Лабораторная работа №8

Деревья решений

Деревья решений являются моделями, широко используемыми для решения задач классификации и регрессии. По сути они задают вопросы и выстраивают иерархию правил «если… то», приводящую к решению.

Эти вопросы похожи на вопросы, которые вы можете спросить в игре «20 Questions». Представьте, вам нужно научиться отличать друг от друга четыре вида животных: медведей, ястребов, пингвинов и дельфинов. Ваша цель состоит в том, чтобы получить правильный ответ, задав несколько вопросов. Вы могли бы начать с вопроса, есть ли у этих видов животных перья, вопроса, который сужает количество возможных видов животных до двух. Если получен ответ «да», вы можете задать еще один вопрос, который может помочь вам различать ястребов и пингвинов. Например, вы могли бы спросить, может ли данный вид животных летать. Если у этого вида животных нет перьев, ваши возможные варианты – дельфины и медведи, и вам нужно задать вопрос, чтобы провести различие между этими двумя видами животных, например, спросить, есть ли плавники у этого вида животных.

Эти вопросы можно выразить в виде дерева решений, как это показано на **рис. 8.1**.

```
mglearn.plots.plot_animal_tree()
plt.show()
```

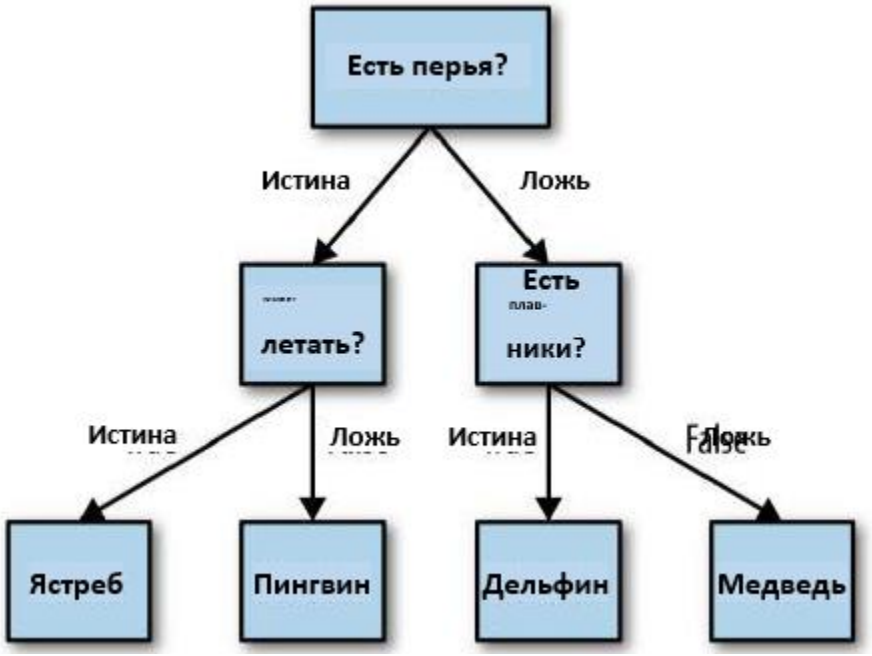

Рис. 8.1 Двумерный синтетический набор данных, содержащий три класса

На этом рисунке каждый узел дерева либо представляет собой либо вопрос, либо терминальный узел (его еще называют листом или **leaf**), который содержит ответ. Ребра соединяют вышестоящие узлы с нижестоящими.

Говоря языком машинного обучения, мы построили модель, различающую четыре класса животных (ястребов, пингвинов, дельфинов и медведей), используя три признака «есть перья», «может летать» и «имеет плавники». Вместо того, чтобы строить эти модели вручную, мы можем построить их с помощью контролируемого обучения.

Построение деревьев решений

Давайте рассмотрим процесс построения дерева решений для двумерного классификационного набора данных, показанного на **рис. 8.2**. Набор данных состоит из точек, обозначаемых маркерами двух типов. Каждому типу маркера соответствует свой класс, на каждый класс приходится по точек данных. Назовем этот набор данных **two_moons.**

Построение дерева решений означает построение последовательности правил «если… то…», которая приводит нас к истинному ответу максимально коротким путем. В машинном обучении эти правила называются тестами (**tests**). Не путайте их с тестовым набором, который мы используем для проверки обобщающей способности нашей модели. Как правило, данные бывают представлены не только в виде бинарных признаков да/нет, как в примере с животными, но и в виде непрерывных признаков, как в двумерном наборе данных, показанном на **рис. 8.2**. Тесты, которые используются для непрерывных данных имеют вид «Признак i больше значения a?"

```
mglearn.plots.plot_tree_progressive()
plt.show()
```

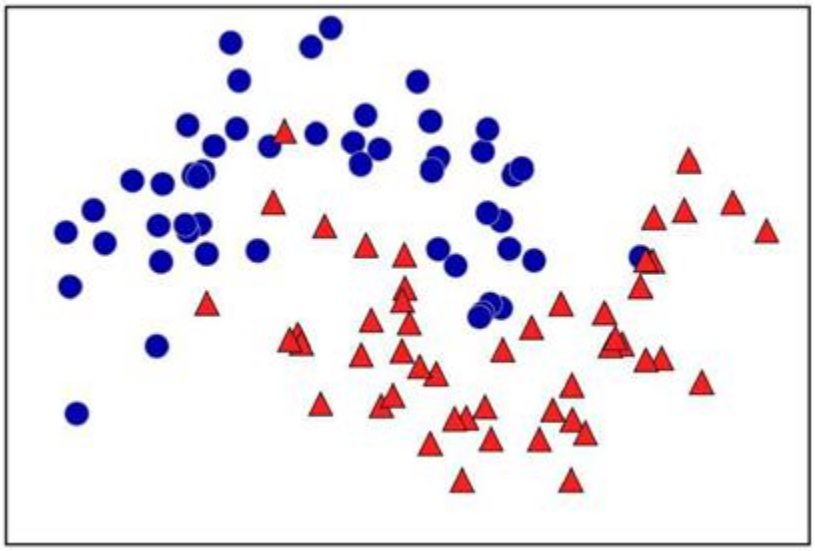

Рис. 8.2 Набор данных two_moons, по которому будет построено дерево решений

Чтобы построить дерево, алгоритм перебирает все возможные тесты и находит тот, который является наиболее информативным с точки зрения прогнозирования значений целевой переменной. **Рис. 8.3** показывает первый выбранный тест. Разделение набора данных по горизонтали в точке x[1]=0.0596 дает наиболее полную информацию. Оно лучше всего разделяет точки класса 0 от точек класса 1. Верхний узел, также называемый корнем (**root**), представляет

собой весь набор данных, состоящий из 50 точек, принадлежащих к классу 0, и 50 точек, принадлежащих к классу 1. Разделение выполняется путем тестирования x[1]<=0.0596, обозначенного черной линией. Если тест верен, точка назначается левому узлу, который содержит 2 точки, принадлежащие классу 0, и 32 точки, принадлежащие классу 1. В противном случае точка будет присвоена правому узлу, который содержит 48 точек, принадлежащих классу 0, и 18 точек, принадлежащих классу 1. Эти два узла соответствуют верхней и нижней областям, показанным на рис. 2-

Несмотря на то что первое разбиение довольно хорошо разделило два класса, нижняя область по-прежнему содержит точки, принадлежащие к классу 0, а верхняя область по-прежнему содержит точки, принадлежащие к классу 1. Мы можем построить более точную модель, повторяя процесс поиска наилучшего теста в обоих областях. **Рис. 8.4** показывает, что следующее наиболее информативное разбиение для левой и правой областей основывается на x[0].

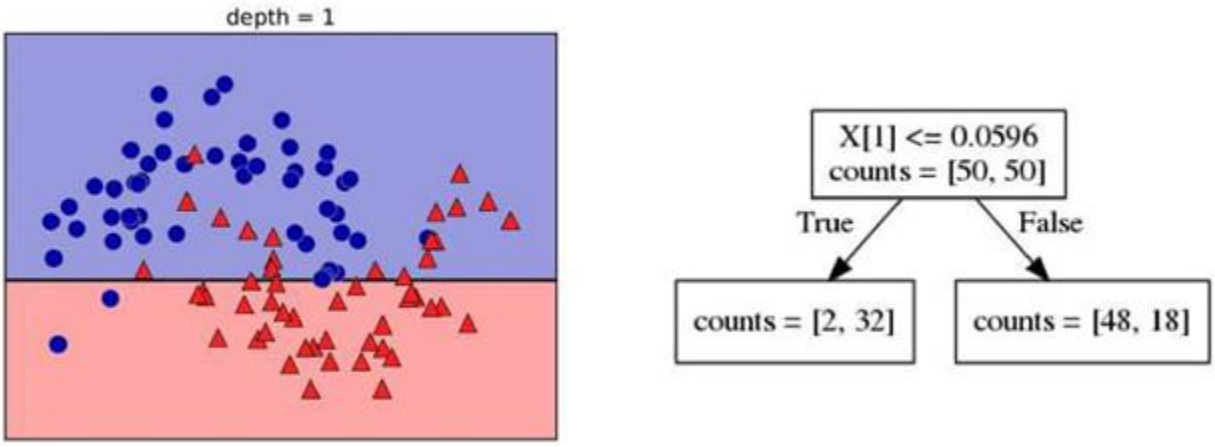

Рис. 8.3 Граница принятия решений, полученная с помощью дерева глубиной 1 (слева) и соответствующее дерево решений (справа)

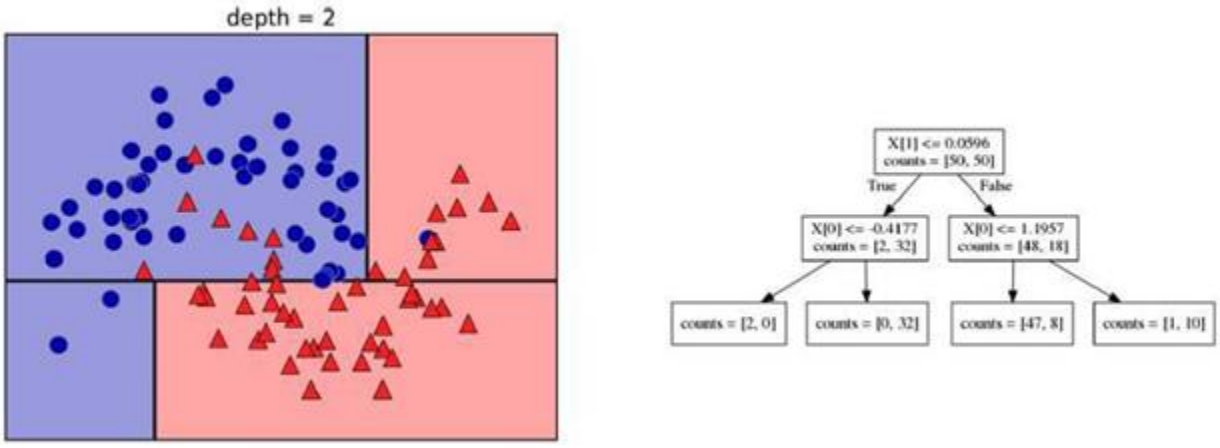

Рис. 8.4 Граница принятия решений, полученная с помощью дерева глубиной 2 (слева) и соответствующее дерево решений (справа)

Рекурсивное разбиение данных повторяется до тех пор, пока все точки данных в каждой области разбиения (каждом листе дерева решений) не будут принадлежать одному и тому же значению целевой переменной (классу или количественному значению). Лист дерева, который содержит точки данных, относящиеся к одному и тому же значению целевой переменной, называется чистым (**pure**). Итоговое разбиение для нашего набора данных показано на **рис. 8.5**.

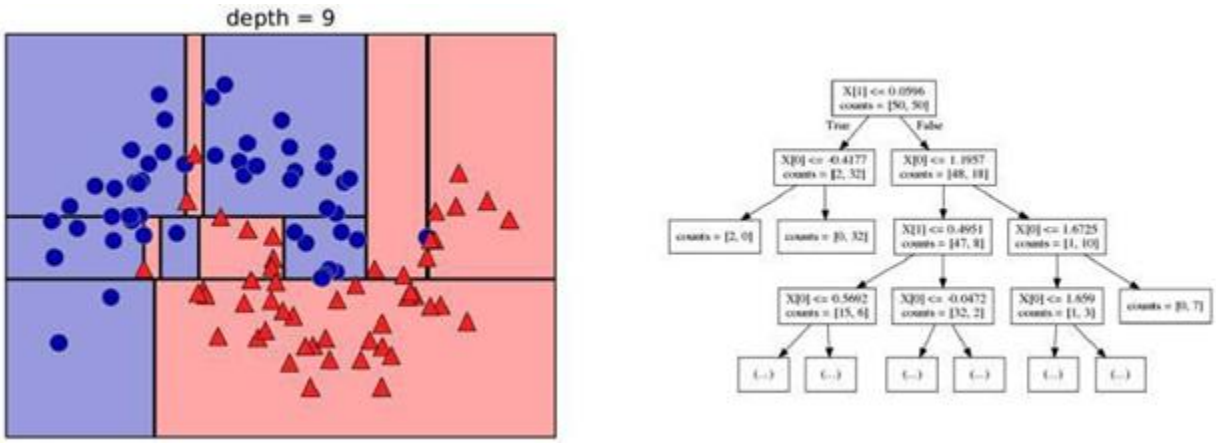

Рис. 8.5 Граница принятия решений, полученная с помощью дерева глубиной 9 (слева) и фрагмент соответствующего дерева (справа), полное дерево имеет довольно большой размер и его сложно визуализировать

Прогноз для новой точки данных получают следующим образом: сначала выясняют, в какой области разбиения пространства признаков находится данная точка, а затем определяют класс, к которому относится большинство точек в этой области (либо единственный класс в области, если лист является чистым). Область может быть найдена с помощью обхода дерева, начиная с корневого узла и путем перемещения влево или вправо, в зависимости от того, выполняется ли тест или нет.

Кроме того, можно использовать деревья для решения задач регрессии, используя точно такой же подход. Для получения прогноза мы обходим дерево на основе тестов в каждом узле и находим лист, в который попадает новая точка данных. Выходом для этой точки данных будет значение целевой переменной, усредненное по всем обучающим точкам в этом листе.

Контроль сложности деревьев

Как правило, построение дерева, описанное здесь и продолжающееся до тех пор, пока все листья не станут чистыми, приводит к получению моделей, которые являются очень сложными и характеризуются сильным переобучением на обучающих данных. Наличие чистых листьев означает, что дерево имеет 100%-ную правильность на обучающей выборке. Каждая точка обучающего набора находится в листе, который имеет правильный мажоритарный класс. Переобучение можно увидеть в левой части **рис. 8.5**. Видно, что точки, определяемые как точки класса 1, находятся посреди точек, принадлежащих к классу 0. С другой стороны, мы видим ряд точек, спрогнозированных как класс 1, вокруг точки, отнесенной к классу 0. Это не та граница принятия решений, которую мы могли бы себе представить. Здесь граница принятия решений

фокусируется больше на отдельных точках-выбросах, которые находятся слишком далеко от остальных точек данного класса.

Есть две общераспространенные стратегии, позволяющие предотвратить переобучение. Первая стратегия – ранняя остановка построения дерева, называемая предварительной обрезкой (**pre-pruning**). Вторая стратегия – построение дерева с последующим удалением или сокращением малоинформативных узлов, называемое пост-обрезкой (**post-pruning**) или просто обрезкой (**pruning**). Возможные критерии предварительной обрезки включают в себя ограничение максимальной глубины дерева, ограничение максимального количества листьев или минимальное количество наблюдений в узле, необходимое для разбиения. В библиотеке scikit-learn деревья решений реализованы в классах **DecisionTreeRegressor** и **DecisionTreeClassifier**. Обратите внимание, в *scikit-learn реализована лишь предварительная обрезка.*

Давайте более детально посмотрим, как работает предварительная обрезка на примере набора данных Breast Cancer. Как всегда, мы импортируем набор данных и разбиваем его на обучающую и тестовую части. Затем мы строим модель, используя настройки по умолчанию для построения полного дерева (выращиваем дерево до тех пор, пока все листья не станут чистыми). Зафиксируем **random_state** для воспроизводимости результатов:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=42)
tree = DecisionTreeClassifier(random_state=0)
tree.fit(X_train, y_train)
print(
    "Правильность на обучающем наборе: {:.3f}"
    .format(tree.score(X_train, y_train))
)
print(
    "Правильность на тестовом наборе: {:.3f}"
    .format(tree.score(X_test, y_test))
)
```

```
Правильность на обучающем наборе: 1.000
Правильность на тестовом наборе: 0.937
```

Как и следовало ожидать, правильность на обучающем наборе составляет 100%, поскольку листья являются чистыми. Дерево имеет

глубину, как раз достаточную для того, чтобы прекрасно запомнить все метки обучающих данных. Правильность на тестовом наборе немного хуже, чем при использовании ранее рассмотренных линейных моделей, правильность которых составляла около 95%.

Если не ограничить глубину, дерево может быть сколь угодно глубоким и сложным. Поэтому необрезанные деревья склонны к переобучению и плохо обобщают результат на новые данные. Теперь давайте применим к дереву предварительную обрезку, которая остановит процесс

построения дерева до того, как мы идеально подгоним модель к обучающим данным. Один из вариантов – остановка процесса построения дерева по достижении определенной глубины. Здесь мы установим **max_depth=4**, то есть можно задать только четыре последовательных вопроса (**рис.8.3** и **8.5**). Ограничение глубины дерева уменьшает переобучение. Это приводит к более низкой правильности на обучающем наборе, но улучшает правильность на тестовом наборе:

```
tree = DecisionTreeClassifier(max_depth=4, random_state=0)
tree.fit(X_train, y_train)

print(
    "Правильность на обучающем наборе: {:.3f}"
    .format(tree.score(X_train, y_train))
)
print(
    "Правильность на тестовом наборе: {:.3f}"
    .format(tree.score(X_test, y_test))
)
```

```
Правильность на обучающем наборе: 0.988
Правильность на тестовом наборе: 0.951
```

Анализ деревьев решений

Мы можем визуализировать дерево, используя функцию **export_graphviz** из модуля **tree**. Она записывает файл в формате **.dot**, который является форматом текстового файла, предназначенным для описания графиков. Мы можем задать цвет узлам, чтобы выделить класс, набравший большинство в каждом узле, и передать имена классов и признаков, чтобы дерево было правильно размечено:

```
from sklearn.tree import export_graphviz
export_graphviz(
    tree,
    out_file="tree.dot",
    class_names=["malignant", "benign"],
    feature_names=cancer.feature_names,
    impurity=False,
    filled=True
)
```

Мы можем прочитать этот файл и визуализировать его (**рис. 8.6)**, используя модуль **graphviz15** (или любую другую программу, которая может читать файлы с расширением **.dot**):

```
import graphviz
with open("tree.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```

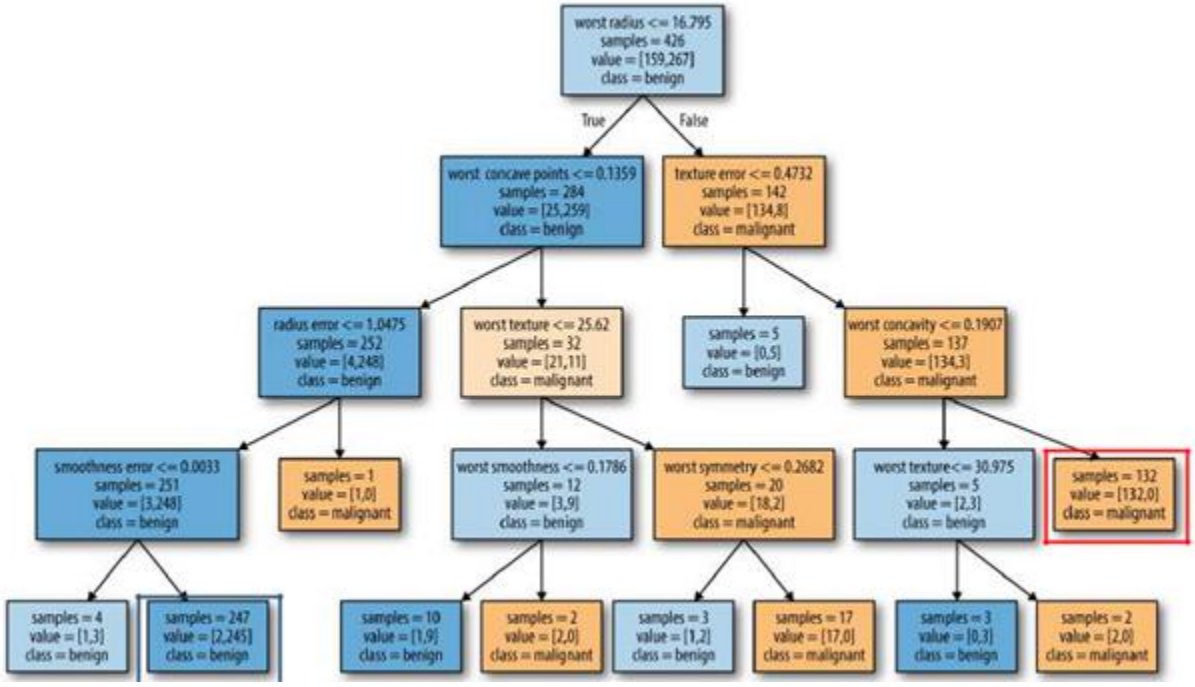

Рис. 8.6 Визуализация дерева решений, построенного на наборе данных Breast Cancer

Как вариант, можно построить диаграмму дерева и записать ее в файл **.pdf**. Дополнительно нам потребуется модуль **pydotplus**.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import mglearn
%matplotlib inline

from sklearn.model_selection import train_test_split from sklearn.datasets impo
rt load_breast_cancer from sklearn import tree
from sklearn.tree import export_graphviz
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
cancer.data, cancer.target, stratify=cancer.target, random_state=42)
clf = tree.DecisionTreeClassifier(max_depth=4, random_state=0)
clf = clf.fit(X_train, y_train)

import pydotplus
dot_data = tree.export_graphviz(clf, out_file=None)
```

pip install graphviz

Затем в переменной окружения ***PATH*** необходимо прописать полный путь к установленной папке **graphviz**. В Windows 7 для этого нажмите кнопку Пуск, выберите Панель управления. Дважды нажмите на Система, затем выберите Дополнительные параметры системы. Во вкладке Дополнительно нажмите на Переменные среды. Выберите **Path** и нажмите на Изменить. В поле Значение переменной введите путь к папке **graphviz** (пример, C:\Anaconda3\Library\bin\graphviz).

```
graph = pydotplus.graph_from_dot_data(dot_data)
graph.write_pdf("cancer.pdf")
```

Можно построить визуализацию дерева с помощью функции Image интерактивной оболочки **IPython**:

```
from IPython.display import Image
dot_data = tree.export_graphviz(clf, out_file=None,
feature_names=cancer.feature_names,
class_names=cancer.target_names,
filled=True, rounded=True,
special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

Визуализация дерева дает более глубокое представление о том, как алгоритм делает прогнозы и является хорошим примером алгоритма машинного обучения, который легко объяснить неспециалистам. Однако, как показано здесь, даже при глубине 4 дерево может стать немного громоздким. Деревья с большим значением глубины (деревья глубиной 10 – не редкость) еще труднее понять. Один из полезных способов исследования дерева заключается в том, чтобы выяснить, какие узлы содержат наибольшее количество данных. Параметр **samples**, выводимый в каждом узле на **рис. 8.6**, показывает общее количество примеров в узле, тогда как параметр **value** показывает количество примеров в каждом классе. Проследовав по правой ветви, отходящей от корневого узла, мы видим, что правилу worst radius > 16.795 соответствует узел, который содержит 134 случая злокачественной опухоли и лишь 8 случаев доброкачественной опухоли. Далее дерево выполняет серию более точных разбиений оставшихся 142 случаев. Из 142 случаев, которые при первоначальном разбиении были записаны в правый узел, почти все в конечном итоге попали в правый лист (для удобства выделен красной рамкой).

Проследовав по левой ветви, отходящей от корневого узла, мы видим, что правилу worst radius <= 16.795 соответствует узел, который содержит 25 случаев злокачественной опухоли и 259 случаев доброкачественной опухоли. Почти все случаи доброкачественной опухоли попадают во второй лист справа (для удобства выделен синей рамкой), остальные случаи распределяются по нескольким листьям, содержащим очень мало наблюдений.

Важность признаков в деревьях

Вместо того, чтобы просматривать все дерево, что может быть обременительно, есть некоторые полезные параметры, которые мы можем использовать как итоговые показатели работы дерева. Наиболее часто используемым показателем является важность признаков (**feature importance**), которая оценивает, насколько важен каждый признак с точки зрения получения решений. Это число варьирует в диапазоне от 0 до 1 для каждого признака, где 0 означает «не используется вообще», а 1 означает, что «отлично предсказывает целевую переменную». Важности признаков в сумме всегда дают 1:

```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier().fit(X_train, y_train)
print("Важности признаков:\n{}".format(tree.feature_importances_))
```

Приведенная сводка не совсем удобна, поскольку мы не знаем, каким именно признакам соответствуют приведенные важности. Чтобы исправить это, воспользуемся программным кодом, приведенным ниже:

```
for name, score in zip(cancer["feature_names"], tree.feature_importances_):
    print(name, score)
```

```
mean radius 0.0
mean texture 0.0
mean perimeter 0.0
mean area 0.0
mean smoothness 0.0
mean compactness 0.0
mean concavity 0.0
mean concave points 0.0
mean symmetry 0.0
mean fractal dimension 0.0
radius error 0.0101973682021
texture error 0.0483982536186
perimeter error 0.0
area error 0.0
smoothness error 0.00241559508532
compactness error 0.0
concavity error 0.0
concave points error 0.0
symmetry error 0.0
fractal dimension error 0.0
worst radius 0.72682850946
worst texture 0.0458158970889
worst perimeter 0.0
worst area 0.0
worst smoothness 0.0141577021047
worst compactness 0.0
worst concavity 0.0181879968645
worst concave points 0.122113199265
worst symmetry 0.0118854783101
worst fractal dimension 0.0
```

Мы можем визуализировать важности признаков аналогично тому, как мы визуализируем коэффициенты линейной модели (**рис. 8.7**):

```
def plot_feature_cancer(model):
    n_features = cancer.data.shape[1]
    plt.barh(range(n_features), model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), cancer.feature_names)
    plt.xlabel("Важность признака")
    plt.ylabel("Признак")
    plt.show()

plot_feature_cancer(tree)
```

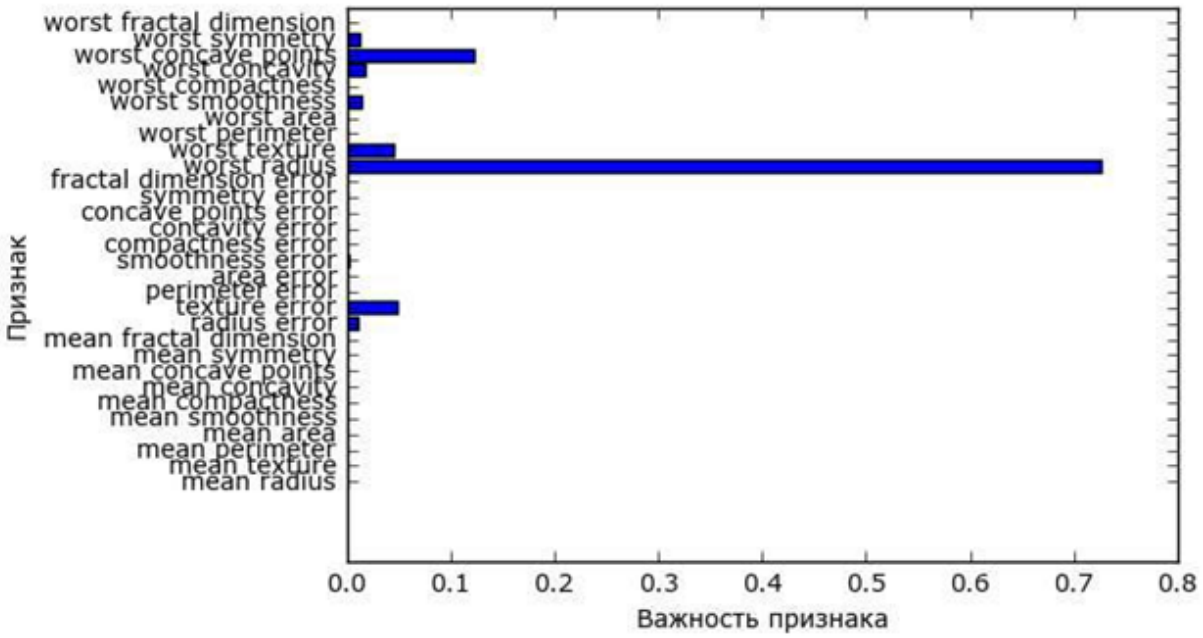

Рис. 8.7 Важности признаков, вычисленные с помощью дерева решений для набора данных Breast Cancer

Здесь мы видим, что признак, использованный в самом верхнем разбиении (**worst radius**), на данный момент является наиболее важным. Это подтверждает наш вывод о том, что уже на первом уровне два класса достаточно хорошо разделены.

Однако, если признак имеет низкое значение **feature_importance_**, это не значит, что он неинформативен. Это означает только то, что данный признак не был выбран деревом, поскольку, вероятно, другой признак содержит ту же самую информацию. отличие от коэффициентов линейных моделей важности признаков всегда положительны и они не указывают на взаимосвязь с каким-то конкретным классом. Важности признаков говорят нам, что **worst radius** важен, но мы не знаем, является ли высокое значение радиуса признаком доброкачественной или злокачественной опухоли. На самом деле, найти такую очевидную взаимосвязь между признаками и классом невозможно, что можно проиллюстрировать на следующем примере (**рис. 8.8** и **8.9**):

```
tree = mglearn.plots.plot_tree_not_monotone()
plt.show()
from IPython.display import display
display(tree)
```

```
Feature importances: [ 0. 1.]
```

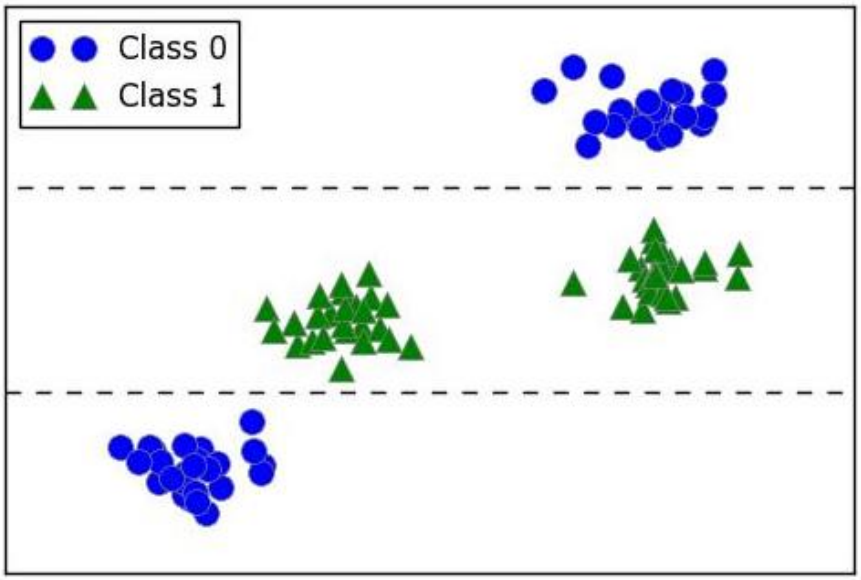

Рис. 8.8 Двумерный массив данных, в котором признак имеет немонотонную взаимосвязь с меткой класса, и границы принятия решений, найденные с помощью дерева

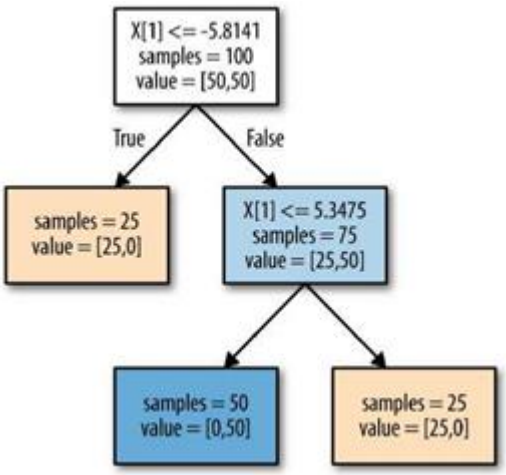

Рис. 8.9 Дерево решений для набора данных, показанном на **рис. 8.8**

График показывает набор данных с двумя признаками и двумя классами. Здесь вся информация содержится в X[1], а X[0] не используется вообще. Но взаимосвязь между X[1] и целевым классом не является монотонной, то есть мы не можем сказать, что «высокое значение X[0] означает класс 0, а низкое значение означает класс 1» (или наоборот).

Несмотря на то что мы сосредоточились здесь на деревьях классификации, все вышесказанное верно и для деревьев регрессии, которые реализованы в **DecisionTreeRegressor**. Применение и анализ деревьев регрессии очень схожи с применением и анализом деревьев классификации. Однако существует одна особенность использования деревьев регрессии, на которую нужно указать. DecisionTreeRegressor (и все остальные регрессионные модели на основе дерева) не умеет экстраполировать или делать прогнозы вне диапазона значений обучающих данных.

Давайте детальнее рассмотрим это, воспользовавшись набором данных RAM Price (содержит исторические данные о ценах на компьютерную память). **Рис. 8.10** визуализирует этот набор данных16, дата отложена по оси x, а цена одного мегабайта оперативной памяти в соответствующем году – по оси y:

```
import pandas as pd
ram_prices = pd.read_csv("D:/Subject/АД/Lab/Lab7/ram_price.csv")

plt.semilogy(ram_prices.date, ram_prices.price)
plt.xlabel("Год")
plt.ylabel("Цена $/Мбайт")
plt.show()
```

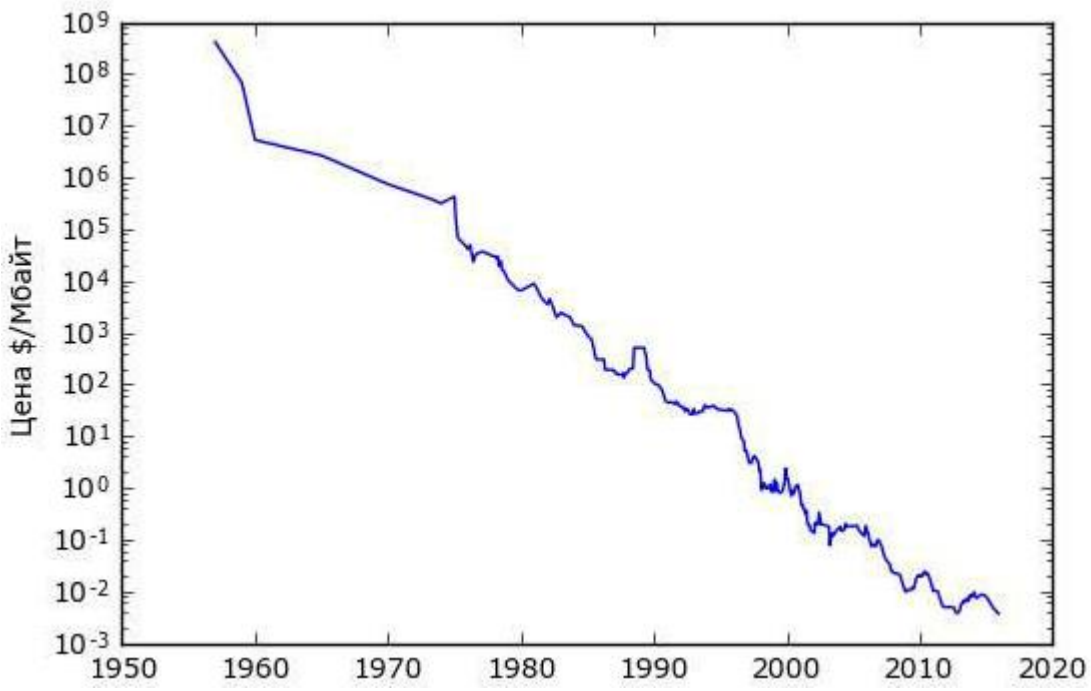

Рис. 8.10 Историческое развитие цен на RAM по логарифмической шкале

Обратите внимание на логарифмическую шкалу оси y. При логарифмическом преобразовании взаимосвязь выглядит вполне линейной и таким образом становится легко прогнозируемой, за исключением некоторых всплесков.

Мы будем прогнозировать цены на период после 2000 года, используя исторические данные до этого момента, единственным признаком будут даты. Мы сравним две простые модели: **DecisionTreeRegressor** и **LinearRegression**. Мы отмасштабируем цены, используя логарифм, таким образом, взаимосвязь будет относительно линейной. Это несущественно для **DecisionTreeRegressor**, однако существенно для **LinearRegression**. После обучения модели и получения прогнозов мы применим экспоненцирование, чтобы обратить логарифмическое преобразование. Мы получим и визуализируем прогнозы для всего набора данных, но для количественной оценки мы будем рассматривать только тестовый набор:

```
from sklearn.tree import DecisionTreeRegressor
data_train = ram_prices[ram_prices.date < 2000]
data_test = ram_prices[ram_prices.date >= 2000]
y_train = np.log(data_train.price)
X_train = data_train.date[:, np.newaxis]

print("X:\n{}".format(X_train))
print("y:\n{}".format(y_train))

tree = DecisionTreeRegressor().fit(X_train, y_train)
from sklearn.linear_model import LinearRegression
linear_reg = LinearRegression().fit(X_train, y_train)

X_all = ram_prices.date[:, np.newaxis]

pred_tree = tree.predict(X_all)
pred_lr = linear_reg.predict(X_all)

price_tree = np.exp(pred_tree)
price_lr = np.exp(pred_lr)
```

Рис. 8.11, созданный здесь, сравнивает прогнозы дерева решений и линейной регрессии с реальными.

```
plt.semilogy(data_train.date, data_train.price, label="Обучающие данные")
plt.semilogy(data_test.date, data_test.price, label="Тестовые данные")
plt.semilogy(ram_prices.date, price_tree, label="Прогнозы дерева")
plt.semilogy(ram_prices.date, price_lr, label="Прогнозы линейной регрессии")
plt.legend()
plt.show()
```

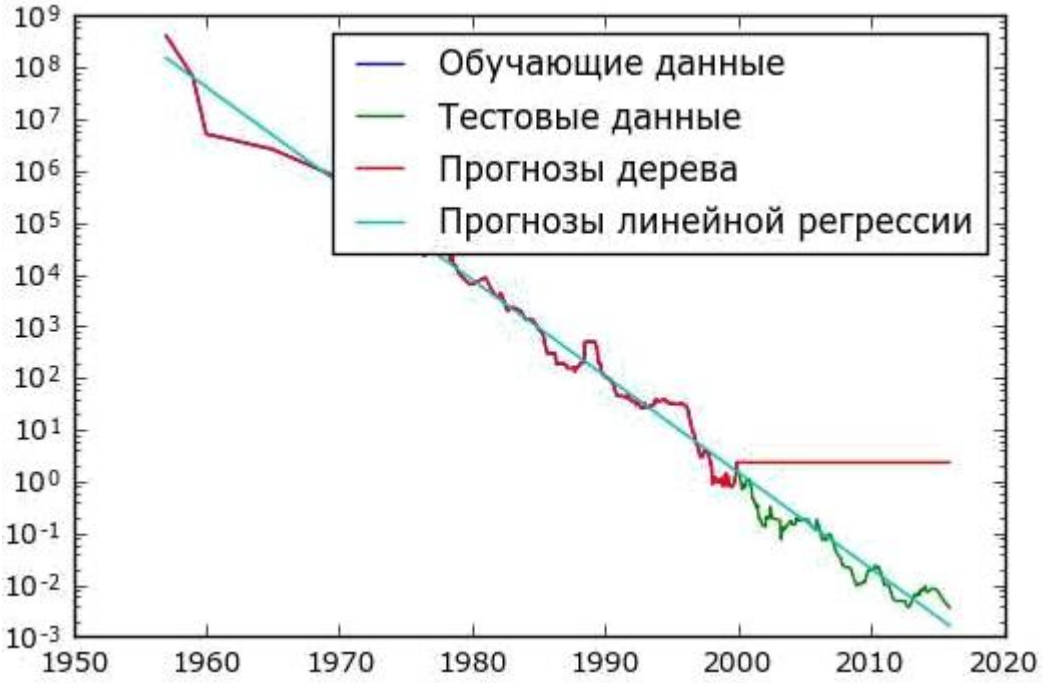

Рис. 8.11 Сравнение прогнозов линейной модели и прогнозов дерева регрессии для набора данных RAM price

Разница между моделями получилась весьма впечатляющая. Линейная модель аппроксимирует данные с помощью уже известной нам прямой линии. Эта линия дает достаточно хороший прогноз для тестовых данных (период после 2000 года), при этом сглаживая некоторые всплески в обучающих и тестовых данных. С другой стороны, модель дерева прекрасно прогнозирует на обучающих данных. Здесь мы не ограничивали сложность дерева, поэтому она полностью запомнила весь набор данных. Однако, как только мы выходим из диапазона значений, известных модели, модель просто продолжает предсказывать последнюю известную точку. Дерево не способно генерировать «новые» ответы, выходящие за пределы значений обучающих данных. Этот недостаток относится ко всем моделям на основе деревьев решений.

Преимущества, недостатки и параметры

Как уже говорилось выше, параметры, которые контролируют сложность модели в деревьях решений – это параметрами предварительной обрезки дерева, которые останавливают построение дерева, прежде чем оно достигнет максимального размера. Обычно, чтобы предотвратить переобучение, достаточно выбрать одну из стратегий предварительной обрезки – настроить **max_depth**, **max_leaf_nodes** или **min_samples_leaf**.

По сравнению со многими алгоритмами, обсуждавшимися до сих пор, деревья решений обладают двумя преимуществами: полученная модель может быть легко визуализирована и понята неспециалистами (по крайней мере это верно для небольших деревьев) и деревья не требуют масштабирования данных. Поскольку каждый признак обрабатывается отдельно, а возможные разбиения данных не зависят от масштабирования, алгоритмы деревьев решений не нуждаются в таких процедурах предварительной обработки, как нормализация или стандартизация признаков. Деревья решений хорошо работают, когда у вас есть признаки, измеренные в совершенно разных шкалах, или когда ваши данные представляют смесь бинарных и непрерывных признаков.

Основным недостатком деревьев решений является то, что даже при использовании предварительной обрезки, они склонны к переобучению имеют низкую обобщающую способность. Поэтому в большинстве случаев, как правило, вместо одиночного дерева решений используются ансамбли деревьев, которые мы обсудим далее.

Код к лабораторной работе:

```
import mglearn
import sklearn
import matplotlib.pyplot as plt
import numpy as np

X = np.array([[0, 1, 0, 1], [1, 0, 1, 1], [0, 0, 0, 1], [1, 0, 1, 0]])

y = np.array([0, 1, 0, 1])

counts = {}
for label in np.unique(y):
    counts[label] = X[y == label].sum(axis=0)
print("Частоты признаков:\n{}".format(counts))

mglearn.plots.plot_animal_tree()
plt.show()

mglearn.plots.plot_tree_progressive()
plt.show()


from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=42)
tree = DecisionTreeClassifier(random_state=0)
tree.fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.3f}".format(tree.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.3f}".format(tree.score(X_test, y_test)))


tree = DecisionTreeClassifier(max_depth=4, random_state=0)
tree.fit(X_train, y_train)

print("Правильность на обучающем наборе: {:.3f}".format(tree.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.3f}".format(tree.score(X_test, y_test)))


from sklearn.tree import export_graphviz
export_graphviz(tree, out_file="tree.dot", class_names=["malignant", "benign"],
    feature_names=cancer.feature_names, impurity=False, filled=True)


import graphviz
with open("tree.dot") as f:
```

```
    dot_graph = f.read()
graphviz.Source(dot_graph)

"Export as pdf"
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
from sklearn import tree
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=42)
clf = tree.DecisionTreeClassifier(max_depth=4, random_state=0)
clf = clf.fit(X_train, y_train)

import pydotplus
dot_data = tree.export_graphviz(clf, out_file=None)
graph = pydotplus.graph_from_dot_data(dot_data)
graph.write_pdf("cancer.pdf")


"Create tree with using IPython"
from IPython.display import Image
dot_data = tree.export_graphviz(clf, out_file=None,
    feature_names=cancer.feature_names,
    class_names=cancer.target_names,
    filled=True, rounded=True,
    special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())

from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier().fit(X_train, y_train)
print("Важности признаков:\n{}".format(tree.feature_importances_))


def plot_feature_cancer(model):
    n_features = cancer.data.shape[1]
    plt.barh(range(n_features), model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), cancer.feature_names)
    plt.xlabel("Важность признака")
    plt.ylabel("Признак")
    plt.show()


plot_feature_cancer(tree)


tree = mglearn.plots.plot_tree_not_monotone()
plt.show()
from IPython.display import display
```

```
display(tree)


import pandas as pd
ram_prices = pd.read_csv("D:/Subject/АД/Lab/Lab7/ram_price.csv")

plt.semilogy(ram_prices.date, ram_prices.price)
plt.xlabel("Год")
plt.ylabel("Цена $/Мбайт")
plt.show()


from sklearn.tree import DecisionTreeRegressor
data_train = ram_prices[ram_prices.date < 2000]
data_test = ram_prices[ram_prices.date >= 2000]
y_train = np.log(data_train.price)
X_train = data_train.date[:, np.newaxis]

print("X:\n{}".format(X_train))
print("y:\n{}".format(y_train))

tree = DecisionTreeRegressor().fit(X_train, y_train)
from sklearn.linear_model import LinearRegression
linear_reg = LinearRegression().fit(X_train, y_train)

X_all = ram_prices.date[:, np.newaxis]

pred_tree = tree.predict(X_all)
pred_lr = linear_reg.predict(X_all)

price_tree = np.exp(pred_tree)
price_lr = np.exp(pred_lr)


plt.semilogy(data_train.date, data_train.price, label="Обучающие данные")
plt.semilogy(data_test.date, data_test.price, label="Тестовые данные")
plt.semilogy(ram_prices.date, price_tree, label="Прогнозы дерева")
plt.semilogy(ram_prices.date, price_lr, label="Прогнозы линейной регрессии")
plt.legend()
plt.show()
```