if not split_function(row)]
return (set1,set2)
```

В этом коде создается функция `split_function`, которая разбивает данные на две части. Это делается по-разному в зависимости от того, является ли множество значений непрерывным (float) или дискретным (int). В первом случае возврат `true` означает, что значение в столбце больше `value`. Во втором `split_function` просто проверяет, совпадает ли значение в столбце с `value`. Затем созданная функция применяется для разбиения множества данных на две части: первая состоит из строк, для которые `split_function` вернула `true`, вторая – из тех, для которых она вернула `false`.

Запустите интерпретатор Python и попробуйте разбить множество результатов по столбцу «Читал FAQ»:

```
$ python
>>> import treepredict
>>> treepredict.divideset(treepredict.my_data,2,'yes')
([['slashdot', 'USA', 'yes', 18, 'None'], ['google', 'France', 'yes', 23,
'Premium'],...]]
[['google', 'UK', 'no', 21, 'Premium'], ['(direct)', 'New Zealand', 'no',
12,
'None'],...])
```

В табл. 7.2 показано получившееся разбиение.

Таблица 7.2. Разбиение множества результатов по значениям в столбце «Читал FAQ»

True	False
Нет	Премиальное
Премиальное	Нет
Базовое	Базовое
Базовое	Премиальное
Нет	Нет
Базовое	Нет
Базовое	Нет

Не похоже, что на данном этапе это хорошая переменная для разбиения результатов, поскольку множества в обоих столбцах неоднородны – они содержат все возможные значения. Нужно отыскать переменную получше.

Выбор наилучшего разбиения

Сделанное нами неформальное наблюдение о том, что переменная выбрана не очень хорошо, может быть и верным, но для реализации программы нужен способ измерения неоднородности множества. Требуется найти такую переменную, чтобы множества как можно меньше пересекались. Первое, что нам понадобится, – это функция для вычисления того, сколько раз каждый результат представлен в множестве строк. Добавьте ее в файл treepredict.py:

```
# Вычислить счетчики вхождения каждого результата в множество строк
# (результат - это последний столбец в каждой строке)
def uniquecounts(rows):
  results={}
  for row in rows:
    # Результат находится в последнем столбце
    r=row[len(row)-1]
    if r not in results: results[r]=0
    results[r]+=1
  return results
```

Функция `uniquecounts` возвращает словарь, в котором для каждого результата указано, сколько раз он встретился. Она используется в других функциях для вычисления неоднородности множества. Существует несколько метрик для этой цели, мы рассмотрим две из них: коэффициент Джини и энтропию.

Коэффициент Джини

Предположим, что есть множество образцов, принадлежащих нескольким категориям. Коэффициентом Джини называется вероятность того, что при случайном выборе образца и категории окажется, что образец не принадлежит к указанной категории. Если все образцы в множестве принадлежат к одной и той же категории, то гипотеза всегда будет верна, поэтому вероятность ошибки равна 0. Если в группе в равной пропорции представлены четыре возможных результата, то в 75% случаев гипотеза окажется неверной, поэтому коэффициент ошибки равен 0,75.

Ниже показана функция для вычисления коэффициента Джини:

```
# Вероятность того, что случайный образец принадлежит не к той категории
def giniimpurity(rows):
  total=len(rows)
  counts=uniquecounts(rows)
  imp=0
```

```
  for k1 in counts:
    p1=float(counts[k1])/total
    for k2 in counts:
      if k1==k2: continue
      p2=float(counts[k2])/total
      imp+=p1*p2
  return imp
```

Эта функция вычисляет вероятность каждого из возможных результатов путем деления счетчика появлений этого результата на общее число строк в множестве. Затем произведения этих вероятностей складываются. Это дает полную вероятность того, что для случайно выбранной строки будет спрогнозирован не тот результат, который в действительности имеет место. Чем выше эта вероятность, тем хуже разбиение. Вероятность 0 – это идеал, поскольку в этом случае все строки уже распределены правильно.

Энтропия

В теории информации *энтропией* называют меру беспорядочности множества – по сути говоря, меру его неоднородности. Добавьте в файл treepredict.py следующую функцию:

```
# Энтропия вычисляется как сумма p(x)log(p(x)) по всем различным
# результатам
def entropy(rows):
  from math import log
  log2=lambda x:log(x)/log(2)
  results=uniquecounts(rows)
  # Теперь вычислим энтропию
  ent=0.0
  for r in results.keys( ):
    p=float(results[r])/len(rows)
    ent=ent-p*log2(p)
  return ent
```

Функция `entropy` вычисляет частоту вхождения каждого образца (количество его вхождений, поделенное на общее число образцов – строк) и применяет следующие формулы:

$p(i)$ = частота вхождения = количество вхождений / количество строк

Энтропия = сумма $p(i) \times \log(p(i))$ по всем результатам

Эта величина является мерой того, насколько результаты отличаются друг от друга. Если все они одинаковы (например, вам повезло и все пользователи оформили премиальную подписку), то энтропия равна 0. Чем менее однородны группы, тем выше их энтропия. Наша цель состоит в том, чтобы разбить данные на две новые группы, так чтобы энтропия уменьшилась. Протестируйте метрики, основанные на коэффициенте Джини и на энтропии, в интерактивном сеансе:

```
>>> reload(treepredict)
<module 'treepredict' from 'treepredict.py'>
```

```
>>> treepredict.giniimpurity(treepredict.my_data)
0.6328125
>>> treepredict.entropy(treepredict.my_data)
1.5052408149441479
>>> set1,set2=treepredict.divideset(treepredict.my_data,2,'yes')
>>> treepredict.entropy(set1)
1.2987949406953985
>>> treepredict.giniimpurity(set1)
0.53125
```

Основное отличие энтропии от коэффициента Джини в том, что энтропия выходит на максимум более медленно. Поэтому она штрафует неоднородные множества чуть сильнее. В оставшейся части главы мы будем пользоваться энтропией, поскольку эта метрика чаще употребляется, но при желании ее легко заменить коэффициентом Джини.

Рекурсивное построение дерева

Чтобы оценить, насколько хорош выбранный атрибут, алгоритм сначала вычисляет энтропию всей группы. Затем он пытается разбить группу по возможным значениям каждого атрибута и вычисляет энтропию двух новых групп. Для определения того, какой атрибут дает наилучшее разбиение, вычисляется *информационный выигрыш*, то есть разность между текущей энтропией и средневзвешенной энтропией двух новых групп. Он вычисляется для каждого атрибута, после чего выбирается тот, для которого информационный выигрыш максимален.

Определив условие для корневого узла, алгоритм создает две ветви: по одной надо будет идти, когда условие истинно, по другой – когда ложно (рис. 7.2).

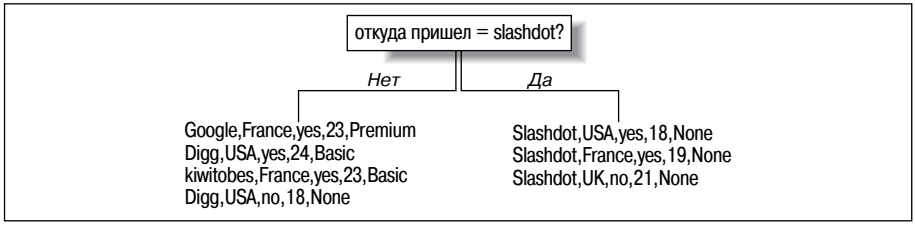

Рис. 7.2. *Дерево решений после одного расщепления*

Наблюдения разбиваются на удовлетворяющие и не удовлетворяющие условию. Для каждой ветви алгоритм определяет, нужно ли разбивать ее далее или мы уже пришли к однозначному заключению. Если какую-то из новых ветвей можно разбить, то с помощью описанного выше метода отыскивается подходящий атрибут. Результат после второго расщепления показан на рис. 7.3.

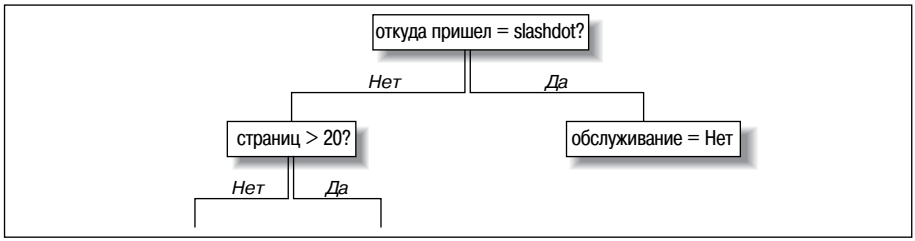

Рис. 7.3. *Дерево решений после двух расщеплений*

Таким образом, вычисляя для каждого узла наилучший атрибут и расщепляя ветви, алгоритм создает дерево. Рост ветви прекращается, когда информационный выигрыш, полученный от расщепления в данном узле, оказывается меньше или равен нулю.

Добавьте в файл treepredict.py функцию `buildtree`. Она рекурсивно строит дерево, выбирая на каждом шаге наилучший критерий расщепления:

```
def buildtree(rows,scoref=entropy):
  if len(rows)==0: return decisionnode( )
  current_score=scoref(rows)

  # Инициализировать переменные для выбора наилучшего критерия
  best_gain=0.0
  best_criteria=None
  best_sets=None

  column_count=len(rows[0])-1
  for col in range(0,column_count):
    # Создать список различных значений в этом столбце
    column_values={}
    for row in rows:
      column_values[row[col]]=1
    # Пробуем разбить множество строк по каждому значению
    # из этого столбца
    for value in column_values.keys( ):
      (set1,set2)=divideset(rows,col,value)

      # Информационный выигрыш
      p=float(len(set1))/len(rows)
      gain=current_score-p*scoref(set1)-(1-p)*scoref(set2)
      if gain>best_gain and len(set1)>0 and len(set2)>0:
        best_gain=gain
        best_criteria=(col,value)
        best_sets=(set1,set2)
  # Создаем подветви
  if best_gain>0:
```

```
    trueBranch=buildtree(best_sets[0])
    falseBranch=buildtree(best_sets[1])
    return decisionnode(col=best_criteria[0],value=best_criteria[1],
                        tb=trueBranch,fb=falseBranch)
  else:
    return decisionnode(results=uniquecounts(rows))
```

При первом вызове этой функции передается весь список строк. Она в цикле перебирает все столбцы (кроме последнего, в котором хранится результат) и по каждому значению, присутствующему в текущем столбце, разбивает множество строк на два подмножества. Для каждой пары подмножеств вычисляется средневзвешенная энтропия, для чего энтропия подмножества умножается на число попавших в него строк. Запоминается, у какой пары эта энтропия оказалась самой низкой.

Если средневзвешенная энтропия наилучшей пары подмножеств не меньше, чем у текущего множества, то рост этой ветви прекращается и сохраняются счетчики возможных результатов. В противном случае для каждого подмножества снова вызывается `buildtree` и результаты вызова присоединяются к ветвям `true` и `false`, исходящим из текущего узла. В итоге будет построено все дерево.

Описанный алгоритм можно применить к исходному набору данных. Приведенный выше код пригоден как для числовых, так и для дискретных данных. Кроме того, предполагается, что в последнем столбце каждой строки находится результат, поэтому для построения дерева достаточно передать только набор строк:

```
>>> reload(treepredict)
<module 'treepredict' from 'treepredict.py'>
>>> tree=treepredict.buildtree(treepredict.my_data)
```

Сейчас в переменной `tree` находится обученное дерево решений. В следующем разделе вы увидите, как построить его визуальное представление, а затем – как с его помощью делать прогнозы.

Отображение дерева

Итак, дерево у нас есть, но что с ним делать? Прежде всего, хотелось бы на него взглянуть. Функция `printtree` распечатывает дерево в текстовом виде. Представление получается не очень красивым, но это простой способ визуализировать небольшие деревья:

```
def printtree(tree,indent=''):
  # Это листовый узел?
  if tree.results!=None:
    print str(tree.results)
  else:
    # Печатаем критерий
    print str(tree.col)+':'+str(tree.value)+'? '

    # Печатаем ветви
    print indent+'T->',
```

```
    printtree(tree.tb,indent+' ')
    print indent+'F->',
    printtree(tree.fb,indent+' ')
```

Это еще одна рекурсивная функция. Она принимает на входе дерево, которое вернула функция `buildtree`, и обходит его сверху вниз. Обход прекращается, когда встретился узел, содержащий результаты (`results`). В противном случае печатается критерий выбора ветви `true` или `false`, а потом для каждой ветви снова вызывается `printtree` с предварительным увеличением ширины отступа.

Если вызвать эту функцию для построенного выше дерева, получится такая картина:

```
>>> reload(treepredict)
>>> treepredict.printtree(tree)
0:google?
T-> 3:21?
  T-> {'Premium': 3}
  F-> 2:yes?
    T-> {'Basic': 1}
    F-> {'None': 1}
F-> 0:slashdot?
  T-> {'None': 3}
  F-> 2:yes?
    T-> {'Basic': 4}
    F-> 3:21?
      T-> {'Basic': 1}
      F-> {'None': 3}
```

Это визуальное представление процедуры, выполняемой деревом решений при попытке классифицировать новый образец. В корневом узле проверяется условие «в столбце 0 находится Google?». Если это условие выполнено, то мы идем по ветви `T->` и обнаруживаем, что каждый пользователь, пришедший с Google, становится платным подписчиком, если просмотрел 21 страницу или более. Если условие не выполнено, мы идем по ветви `F->` и проверяем условие «в столбце 0 находится Slashdot?». Так продолжается до тех пор, пока мы не достигнем узла с результатами. Как уже упоминалось выше, возможность увидеть логику рассуждений – одно из существенных достоинств деревьев решений.

Графическое представление

Текстовое представление хорошо подходит для небольших деревьев, но по мере роста следить за тем, как выполнялся обход дерева, становится все труднее. В этом разделе мы покажем, как построить графическое представление дерева; это будет полезно для просмотра деревьев, которые мы будем создавать далее.

Код рисования дерева похож на код рисования дендрограмм в главе 3. В обоих случаях требуется изобразить двоичное дерево произвольной глубины, поэтому нам понадобятся функции для определения того,

сколько места отвести под каждый узел. Для этого нужно знать полную ширину всех его потомков и глубину узла, чтобы оценить, сколько места по вертикали потребуется для его ветвей. Полная ширина узла равна сумме ширин его дочерних узлов или 1, если дочерних узлов нет:

```
def getwidth(tree):
  if tree.tb==None and tree.fb==None: return 1
  return getwidth(tree.tb)+getwidth(tree.fb)
```

Глубина узла равна 1 плюс глубина самого глубокого из дочерних узлов:

```
def getdepth(tree):
  if tree.tb==None and tree.fb==None: return 0
  return max(getdepth(tree.tb),getdepth(tree.fb))+1
```

Для рисования дерева нам потребуется библиотека Python Imaging Library. Ее можно скачать с сайта *http://pythonware.com*, а в приложении А приведена дополнительная информация по установке. Добавьте в начало файла treepredict.py следующее предложение:

```
from PIL import Image,ImageDraw
```

Функция `drawtree` вычисляет требуемый размер и подготавливает холст. Затем она передает холст и корневой узел дерева функции `drawnode`. Добавьте ее в файл treepredict.py:

```
def drawtree(tree,jpeg='tree.jpg'):
  w=getwidth(tree)*100
  h=getdepth(tree)*100+120

  img=Image.new('RGB',(w,h),(255,255,255))
  draw=ImageDraw.Draw(img)

  drawnode(draw,tree,w/2,20)
  img.save(jpeg,'JPEG')
```

Функция `drawnode` отвечает за изображение узлов дерева решений. Она сначала рисует текущий узел и вычисляет позиции его дочерних узлов, а затем рекурсивно вызывает себя для каждого из дочерних узлов. Добавьте ее в файл treepredict.py:

```
def drawnode(draw,tree,x,y):
  if tree.results==None:
    # Вычислить ширину каждой ветви
    w1=getwidth(tree.fb)*100
    w2=getwidth(tree.tb)*100

    # Вычислить, сколько всего места нужно данному узлу
    left=x-(w1+w2)/2
    right=x+(w1+w2)/2

    # Вывести строку, содержащую условие
    draw.text((x-20,y-10),str(tree.col)+':'+str(tree.value),(0,0,0))

    # Нарисовать линии, ведущие к дочерним узлам
    draw.line((x,y,left+w1/2,y+100),fill=(255,0,0))
    draw.line((x,y,right-w2/2,y+100),fill=(255,0,0))
```

```
    # Нарисовать дочерние узлы
    drawnode(draw,tree.fb,left+w1/2,y+100)
    drawnode(draw,tree.tb,right-w2/2,y+100)
  else:
    txt=' \n'.join(['%s:%d'%v for v in tree.results.items( )])
    draw.text((x-20,y),txt,(0,0,0))
```

Попробуйте нарисовать текущее дерево в интерактивном сеансе:

```
>>> reload(treepredict)
<module 'treepredict' from 'treepredict.pyc'>
>>> treepredict.drawtree(tree,jpeg='treeview.jpg')
```

В результате будет создан файл treeview.jpg, изображенный на рис. 7.4. Метки True и False для ветвей не печатаются, так как они лишь загромождают диаграмму; просто следует иметь в виду, что ветвь True всегда правая. При таком соглашении становится легко следить за процессом рассуждения.

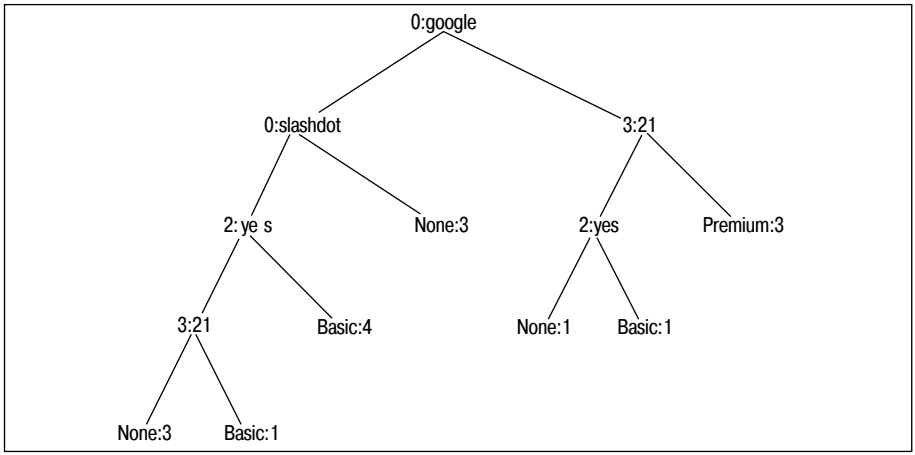

Рис. 7.4. *Дерево решений для прогноза платных подписчиков*

Классификация новых наблюдений

Теперь нам необходима функция, которая классифицирует новое наблюдение в соответствии с деревом решений. Добавьте ее в файл treepredict.py:

```
def classify(observation,tree):
  if tree.results!=None:
    return tree.results
  else:
    v=observation[tree.col]
    branch=None
    if isinstance(v,int) or isinstance(v,float):
      if v>=tree.value: branch=tree.tb
      else: branch=tree.fb
```

```
    else:
      if v==tree.value: branch=tree.tb
      else: branch=tree.fb
    return classify(observation,branch)
```

Эта функция обходит дерево примерно так же, как `printtree`. После каждого вызова проверяется, достигнут ли конец дерева, то есть имеется ли в узле список результатов `results`. Если нет, то для данного наблюдения проверяется условие в текущем узле. Если оно выполнено, то `classify` рекурсивно вызывается для ветви `true`, иначе – для ветви `false`.

Давайте воспользуемся функцией `classify` для получения прогноза относительного нового наблюдения:

```
>>> reload(treepredict)
<module 'treepredict' from 'treepredict.pyc'>
>>> treepredict.classify(['(direct)','USA','yes',5],tree)
{'Basic': 4}
```

Итак, мы имеем функции для создания дерева решений по набору данных, для отображения и интерпретации дерева и для классификации новых наблюдений. Эти функции можно применить к любому набору данных, состоящему из нескольких строк, каждая из которых содержит наблюдения и результат.

Отсечение ветвей дерева

У описанных выше методов обучения дерева есть один недостаток: оно может оказаться *переученным* (overfitted), то есть излишне ориентированным на данные, предъявленные в процессе обучения. Вероятность ответа, возвращенного переученным деревом, может оказаться выше, чем на самом деле, из-за того что были созданы ветви, лишь немного уменьшающие энтропию предъявленного множества наблюдений, хотя выбранное условие расщепления в действительности ничего не характеризует.

Деревья решений в реальном мире

Благодаря простоте интерпретации деревья решений являются одним из самых широко используемых методов добычи данных в анализе бизнеса, принятии решений в медицине и в выработке стратегии. Часто дерево решений создается автоматически, эксперт на его основе выделяет ключевые факторы, а затем уточняет дерево. Эта процедура позволяет эксперту воспользоваться помощью машины, а возможность наблюдать за процессом рассуждений помогает оценить качество прогноза.

Деревья решений применяются в таких приложениях, как профилирование клиентов, анализ финансовых рисков, помощь в диагностике и транспортное прогнозирование.

Описанный выше алгоритм продолжает расщеплять ветви, пока энтропия не перестанет уменьшаться. Остановить ветвление можно, например, задав минимальный порог уменьшения энтропии. Такая стратегия применяется часто, но страдает от одного мелкого недостатка – возможны такие наборы данных, для которых при одном расщеплении энтропия уменьшается чуть-чуть, а при последующих – очень сильно. Альтернативный подход – построить дерево целиком, как описано выше, а затем попытаться удалить лишние узлы. Эта процедура называется *сокращением* (pruning) дерева.

Для сокращения необходимо проверить пары узлов, имеющих общего родителя, и посмотреть, насколько увеличится энтропия при их объединении. Если увеличение окажется меньше заданного порога, то оба листа сливаются в один, для которого множество результатов получается объединением результатов в исходных листьях. Таким образом, мы избегаем феномена переучивания и не даем дереву сделать прогноз с большей долей уверенности, чем позволяют данные.

Добавьте в файл treepredict.py функцию сокращения дерева:

```
def prune(tree,mingain):
  # Если ветви не листовые, вызвать рекурсивно
  if tree.tb.results==None:
    prune(tree.tb,mingain)
  if tree.fb.results==None:
    prune(tree.fb,mingain)

  # Если обе подветви заканчиваются листьями, смотрим, нужно ли их
  # объединить
  if tree.tb.results!=None and tree.fb.results!=None:
    # Строим объединенный набор данных
    tb,fb=[],[]
    for v,c in tree.tb.results.items( ):
      tb+=[[v]]*c
    for v,c in tree.fb.results.items( ):
      fb+=[[v]]*c

  # Вычисляем, насколько уменьшилась энтропия
  delta=entropy(tb+fb)-(entropy(tb)+entropy(fb)/2)
  if delta<mingain:
    # Объединить ветви
    tree.tb,tree.fb=None,None
    tree.results=uniquecounts(tb+fb)
```

При вызове для корневого узла эта функция обходит все дерево, спускаясь до узлов, единственными дочерними узлами которых являются листья. Она объединяет списки результатов, хранящиеся в обоих листьях, и проверяет, насколько изменилась энтропия. Если изменение меньше параметра `mingain`, то оба листовых узла удаляются, а хранившиеся в них результаты перемещаются в узел-родитель. Теперь объединенный узел сам становится кандидатом на удаление и объединение с другим узлом.

Протестируем эту функцию на текущем наборе данных и посмотрим, приведет ли это к объединению каких-нибудь узлов:

```
>>> reload(treepredict)
<module 'treepredict' from 'treepredict.pyc'>
>>> treepredict.prune(tree,0.1)
>>> treepredict.printtree(tree)
0:google?
T-> 3:21?
  T-> {'Premium': 3}
  F-> 2:yes?
    T-> {'Basic': 1}
    F-> {'None': 1}
F-> 0:slashdot?
  T-> {'None': 3}
  F-> 2:yes?
    T-> {'Basic': 4}
    F-> 3:21?
      T-> {'Basic': 1}
      F-> {'None': 3}
>>> treepredict.prune(tree,1.0)
>>> treepredict.printtree(tree)
0:google?
T-> 3:21?
  T-> {'Premium': 3}
  F-> 2:yes?
    T-> {'Basic': 1}
    F-> {'None': 1}
F-> {'None': 6, 'Basic': 5}
```

В этом примере данные разбиваются довольно просто, поэтому сокращение с разумным минимумом ничего не дает. Только сделав минимум очень высоким, удается отсечь ветвь, исходящую из одного узла. Позже вы увидите, что реальные наборы данных разбиваются совсем не так удачно, как этот, поэтому отсечение ветвей оказывается куда более эффективным.

Восполнение отсутствующих данных

Еще одно достоинство деревьев решений заключается в их способности восполнять отсутствующие данные. В имеющемся у вас наборе данных какая-то информация может отсутствовать. Так, в рассматриваемом примере не всегда удается определить географическое местонахождение по IP-адресу, поэтому соответствующее поле может быть пусто. Чтобы приспособить дерево решений к такой ситуации, нужно будет по-другому реализовать функцию прогнозирования.

Если отсутствуют данные, необходимые для принятия решения о том, по какой ветви идти, то следует идти по *обеим* ветвям. Но при этом результаты по обе стороны не считаются равноценными, а взвешиваются. В обычном дереве решений каждому узлу неявно приписывается

вес 1, то есть считается, что при вычислении вероятности попадания образца в некоторую категорию имеющихся наблюдений достаточно. Если же мы идем одновременно по нескольким ветвям, то каждой ветви следует приписать вес, равный доле строк, оказавшихся по соответствующую сторону от узла.

Реализующая эту идею функция `mdclassify` получается небольшой модификацией `classify`. Добавьте ее в файл treepredict.py:

```
def mdclassify(observation,tree):
  if tree.results!=None:
    return tree.results
  else:
    v=observation[tree.col]
    if v==None:
      tr,fr=mdclassify(observation,tree.tb),mdclassify(observation,tree.fb)
      tcount=sum(tr.values( ))
      fcount=sum(fr.values( ))
      tw=float(tcount)/(tcount+fcount)
      fw=float(fcount)/(tcount+fcount)
      result={}
      for k,v in tr.items( ): result[k]=v*tw
      for k,v in fr.items( ): result[k]=v*fw
      return result
    else:
      if isinstance(v,int) or isinstance(v,float):
        if v>=tree.value: branch=tree.tb
        else: branch=tree.fb
      else:
        if v==tree.value: branch=tree.tb
        else: branch=tree.fb
      return mdclassify(observation,branch)
```

Единственное отличие имеется в конце: если существенная информация отсутствует, то мы вычисляем результаты для каждой ветви, а затем складываем их, предварительно умножив на соответствующие веса.

Протестируем функцию `mdclassify` на строке, в которой отсутствует важная информация:

```
>>> reload(treepredict)
<module 'treepredict' from 'treepredict.py'>
>>> treepredict.mdclassify(['google',None,'yes',None],tree)
{'Premium': 1.5, 'Basic': 1.5}
>>> treepredict2.mdclassify(['google','France',None,None],tree)
{'None': 0.125, 'Premium': 2.25, 'Basic': 0.125}
```

Как и следовало ожидать, отсутствие переменной «Сколько просмотрел страниц» дает сильные шансы на премиальное обслуживание и слабые шансы на базовое. Отсутствие переменной «Читал FAQ» дает другое распределение, где шансы на обоих концах взвешены с учетом количества образцов по обе стороны.

Числовые результаты

И моделирование поведения пользователей, и распознавание фруктов – это задачи классификации, поскольку результатом являются категории, а не числа. В следующих примерах, относящихся к ценам на недвижимость и к оценке степени привлекательности, мы рассмотрим задачи с числовыми результатами.

Хотя функцию `buildtree` можно применить к набору данных, где результатами являются числа, ничего хорошего из этого, скорее всего, не получится. Если все числа трактовать как различные категории, то алгоритм не будет принимать во внимание то, насколько сильно числа отличаются; все они будут рассматриваться как абсолютно независимые величины. Чтобы справиться с этой проблемой, необходимо в качестве критерия расщепления использовать дисперсию вместо энтропии или коэффициента Джини. Добавьте в файл treepredict.py функцию `variance`:

```
def variance(rows):
  if len(rows)==0: return 0
  data=[float(row[len(row)-1]) for row in rows]
  mean=sum(data)/len(data)
  variance=sum([(d-mean)**2 for d in data])/len(data)
  return variance
```

Эта функция, передаваемая `buildtree` в качестве параметра, вычисляет статистический разброс набора строк. Низкая дисперсия означает, что числа близки друг к другу, а высокая – что они сильно отличаются. При построении дерева с такой критериальной функцией условие расщепления в узле выбирается так, чтобы большие числа остались по одну сторону от узла, а малые – по другую. В результате должна уменьшиться суммарная дисперсия ветвей.

Моделирование цен на недвижимость

У деревьев решений много потенциальных применений, но наиболее полезны они в тех случаях, когда есть несколько возможных переменных и вас интересует процесс рассуждения. Иногда результаты уже известны, а смысл моделирования заключается в том, чтобы понять, почему они получились именно такими. Одна из областей, где подобная постановка вопроса особенно интересна, – это анализ цен на товары, особенно на такие, для которых измеряемые параметры существенно различаются. В этом разделе мы рассмотрим построение деревьев решений для моделирования цен на недвижимость, поскольку цены на дома различаются очень сильно и при этом существует много числовых и дискретных переменных, легко поддающихся измерению.

API сайта Zillow

Zillow – это бесплатная веб-служба, которая отслеживает цены на недвижимость и с помощью собранной информации дает оценку стоимости других домов. Она анализирует группы похожих домов и на этой основе прогнозирует новое значение. Процедура напоминает ту, которой пользуются оценщики. На рис. 7.5 приведена часть страницы сайта Zillow, на которой показана информация о доме и оценка его стоимости.

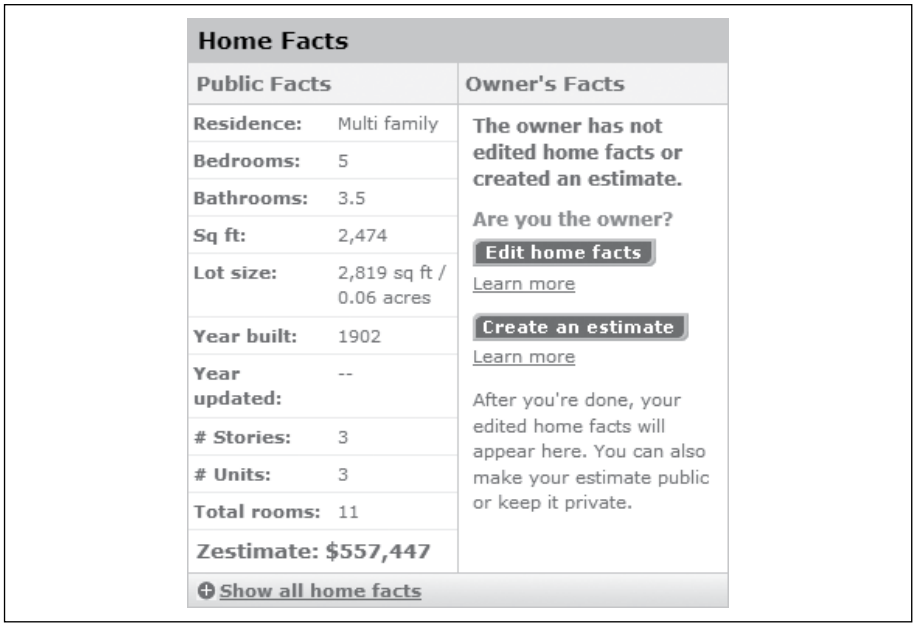

***Рис. 7.5.** Страница сайта zillow.com*

На наше счастье, сайт Zillow предоставляет API, позволяющий получить подробную информацию о доме и его оценочную стоимость. Этот API описан на странице *http://www.zillow.com/howto/api/APIOverview.htm*.

Для доступа к API необходим ключ разработчика, который можно бесплатно получить на сайте. Сам API несложен – достаточно включить в URL все параметры поиска, обратиться к сайту с запросом и разобрать полученный в ответ XML-документ, выделив из него такую информацию, как число спален и оценочная стоимость. Создайте новый файл zillow.py и включите в него такой код:

```
import xml.dom.minidom
import urllib2

zwskey="X1-ZWz1chwxis15aj_9skq6"
```

Как и в главе 5, для разбора XML-документа мы воспользуемся библиотекой minidom. Функция `getaddressdata` получает на входе адрес и город и создает URL с запросом к Zillow. Затем она разбирает полученный ответ, выделяет из него существенную информацию и возвращает ее в виде кортежа. Добавьте эту функцию в файл zillow.py:

```
def getaddressdata(address,city):
  escad=address.replace(' ','+')

  # Создаем URL
  url='http://www.zillow.com/webservice/GetDeepSearchResults.htm?'
  url+='zws-id=%s&address=%s&citystatezip=%s' % (zwskey,escad,city)

  # Разбираем возвращенный XML-документ
  doc=xml.dom.minidom.parseString(urllib2.urlopen(url).read( ))
  code=doc.getElementsByTagName('code')[0].firstChild.data

  # Код 0 означает успех; иначе произошла ошибка
  if code!='0': return None

  # Извлекаем информацию о данной недвижимости
  try:
    zipcode=doc.getElementsByTagName('zipcode')[0].firstChild.data
    use=doc.getElementsByTagName('useCode')[0].firstChild.data
    year=doc.getElementsByTagName('yearBuilt')[0].firstChild.data
    bath=doc.getElementsByTagName('bathrooms')[0].firstChild.data
    bed=doc.getElementsByTagName('bedrooms')[0].firstChild.data
    rooms=doc.getElementsByTagName('totalRooms')[0].firstChild.data
    price=doc.getElementsByTagName('amount')[0].firstChild.data
  except:
    return None

  return (zipcode,use,int(year),float(bath),int(bed),int(rooms),price)
```

Возвращенный этой функцией кортеж можно поместить в список в качестве наблюдения, поскольку «результат» – группа цен – находится в конце. Чтобы воспользоваться этой функцией для генерирования всего набора данных, нам необходим список адресов. Можете составить его сами или скачать список случайно сгенерированных адресов для города Кембридж, штат Массачусетс, со страницы *http://kiwitobes.com/addresslist.txt*.

Добавьте функцию `getpricelist`, которая читает этот файл и генерирует список данных:

```
def getpricelist( ):
  l1=[]
  for line in file('addresslist.txt'):
    data=getaddressdata(line.strip( ),'Cambridge,MA')
    l1.append(data)
  return l1
```

С помощью этих функций можно создать набор данных и построить дерево решений. Сделайте это в интерактивном сеансе:

```
>>> import zillow
>>> housedata=zillow.getpricelist( )
>>> reload(treepredict)
>>> housetree=treepredict.buildtree(housedata,scoref=treepredict.variance)
>>> treepredict.drawtree(housetree,'housetree.jpg')
```

На рис. 7.6 изображен созданный в результат файл housetree.jpg.

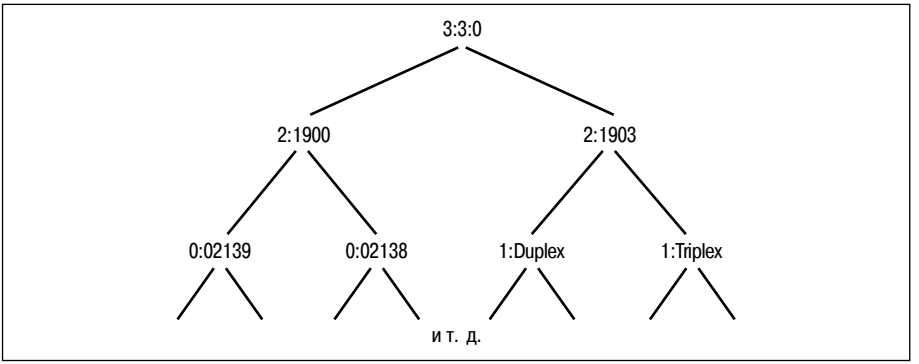

Рис. 7.6. *Дерево решений для цен на дома*

Разумеется, если вам нужна только оценка стоимости конкретного дома, то можно просто воспользоваться API сайта Zillow. Но заметьте следующее: вы только что построили модель факторов, принимаемых в расчет при определении цен на недвижимость. Обратите внимание, что в корне дерева оказался параметр `bathrooms` (число ванных комнат). Это означает, что дисперсия уменьшается больше всего, если разбить набор данных по числу ванных комнат. Решающим фактором при определении цен на дом в Кембридже является то, есть ли в нем три или более ванных комнат (обычно это означает, что речь идет о большом доме на несколько семей).

Очевидный недостаток использования деревьев решений в данном случае – это необходимость создавать группы цен, так как все они различны и должны быть как-то объединены, чтобы создать полезный листовый узел. Возможно, что для реальных данных о ценах больше подошел бы какой-нибудь другой метод прогнозирования. Один такой метод обсуждается в главе 8.

Моделирование степени привлекательности

Hot or Not – это сайт, на который пользователи могут загружать собственные фотографии. Первоначально идея состояла в том, чтобы одни пользователи могли оценивать внешность других. Собранные результаты обрабатывались, и каждому человеку выставлялась оценка от 1 до 10. С тех пор Hot or Not превратился в сайт знакомств и теперь предоставляет API, позволяющий получать демографическую информацию

о пользователях вместе с рейтингом их «привлекательности». Это интересный тестовый пример для модели деревьев решений, поскольку мы имеем набор входных переменных, единственную выходную переменную и потенциально любопытный процесс рассуждения. Да и сам по себе сайт представляет хороший пример коллективного разума.

Как обычно, для доступа к API нужен ключ разработчика. Вы можете зарегистрироваться и получить ключ на странице *http://dev.hotornot.com/signup*.

API сайта Hot or Not работает практически так же, как и другие рассмотренные выше API. В URL передаются параметры, и разбирается возвращенный XML-документ. Для начала создайте файл hotornot.py и включите в него предложения импорта и определение вашего ключа:

```
import urllib2
import xml.dom.minidom

api_key="479NUNJHETN"
```

Далее нужно получить случайный список людей, составляющих набор данных. К счастью, в API сайта Hot or Not предусмотрен вызов, возвращающий список людей, отвечающих заданным критериям. В нашем примере единственным критерием будет наличие профиля meet me (встретиться со мной), так как только из него можно получить такую информацию, как местонахождение и интересы. Добавьте в файл hotornot.py следующую функцию:

```
def getrandomratings(c):
  # Создаем URL для вызова функции getRandomProfile
  url="http://services.hotornot.com/rest/?app_key=%s" % api_key
  url+="&method=Rate.getRandomProfile&retrieve_num=%d" % c
  url+="&get_rate_info=true&meet_users_only=true"

  f1=urllib2.urlopen(url).read( )

  doc=xml.dom.minidom.parseString(f1)

  emids=doc.getElementsByTagName('emid')
  ratings=doc.getElementsByTagName('rating')

  # Объединяем идентификаторы и рейтинги в один список
  result=[]
  for e,r in zip(emids,ratings):
    if r.firstChild!=None:
      result.append((e.firstChild.data,r.firstChild.data))
  return result
```

После того как список идентификаторов и рейтингов пользователей сгенерирован, нам потребуется функция для загрузки информации о людях: пола, возраста, местонахождения и ключевых слов. Если в качестве местонахождения брать все 50 штатов, то получится слишком много ветвлений. Поэтому объединим некоторые штаты в регионы. Добавьте следующий код для задания регионов:

```
stateregions={'New England':['ct','mn','ma','nh','ri','vt'],
              'Mid Atlantic':['de','md','nj','ny','pa'],
              'South':['al','ak','fl','ga','ky','la','ms','mo',
                       'nc','sc','tn','va','wv'],
              'Midwest':['il','in','ia','ks','mi','ne','nd','oh','sd','wi'],
              'West':['ak','ca','co','hi','id','mt','nv','or','ut','wa','wy']
```

API предоставляет метод загрузки демографических данных об одном человеке, поэтому функция `getpeopledata` просто обходит в цикле результаты первого поиска и запрашивает у API детали. Добавьте ее в файл hotornot.py:

```
def getpeopledata(ratings):
  result=[]
  for emid,rating in ratings:
    # URL метода MeetMe.getProfile
    url="http://services.hotornot.com/rest/?app_key=%s" % api_key
    url+="&method=MeetMe.getProfile&emid=%s&get_keywords=true" % emid

    # Получить всю информацию об этом человеке
    try:
      rating=int(float(rating)+0.5)
      doc2=xml.dom.minidom.parseString(urllib2.urlopen(url).read( ))
      gender=doc2.getElementsByTagName('gender')[0].firstChild.data
      age=doc2.getElementsByTagName('age')[0].firstChild.data
      loc=doc2.getElementsByTagName('location')[0].firstChild.data[0:2]

      # Преобразуем штат в регион
      for r,s in stateregions.items( ):
        if loc in s: region=r

      if region!=None:
        result.append((gender,int(age),region,rating))
    except:
      pass
  return result
```

Теперь можно импортировать этот модуль в интерактивном сеансе и сгенерировать набор данных:

```
>>> import hotornot
>>> l1=hotornot.getrandomratings(500)
>>> len(l1)
442
>>> pdata=hotornot.getpeopledata(l1)
>>> pdata[0]
(u'female', 28, 'West', 9)
```

Список содержит информацию о каждом пользователе, причем в последнем поле представлен рейтинг. Эту структуру данных можно передать функции `buildtree` для построения дерева:

```
>>> hottree=treepredict.buildtree(pdata,scoref=treepredict.variance)
>>> treepredict.prune(hottree,0.5)
>>> treepredict.drawtree(hottree,'hottree.jpg')
```

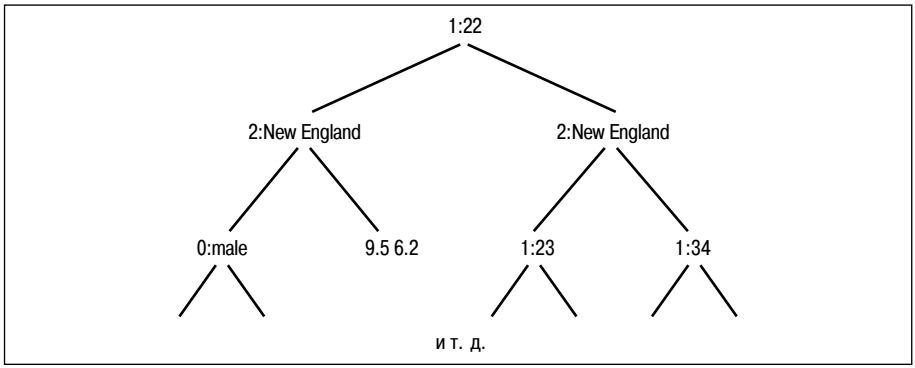

Рис. 7.7. *Моделирование привлекательности с помощью деревьев решений*

Построенное дерево изображено на рис. 7.7.

Корневой узел, по которому проводится начальное разбиение, соответствует полу. Остаток дерева довольно сложен, и читать его трудно. Но его, безусловно, можно использовать для прогнозов о людях, которые дереву не предъявлялись. Кроме того, поскольку алгоритм поддерживает восполнение отсутствующих данных, можно группировать людей по крупным переменным. Так, можно сравнить привлекательность живущих в южных и в среднеатлантических штатах:

```
>>> south=treepredict2.mdclassify((None,None,'South'),hottree)
>>> midat=treepredict2.mdclassify((None,None,'Mid Atlantic'),hottree)
>>> south[10]/sum(south.values( ))
0.055820815183261735
>>> midat[10]/sum(midat.values( ))
0.048972797320600864
```

Как видим, на основе этого набора данных можно сделать вывод о том, что выходцы с юга чуть более привлекательны. Можете попробовать рассмотреть другие факторы, например возраст, или проверить, у кого рейтинг выше – у мужчин или у женщин.

В каких случаях применять деревья решений

Пожалуй, основное достоинство деревьев решений – это простота интерпретации обученной модели. Применив алгоритм к рассмотренной задаче, мы получили не только дерево, способное делать прогнозы о поведении новых пользователей, но и список вопросов, на которые нужно ответить для выработки решения. Например, видно, что люди, приходящие на данный сайт с сайта Slashdot, никогда не становятся платными подписчиками, тогда как нашедшие его с помощью Google и просмотревшие по меньшей мер 20 страниц, вероятно, оформят подписку на премиальное обслуживание. Это, в свою очередь, наводит на мысль изменить рекламную стратегию, сфокусировавшись на сайтах, дающих наиболее высококачественный трафик. Мы выяснили также, что

некоторые переменные, например место проживания, не влияют на результат. Если какие-то данные трудно собрать, а в итоге оказывается, что они несущественны, то можно прекратить их сбор.

В отличие от других алгоритмов машинного обучения, деревья решения могут работать как с числовыми, так и с дискретными данными. В первом примере мы классифицировали страницы по нескольким дискретным показателям. Далее некоторые алгоритмы требуют предварительной подготовки или нормализации данных, а программы из этой главы принимают любой список данных, содержащих числовые или дискретные параметры, и строят соответствующее им дерево решений.

Деревья решений допускают также вероятностные прогнозы. В некоторых задачах для проведения четкого разграничения иногда не хватает данных – в дереве решений может встретиться узел, для которого есть несколько возможностей, а дальнейшее расщепление невозможно. Программа, представленная в этой главе, возвращает словарь счетчиков для различных результатов, и с помощью этой информации мы можем решить, в какой мере результат заслуживает доверия. Не все алгоритмы способны оценить вероятность результата в условиях неопределенности.

Однако у деревьев решений есть и очевидные недостатки. Они хорошо подходят для задач с небольшим числом возможных результатов, но неприменимы к наборам данных, где число возможных исходов велико. В нашем первом примере было всего три результата: «Нет», «Базовое» и «Премиальное». Если бы количество результатов исчислялось сотнями, то построенное дерево оказалось бы слишком сложным и, скорее всего, давало бы плохие прогнозы.

Еще один крупный недостаток рассмотренных выше деревьев решений заключается в том, что хотя они и способны работать с простыми числовыми данными, но условие может формулироваться только в терминах «больше/меньше». Это затрудняет применение деревьев решений к задачам, где класс определяется более сложным сочетанием переменных. Например, если бы результат определялся на основе величины разности между двумя переменными, то дерево выросло бы до невообразимых размеров и очень быстро утратило бы точность прогнозирования.

Подводя итог, можно сказать, что деревья решения – не самый удачный выбор для задач с большим количеством числовых входов и выходов или со сложными взаимосвязями между числовыми входами, какие встречаются, например, при интерпретации финансовых данных или анализе изображений. Напротив, деревья решения – отличный инструмент анализа наборов с большим числом дискретных и числовых данных с четкими точками расщепления. Они оптимальны, когда важно понимать процесс принятия решения; как вы могли убедиться, наблюдение за рассуждением иногда не менее важно, чем конечный прогноз.

Упражнения

1. *Вероятности результатов.* Текущие версии функции `classify` и `mdclassify` возвращают результат в виде набора счетчиков. Модифицируйте их так, чтобы они возвращали вероятности совпадения результатов с той или иной категорией.
2. *Диапазоны отсутствующих данных.* Функция `mdclassify` позволяет указывать в качестве отсутствующего значения строку `None`. Для числовых значений может случиться так, что точное значение неизвестно, но известно, что оно находится в некотором диапазоне. Модифицируйте `mdclassify` так, чтобы она могла принимать кортеж вида (20,25) в качестве значения и обходила обе ветви, если это необходимо.
3. *Ранний останов.* Вместо того чтобы сокращать построенное дерево, функция `buildtree` может просто прекратить расщепление, когда энтропия перестает уменьшаться достаточно ощутимо. В некоторых случаях такое решение не идеально, зато позволяет исключить один шаг. Модифицируйте `buildtree` так, чтобы она принимала в качестве параметра минимальный выигрыш и прекращала расщеплять ветвь, когда энтропия уменьшается на меньшую величину.
4. *Построение дерева в случае отсутствия данных.* Мы написали функцию, которая способна классифицировать строку с отсутствующими данными, но что если данные отсутствуют в обучающем наборе? Модифицируйте `buildtree` так, чтобы она проверяла отсутствие данных и в случае, когда невозможно отправить результат в одну ветвь, отправляла бы его в обе.
5. *Многопутевое расщепление* (трудная задача). Все рассмотренные в этой главе деревья решений были двоичными. Но для некоторых наборов данных структуру дерева можно было бы упростить, если бы было разрешено создавать более двух ветвей из одного узла. Как бы вы представили такое дерево? А как бы вы стали его обучать?