Лабораторная работа №3

Классификация и регрессия

Есть два основные задачимашинного обучения с учителем: ***классификация*** *(classification)* и ***регрессия*** *(regression)*.

Цель классификации состоит в том, чтобы спрогнозировать ***метку класса*** *(class label)*, которая представляет собой выбор из заранее определенного списка возможных вариантов. В главе 1 мы использовали пример классификации ирисов, когда относили цветок к одному из трех возможных сортов. Классификация иногда разделяется на ***бинарную классификацию*** *(binary classification)*, которая является частным случаем разделения на два класса, и ***мультиклассовую классификацию*** *(multiclass classification)*, когда в классификации участвует более двух классов. Бинарную классификацию можно представить как попытку ответить на поставленный вопрос в формате «да/нет». Классификация электронных писем на спам и не-спам является примером бинарной классификации. Данной задаче бинарной классификации ответ «да/нет» дается на вопрос «является ли это электронное письмо спамом?»

С другой стороны, пример классификации ирисов является примером мультиклассовой классификации. Еще один пример – прогнозирование языка веб-сайта. Классами здесь будет заранее определенный список возможных языков.

Цель регрессии состоит в том, чтобы спрогнозировать непрерывное число или ***число с плавающей точкой*** *(floating-point number)*, если использовать термины программирования, или ***вещественное число*** *(real number)*, если говорить языком математических терминов. Прогнозирование годового дохода человека в зависимости от его образования, возраста и места жительства является примером регрессионной задачи. Прогнозируемое значение дохода представляет собой ***сумму*** *(amount)* и может быть любым числом в заданном диапазоне. Другой пример регрессионной задачи – прогнозирование объема урожая зерна на ферме в зависимости от таких атрибутов, как объем предыдущего урожая, погода, и количество сотрудников, работающих на ферме. И снова объем урожая может быть любым числом.

Самый простой способ отличить классификацию от регрессии – спросить, заложена ли в полученном ответе определенная непрерывность (преемственность). Если полученные результаты непрерывно связаны друг с другом, то решаемая задача является задачей регрессии. Возьмем прогнозирование годового дохода. Здесь ясно видна непрерывность ответа. Разница между годовым доходом в 40000$ или 40001$ не существенна, хотя речь идет о разных денежных суммах. Если наш алгоритм предсказывает 39999$ или 40001$, в то время как он должен предсказать 40000$ (реальное значение годового дохода), мы не будем настаивать на том, что разница существенна. Наоборот, в задаче распознавании языка веб-сайта (задаче классификации) ответы четко определены. Контент сайта может быть написан либо на одном конкретном языке, либо на другом. Между языками нет непрерывной связи, не существует языка, находящегося между английским и французским

Обобщающая способность, переобучение и недообучение

В машинном обучении с учителем нам нужно построить модель на обучающих данных, а затем получить точные прогнозы для новых, еще не встречавшихся нам данных, которые имеют те же самые характеристики, что и использованный нами обучающий набор. Если модель может выдавать точные прогнозы на ранее не встречавшихся данных, мы говорим, что модель обладает способностью ***обобщать*** *(generalize)* результат на тестовые данные. Нам необходимо построить модель, которая будет обладать максимальной обобщающей способностью.

Обычно мы строим модель таким образом, чтобы она давала точные прогнозы на обучающем наборе. Если обучающий и тестовый наборы имеют много общего между собой, можно ожидать, что модель будет точной и на тестовом наборе. Однако в некоторых случаях этого не происходит. Например, если мы строим очень сложные модели, необходимо помнить, что на обучающей выборке можно получить произвольную правильность.

Давайте взглянем на выдуманный пример, чтобы проиллюстрировать этот тезис. Скажем, начинающий специалист по анализу данных хочет спрогнозировать покупку клиентом лодки на основе записей о клиентах, которые ранее приобрели лодку, и клиентах, которые не заинтересованы в покупке лодки. 7 Цель состоит в том, чтобы отправить рекламные письма клиентам, которые, вероятно, хотят совершить покупку, и не беспокоить клиентов, не заинтересованных в покупке.

Предположим, есть записи о клиентах, приведенные в таблице 2.1.

Возраст	Кол-во автомобилей в собственности	Есть собственный дом	Кол-во детей	Семейное положение	Есть собака	Купил лодку
66	1	Да	2	Вдовец	нет	Да
52	2	Да	3	Женат	Нет	Да
22	0	Нет	0	Женат	Да	Нет
25	1	Нет	1	Холост	Нет	Нет
44	0	Нет	2	Разведен	Да	Нет
39	1	Да	2	Женат	Да	Нет
26	1	нет	2	Холост	Нет	нет

Таблица 2.1 Пример данных о клиентах

Поработав с данными некоторое время, наш начинающий специалист формулирует следующее правило «если клиент старше 45 лет, у него менее трех детей, либо у него трое и он женат, то он скорее всего купит лодку». Если спросить, насколько хорошо работает это правило, наш специалист воскликнет «оно дает 100%-ную правильность!» И в самом деле, для данных, приведенных в таблице, это правило демонстрирует идеальную правильность. Мы могли бы сформулировать массу правил, объясняющих покупку лодки. Значения возраста появляются в наборе данных лишь один раз, таким образом, мы могли бы сказать, что люди в возрасте 66, 52, 53 и 58 лет хотят купить лодку, тогда как все остальные не собираются ее покупать. Несмотря на то что можно сформулировать множество правил, которые хорошо работают для этих наблюдений, следует помнить о том, что нам не интересны прогнозы для этого набора

данных, мы уже знаем ответы для этих клиентов. Мы хотим знать, могут ли новые клиенты купить лодку. Поэтому нам нужно правило, которое будет хорошо работать для новых клиентов, и достижение 100%-ной правильности на обучающей выборке не поможет нам в этом. Нельзя ожидать, что правило, сформулированное нашим специалистом, будет так же хорошо работать и для новых клиентов. Похоже, с этим у нас сложность, ведь у нас мало данных. Например, часть правила «либо трое и он женат» сформулировано по одному клиенту.

Единственный показатель качества работы алгоритма на новых данных – это использование тестового набора. Однако интуитивно[8] мы ожидаем, что простые модели должны лучше обобщать результат на новые данные. Если бы правило звучало «люди старше 50 лет хотят купить лодку» и оно объясняло бы поведение всех клиентов, мы доверяли бы ему больше, чем правилу, которое помимо возраста включало бы количество детей и семейное положение. Поэтому нам всегда нужно искать самую простую модель. Построение модели, которая слишком сложна для имеющегося у нас объема информации (что и сделал наш начинающий специалист по анализу данных), называется ***переобучением*** *(overfitting)*. Переобучение происходит, когда ваша модель слишком точно подстраивается под особенности обучающего набора и вы получаете модель, которая хорошо работает на обучающем наборе, но не умеет обобщать результат на новые данные. С другой стороны, если ваша модель слишком проста, скажем, вы сформулировали правило «все, у кого есть собственный дом, покупает лодку», вы, возможно, не смогли охватить все многообразие и изменчивость данных, ваша модель будет плохо работать даже на обучающем наборе. Выбор слишком простой модели называется ***недообучением*** *(underfitting)*.

Чем сложнее модель, тем лучше она будет работать на обучающих данных. Однако, если наша модель становится слишком сложной, мы начинаем уделять слишком много внимания каждой отдельной точке данных в нашем обучающем наборе, и эта модель не будет хорошо обобщать результат на новые данные.

Существует оптимальная точка, которая позволяет получить наилучшую обобщающую способность. Собственно это и есть модель, которую нам нужно найти. Компромисс между переобучением и недообучением показан на рис. 2.1.

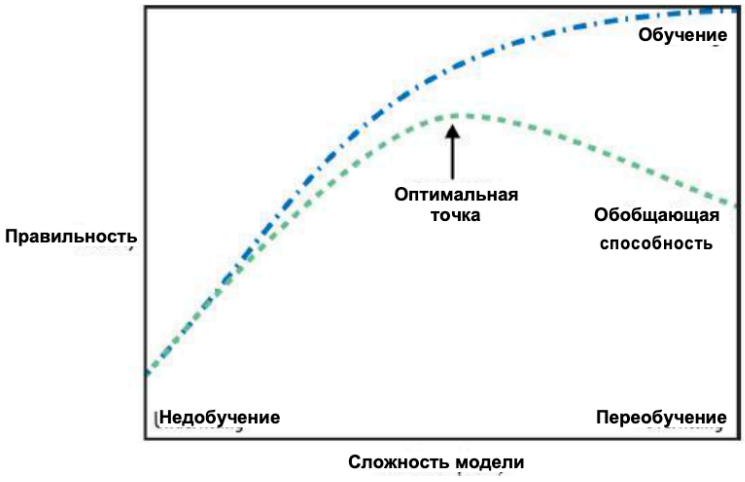

Рис. 2.1 Компромисс между сложностью модели и правильностью на обучающей и тестовой выборках

Взаимосвязь между сложностью модели и размером набора данных

Важно отметить, что сложность модели тесно связана с изменчивостью входных данных, содержащихся в вашем обучающем наборе: чем больше разнообразие точек данных в вашем наборе, тем более сложную модель можно использовать, не беспокоясь о переобучении. Обычно больший объем данных дает большее разнообразие, таким образом, большие наборы данных позволяют строить более сложные модели. Однако простое дублирование одних и тех же точек данных или сбор очень похожих данных здесь не поможет.

Возвращаясь к продажам лодок, можно сказать, что если бы у нас было более 10000 строк данных о клиентах и все они подчинялись бы правилу «если клиент старше 45 лет, у него менее трех детей, либо трое, он женат, то он скорее всего купит лодку», мы бы с гораздо большей вероятностью поверили в это правило, чем если бы оно было сформулировано лишь по 12 строкам таблицы 2.1.

Увеличение объема данных и построение более сложных моделей часто творят чудеса при решении задач машинного обучения с учителем. В этой книге мы сосредоточимся на работе с данными фиксированного размера. В действительности вы, как правило, сами можете определить объем собираемых данных, и это может оказаться более полезным, чем корректировка и настройка вашей модели. Никогда не стоит недооценивать преимущества увеличения объема данных.

Алгоритм машинного обучения с учителем

Теперь мы рассмотрим наиболее популярные алгоритмы машинного обучения и объясним, как они обучаются на основе данных и как вычисляют прогнозы. Кроме того, мы расскажем о том, как принцип сложности реализуется для каждой из этих моделей, и покажем, как тот или иной алгоритм строит модель. Мы рассмотрим преимущества и недостатки каждого алгоритма, а также расскажем о том, применительно каким данным лучше всего использовать тот или иной алгоритм. Мы также объясним значение наиболее важных параметров и опций. Многие алгоритмы имеют опции классификации и регрессии, поэтому мы опишем обе опции.

Необязательно детально вчитываться в описание каждого алгоритма, но понимание модели даст вам лучшее представление о различных способах работы алгоритмов машинного обучения. Кроме того, эту главу можно использовать в качестве справочного руководства, и вы можете вернуться к ней, если не знаете, как работает тот или иной алгоритм.

Некоторые наборы данных

Для иллюстрации различных алгоритмов мы будем использовать несколько наборов данных. Некоторые наборы данных будет небольшими и синтетическими (то есть выдуманными), призванными подчеркнуть отдельные аспекты алгоритмов. Другие наборы данных будут большими, реальными примерами.

Примером синтетического набора данных для двухклассовой классификации является набор данных forge, который содержит два признака. Программный код, приведенный ниже, создает диаграмму рассеяния (рис. 2.2), визуализируя все точки данных в этом наборе. На графике первый признак отложен на оси x, а второй – по оси y. Как это всегда бывает в диаграммах рассеяния, каждая точка данных представлена в виде одного маркера. Цвет и форма маркера указывает на класс, к которому принадлежит точка:

```
генерируем набор данных
строим график для набора данных
%matplotlib inline
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.legend(["Класс 0", "Класс 1"], loc=4)
plt.xlabel("Первый признак")
plt.ylabel("Второй признак")
print("форма массива X: {}".format(X.shape))
```

```
форма массива X: (26, 2)
```

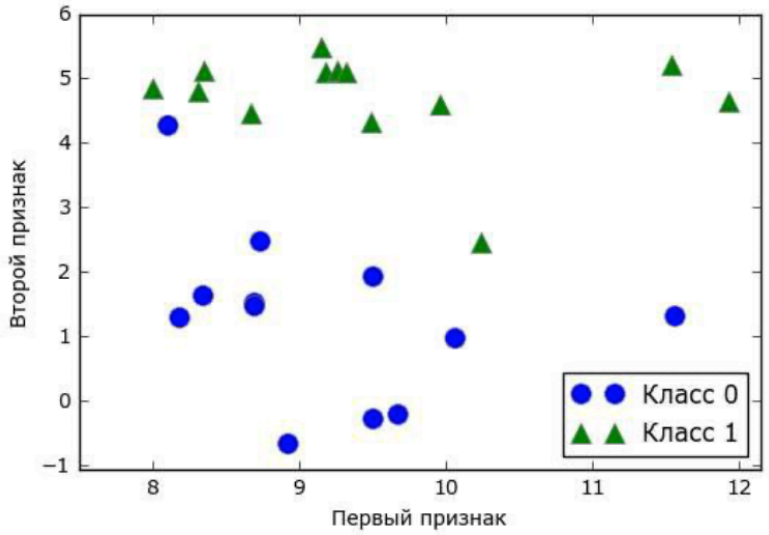

Рис. 2.2 Диаграмма рассеяния для набора данных forge

Как видно из сводки по массиву X, этот набор состоит из 26 точек данных и 2 признаков. Для иллюстрации алгоритмов регрессии, мы воспользуемся синтетическим набором *wave*. Набор данных имеет единственный входной признак и непрерывную

целевую переменную или ***отклик*** *(response)*, который мы хотим смоделировать. На рисунке, построенном здесь (рис. 2.3), по оси x располагается единственный признак, а по оси y – целевая переменная (ответ).

```
X, y = mglearn.datasets.make_wave(n_samples=40)
plt.plot(X, y, 'o')
plt.ylim(-3, 3)
plt.xlabel("Признак")
plt.ylabel("Целевая переменная")
```

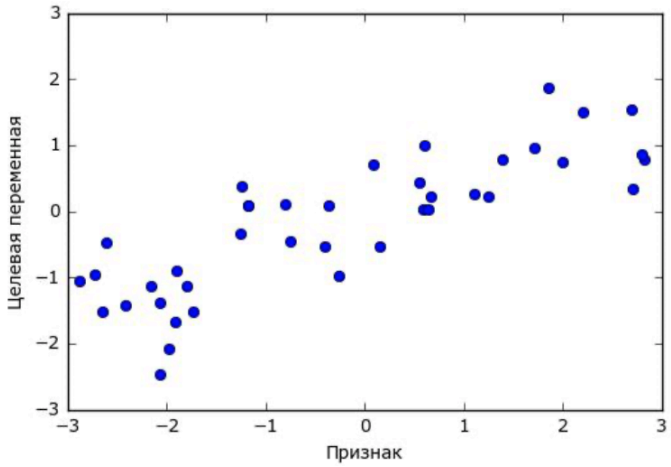

Рис. 2.3 График для набора данных wave, по оси x отложен признак, по оси y – целевая переменная

Мы используем эти очень простые, низкоразмерные наборы данных, потому что их легко визуализировать – печатная страница имеет два измерения, и данные, которые содержат более двух признаков, графически представить трудно. Вывод, полученный для набора с небольшим числом признаков или ***низкоразмерном*** *(low-dimensional)* наборе, возможно, не подтвердится для набора данных с большим количеством признаков или ***высокоразмерного*** *(high-dimensional)* набора. Если вы помните об этом, проверка алгоритма на низкоразмерном наборе данных может оказаться очень полезной.

Мы дополним эти небольшие синтетические наборы данных двумя реальными наборами, которые включены в scikit-learn. Один из них – набор данных по раку молочной железы Университета Висконсин (cancer для краткости), в котором записаны клинические измерения опухолей молочной железы. Каждая опухоль обозначается как «benign» («доброкачественная», для неагрессивных опухолей) или malignant («злокачественная», для раковых опухолей), и задача состоит в том, чтобы на основании измерений ткани дать прогноз, является ли опухоль злокачественной.

Данные можно загрузить из scikit-learn с помощью функции ***load_breast_cancer***:

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
print("Ключи cancer(): \n{}".format(cancer.keys()))

# Набор данных включает 569 точек данных и 30 признаков.
```

```
print("Форма массива data для набора cancer: {}".format(cancer.data.shape))

# Из 569 точек данных 212 помечены как злокачественные, а 357 как доброкачественные.
print("Количество примеров для каждого класса:\n{}".format(
{n: v for n, v in zip(cancer.target_names, np.bincount(cancer.target))}))

# Чтобы получить содержательное описание каждого признака, взглянем на атрибут
feature_names:
print("Имена признаков:\n{}".format(cancer.feature_names))
```

```
Ключи cancer():
dict_keys(['feature_names', 'data', 'DESCR', 'target', 'target_names'])

Форма массива data для набора cancer: (569, 30)

Количество примеров для каждого класса:
{'benign': 357, 'malignant': 212}

Имена признаков:
[
    'mean radius' 'mean texture' 'mean perimeter'
    'mean area' 'mean smoothness' 'mean compactness'
    'mean concavity' 'mean concave points' 'mean symmetry'
    'mean fractal dimension' 'radius error' 'texture error'
    'perimeter error' 'area error' 'smoothness error'
    'compactness error' 'concavity error' 'concave points error'
    'symmetry error' 'fractal dimension error' 'worst radius'
    'worst texture' 'worst perimeter' 'worst area'
    'worst smoothness' 'worst compactness' 'worst concavity'
    'worst concave points' 'worst symmetry' 'worst fractal dimension'
]
```

Если вам интересно, то более подробную информацию о данных можно получить, прочитав *cancer.DESCR*.

Кроме того, для задач регрессии мы будем использовать реальный набор данных – набор данных *Boston Housing*. Задача, связанная с этим набором данных, заключается в том, чтобы спрогнозировать медианную стоимость домов в нескольких районах Бостона в 1970-е годы на основе такой информации, как уровень преступности, близость к Charles River, удаленность от радиальных магистралей и т.д. Набор данных содержит 506 точек данных и 13 признаков:

```
from sklearn.datasets import load_boston
boston = load_boston()
print("форма массива data для набора boston: {}".format(boston.data.shape))
```

```
форма массива data для набора boston: (506, 13)
```

Опять же, вы можете получить более подробную информацию о наборе данных, прочитав атрибут *boston.DESCR*. В данном случае мы более детально проанализируем набор данных, учтя не только 13 измерений в качестве входных признаков, но и приняв во внимание все ***взаимодействия*** *(interactions)* между признаками. Иными словами, мы

будем учитывать в качестве признаков не только уровень преступности и удаленность от радиальных магистралей по отдельности, но и взаимодействие уровень преступности–удаленность от радиальных магистралей. Включение производных признаков называется ***конструированием признаков*** *(feature engineering)*, которое мы рассмотрим более подробно в главе 4. Набор данных с производными признаками можно загрузить с помощью функции *load_extended_boston*:

```
X, y = mglearn.datasets.load_extended_boston()
print("форма массива X: {}".format(X.shape))
```

```
форма массива X: (506, 104)
```

Полученные 104 признака – 13 исходных признаков плюс 91 производный признак.

Мы будем использовать эти наборы данных, чтобы объяснить и проиллюстрировать свойства различных алгоритмов машинного обучения. Однако сейчас давайте перейдем к самим алгоритмам. Во-первых, мы вернемся к алгоритму k ближайших соседей, который рассматривали ранее.

Метод k ближайших соседей

Алгоритм k ближайших соседей, возможно, является самым простым алгоритмом машинного обучения. Построение модели заключается в запоминании обучающего набора данных. Для того, чтобы сделать прогноз для новой точки данных, алгоритм находит ближайшие к ней точки обучающего набора, то есть находит «ближайших соседей».

Классификация с помощью k ближайших соседей

Простейшем варианте алгоритм k ближайших соседей рассматривает лишь одного ближайшего соседа – точку обучающего набора, ближе всего расположенную к точке, для которой мы хотим получить прогноз. Прогнозом является ответ, уже известный для данной точки обучающего набора. На рис. 2.4 показано решение задачи классификации для набора данных forge:

```
mglearn.plots.plot_knn_classification(n_neighbors=1)
```

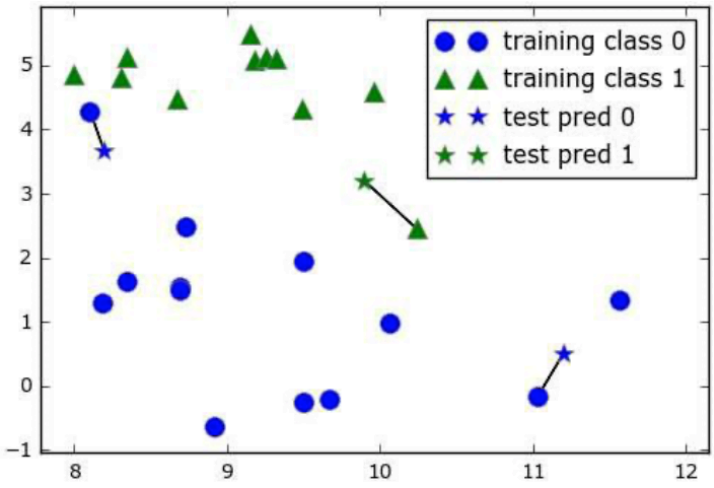

Рис. 2.4 Прогнозы, полученные для набора данных forge с помощью модели одного ближайшего соседа

Здесь мы добавили три новые точки данных, показанные в виде звездочек. Для каждой мы отметили ближайшую точку обучающего набора. Прогноз, который дает алгоритм одного ближайшего соседа – метка этой точки (показана цветом маркера).

Вместо того, чтобы учитывать лишь одного ближайшего соседа, мы можем рассмотреть произвольное количество (k) соседей. Отсюда и происходит название алгоритма k ближайших соседей. Когда мы рассматриваем более одного соседа, для присвоения метки используется ***голосование*** *(voting)*. Это означает, что для каждой точки тестового набора мы подсчитываем количество соседей, относящихся к классу 0, и количество соседей, относящихся к классу 1. Затем мы присваиваем точке тестового

набора наиболее часто встречающийся класс: другими словами, мы выбираем класс, набравший большинство среди k ближайших соседей. В примере, приведенном ниже (рис. 2.5), используются три ближайших соседа:

```
mglearn.plots.plot_knn_classification(n_neighbors=3)
```

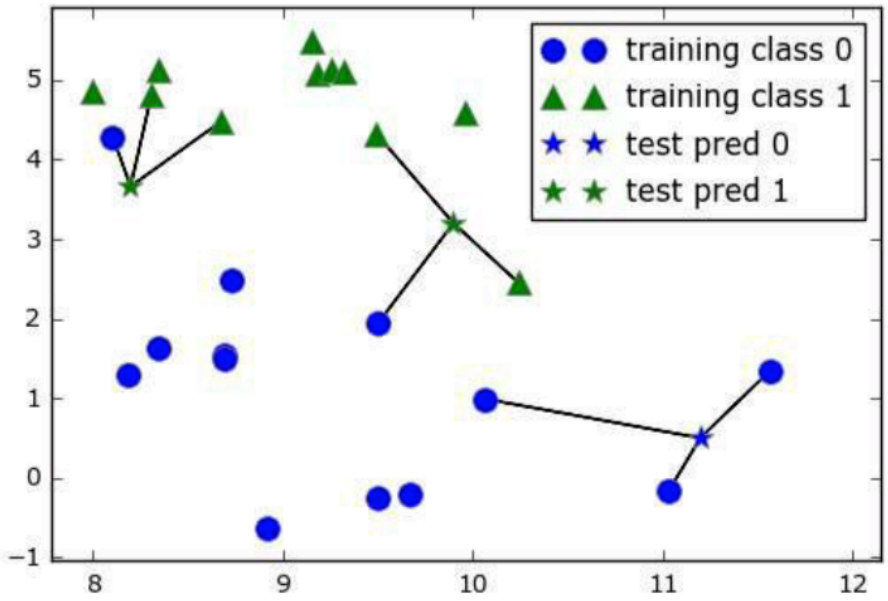

Рис. 2.5 Прогнозы, полученные для набора данных forge с помощью модели трех ближайшего соседа

Снова прогнозы переданы цветом маркера. Видно, что прогноз для новой точки данных в верхнем левом углу отличается от прогноза, полученного при использовании одного ближайшего соседа.

Хотя данный рисунок иллюстрирует задачу бинарной классификации, этот метод можно применить к наборам данных с любым количеством классов. В случае мультиклассовой классификации мы подсчитываем количество соседей, принадлежащих к каждому классу, и снова прогнозируем наиболее часто встречающийся класс.

Теперь давайте посмотрим, как можно применить алгоритм k ближайших соседей, используя scikit-learn. Во-первых, мы разделим наши данные на обучающий и тестовый наборы, чтобы оценить обобщающую способность модели, рассмотренную в главе 1:

```
from sklearn.model_selection import train_test_split X, y =
mglearn.datasets.make_forge()

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

Далее выполняем импорт и создаем объект-экземпляр класса, задавая параметры, например, количество соседей, которое будем использовать для классификации. В данном случае мы устанавливаем его равным 3:

```
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)
```

Затем подгоняем классификатор, используя обучающий набор. Для KNeighborsClassifier это означает запоминание набора данных, таким образом, мы можем вычислить соседей в ходе прогнозирования:

```
clf.fit(X_train, y_train)
```

Чтобы получить прогнозы для тестовых данных, мы вызываем метод predict. Для каждой точки тестового набора он вычисляет ее ближайших соседей в обучающем наборе и находит среди них наиболее часто встречающийся класс:

```
print("Прогнозы на тестовом наборе: {}".format(clf.predict(X_test)))
```

```
Прогнозы на тестовом наборе: [1 0 1 0 1 0 0]
```

Для оценки обобщающей способности модели мы вызываем метод score с тестовыми данными и тестовыми метками:

```
print("Правильность на тестовом наборе: {:.2f}".format(clf.score(X_test, y_test)))
```

```
Правильность на тестовом наборе: 0.86
```

Мы видим, что наша модель имеет правильность 86%, то есть модель правильно предсказала класс для 86% примеров тестового набора.

Анализ KNeughborsClassifier

Кроме того, для двумерных массивов данных мы можем показать прогнозы для всех возможных точек тестового набора, разместив в плоскости xy. Мы зададим цвет плоскости в соответствии с тем классом, который будет присвоен точке в этой области. Это позволит нам сформировать ***границу принятия решений*** *(decision boundary)*, которая разбивает плоскость на две области: область, где алгоритм присваивает класс 0, и область, где алгоритм присваивает класс 1.

Программный код, приведенный ниже, визуализирует границы принятия решений для одного, трех и девяти соседей (показаны на рис. 2.6):

```
fig, axes = plt.subplots(1, 3, figsize=(10, 3))

for n_neighbors, ax in zip([1, 3, 9], axes):
    # создаем объект-классификатор и подгоняем в одной строке
    clf = KNeighborsClassifier(n_neighbors=n_neighbors).fit(X, y)
    mglearn.plots.plot_2d_separator(clf, X, fill=True, eps=0.5, ax=ax, alpha=.4)
    mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)
    ax.set_title("количество соседей:{}".format(n_neighbors))
    ax.set_xlabel("признак 0")
    ax.set_ylabel("признак 1")
axes[0].legend(loc=3)
```

Рис. 2.6 Границы принятия решений, созданные моделью ближайших соседей для различных значений n_neighbors

На рисунке слева можно увидеть, что использование модели одного ближайшего соседа дает границу принятия решений, которая очень хорошо согласуется с обучающими данными. Увеличение числа соседей приводит к сглаживанию границы принятия решений. Более гладкая граница соответствует более простой модели. Другими словами, использование нескольких соседей соответствует высокой сложности модели (как показано в правой части рис. 2.1), а использование большого количества соседей соответствует низкой сложности модели (как показано в левой части рис. 2.1). Если взять крайний случай, когда количество соседей будет равно количеству точек данных

обучающего набора, каждая точка тестового набора будет иметь одних и тех же соседей (соседями будет все точки обучающего набора) и все прогнозы будут одинаковыми: будет выбран класс, который является наиболее часто встречающимся в обучающем наборе.

Давайте выясним, существует ли взаимосвязь между сложностью модели и обобщающей способностью, о которой мы говорили ранее. Мы сделаем это с помощью набора данных *Breast Cancer*. Начнем того, что разобьем данные на обучающий и тестовый наборы. Затем мы оценим качество работы модели на обучающем и тестовом наборах с использованием разного количества соседей. Рез.показаны на рис. 2.7:

```
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=66)

training_accuracy = []
test_accuracy = []

# пробуем n_neighbors от 1 до 10
neighbors_settings = range(1, 11)

for n_neighbors in neighbors_settings:
    # строим модель
    clf = KNeighborsClassifier(n_neighbors=n_neighbors)
    clf.fit(X_train, y_train)
    #записываем правильность на обучающем наборе
    training_accuracy.append(clf.score(X_train, y_train))
    #записываем правильность на тестовом наборе
    test_accuracy.append(clf.score(X_test, y_test))

plt.plot(neighbors_settings, training_accuracy, label="правильность на обучающем
наборе")
plt.plot(neighbors_settings, test_accuracy, label="правильность на тестовом наборе")
plt.ylabel("Правильность")
plt.xlabel("количество соседей")
plt.legend()
```

На этом графике по оси y отложена правильность на обучающем наборе и правильность на тестовом наборе, а по оси x – количество соседей. В реальности подобные графики редко бывают гладкими, мы по-прежнему можем увидеть некоторые признаки переобучения и недообучения (обратите внимание, что поскольку использование небольшого количества соседей соответствует более сложной модели, график представляет собой изображение рис. 2.1, зеркально отраженное по горизонтали). При использовании модели одного ближайшего соседа правильность на обучающем наборе идеальна. Однако при использовании большего количества соседей модель становится все проще и правильность на обучающем наборе падает. Правильность на тестовом наборе в случае использования одного соседа ниже, чем при использовании

нескольких соседей. Это указывает на то, что использование одного ближайшего соседа приводит к построению слишком сложной модели. С другой стороны, когда используются 10 соседей, модель становится слишком простой и она работает еще хуже. Оптимальное качество работы модели наблюдается где-то посередине, когда используются шесть соседей. Однако посмотрим на шкалу y. Худшая по качеству модель дает правильность на тестовом наборе около 88%, что по-прежнему может быть приемлемым результатом.

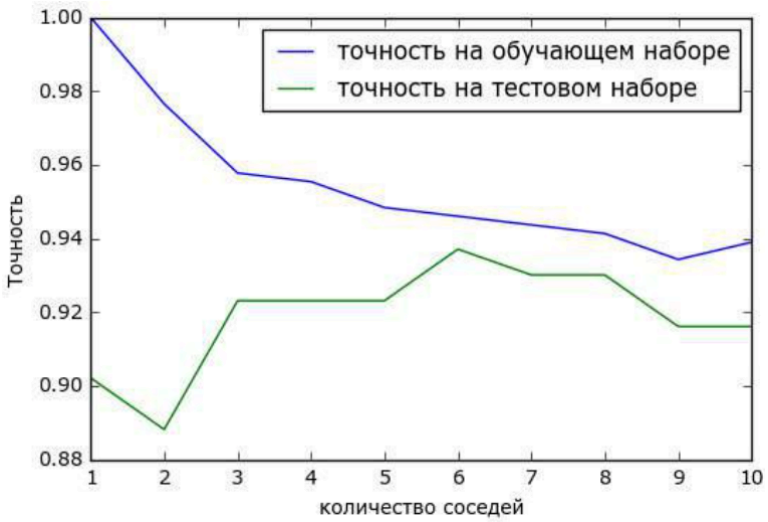

Рис. 2.7 Сравнение правильности на обучающем и тестовом наборах как функции от количества соседей

Регрессия k ближайших соседей

Существует также регрессионный вариант алгоритма k ближайших соседей. Опять же, давайте начнем с рассмотрения одного ближайшего соседа, на этот раз воспользуемся набором данных *wave*. Мы добавили три точки тестового набора в виде зеленых звездочек по оси x. Прогноз использованием одного соседа – это целевое значение ближайшего соседа. На рис. 2.8 прогнозы показаны в виде синих звездочек:

```
mglearn.plots.plot_knn_regression(n_neighbors=1)
```

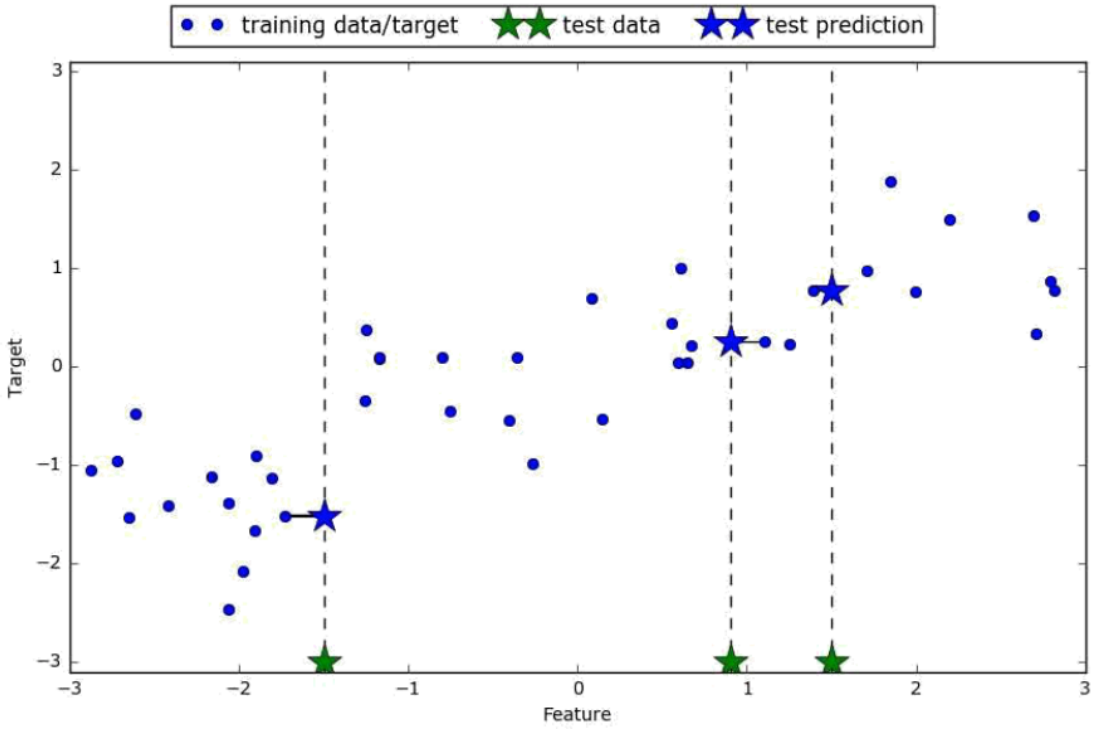

Рис. 2.8 Прогнозы, полученные с помощью регрессионной модели одного ближайшего соседа для набора данных wave

Снова для регрессии мы можем использовать большее количество ближайших соседей. При использовании нескольких ближайших соседей прогнозом становится среднее значение соответствующих соседей (рис.2.9):

```
mglearn.plots.plot_knn_regression(n_neighbors=3)
```

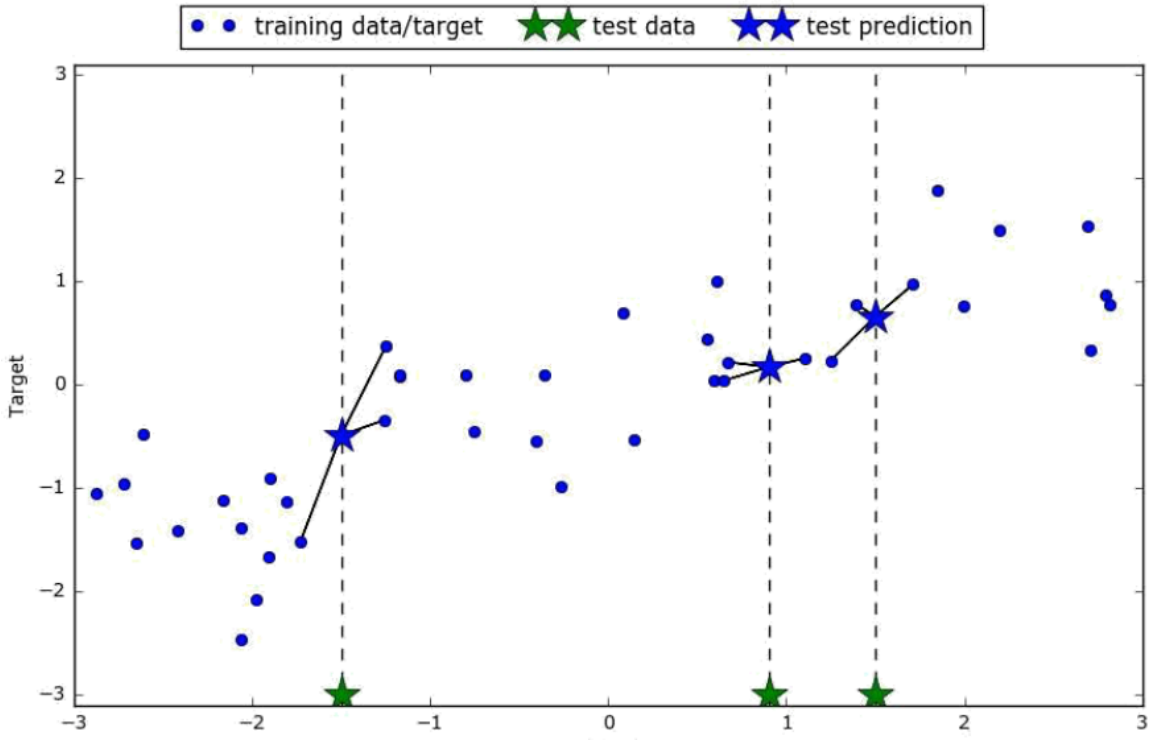

Рис. 2.9 Прогнозы, полученные с помощью регрессионной модели трех ближайших соседей для набора данных wave

Алгоритм регрессии k ближайших соседей реализован в классе ***KNeighborsRegressor***. Он используется точно так же, как ***KNeighborsClassifier***:

```
from sklearn.neighbors import KNeighborsRegressor
X, y = mglearn.datasets.make_wave(n_samples=40)
# разбиваем набор данных wave на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
# создаем экземпляр модели и устанавливаем количество соседей равным 3
reg = KNeighborsRegressor(n_neighbors=3)
#подгоняем модель с использованием обучающих данных и обучающих ответов
reg.fit(X_train, y_train)
#получим прогнозы для тестового набора
print("Прогнозы для тестового набора:\n{}".format(reg.predict(X_test)))
```

```
Прогнозы для тестового набора:
[-0.054 0.357 1.137 -1.894 -1.139 -1.631 0.357 0.912 -0.447 -1.139]
```

Кроме того, мы можем оценить качество модели с помощью метода ***score***, который для регрессионных моделей возвращает значение **R^2**. **R^2**, также известный как коэффициент детерминации, является показателем качества регрессионной модели и принимает значения от 0 до 1. Значение 1 соответствует идеальной прогнозирующей способности, а

значение 0 соответствует константе модели, которая лишь предсказывает среднее значение ответов в обучающем наборе, ***y_train***:

```
print("R^2 на тестовом наборе: {:.2f}".format(reg.score(X_test, y_test)))
```

```
R^2 на тестовом наборе: 0.83
```

Данном случае значение R^2 составляет 0.83, что указывает на относительно хорошее качество подгонки модели.

Анализ модели KNeighborsRegressor

Применительно к нашему одномерному массиву данных мы можем увидеть прогнозы для всех возможных значений признаков (рис. 2.10). Для этого мы создаем тестовый набор данных и визуализируем полученные линии прогнозов:

```
fig, axes = plt.subplots(1, 3, figsize=(15, 4))
line = np.linspace(-3, 3, 1000).reshape(-1, 1)

for n_neighbors, ax in zip([1, 3, 9], axes):
    reg = KNeighborsRegressor(n_neighbors=n_neighbors)
    reg.fit(X_train, y_train)
    ax.plot(line, reg.predict(line))
    ax.plot(X_train, y_train, '^', c=mglearn.cm2(0), markersize=8)
    ax.plot(X_test, y_test, 'v', c=mglearn.cm2(1), markersize=8)
    ax.set_title(
        "{} neighbor(s)\n train score: {:.2f} test score: {:.2f}".format(
            n_neighbors, reg.score(X_train, y_train), reg.score(X_test, y_test)))
    ax.set_xlabel("Признак")
    ax.set_ylabel("Целевая переменная")
axes[0].legend(["Прогнозы модели", "Обучающие данные/ответы",
    "Тестовые данные/ответы"], loc="best")
plt.show()
```

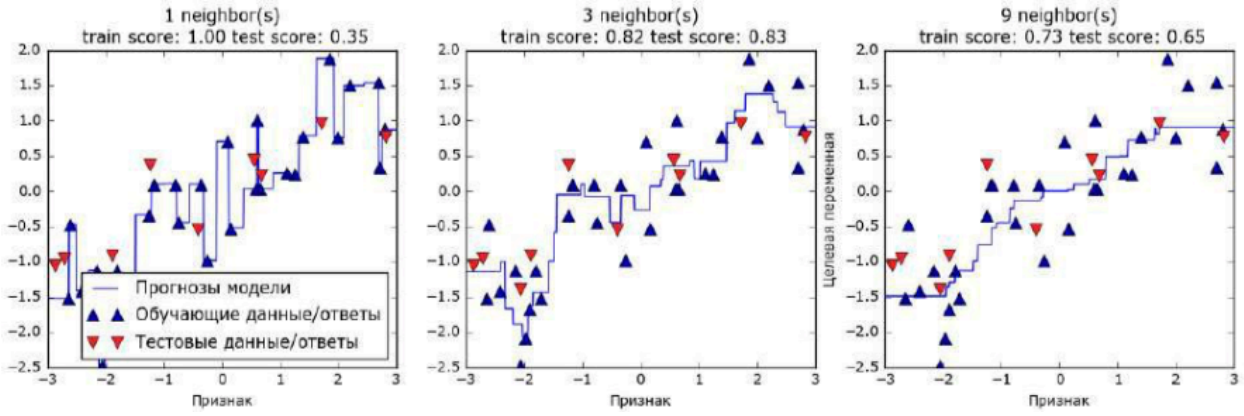

Рис. 2.10 Сравнение прогнозов, полученных с помощью регрессии ближайших соседей для различных значений n_neighbors

Как видно на графике, при использовании лишь одного соседа каждая точка обучающего набора имеет очевидное влияние на прогнозы, и предсказанные значения проходят через все точки данных. Это приводит к очень неустойчивым прогнозам. Увеличение числа соседей приводит к получению более сглаженных прогнозов, но при этом снижается правильность подгонки к обучающим данным.

Преимущества и недостатки

В принципе, в классификаторе KNeighbors есть два важных параметра: количество соседей и мера расстояния между точками данных. На практике использование небольшого числа соседей (например, 3-5) часто работает хорошо, но вы, конечно, можете самостоятельно настроить этот параметр. Вопрос, связанный с выбором правильной меры расстояния, выходит за рамки этой книги. По умолчанию используется евклидово расстояние, которое хорошо работает во многих ситуациях.

Одним из преимуществ метода ближайших соседей является то, что эту модель очень легко интерпретировать и, как правило, этот метод дает приемлемое качество без необходимости использования большого количества настроек. Он является хорошим базовым алгоритмом, который нужно попробовать в первую очередь, прежде чем рассматривать более сложные методы. Как правило, построение модели ближайших соседей происходит очень быстро, но, когда ваш обучающий набор очень большой (с точки зрения количества характеристик или количества наблюдений) получение прогнозов может занять некоторое время. При использовании алгоритма ближайших соседей важно выполнить предварительную обработку данных (смотрите главу 3). Данный метод не так хорошо работает, когда речь идет о наборах данных с большим количеством признаков (сотни и более), и особенно плохо работает в ситуации, когда подавляющее число признаков в большей части наблюдений имеют нулевые значения (так называемые ***разреженные наборы данных*** или *sparse datasets*).

Таким образом, несмотря на то что алгоритм ближайших соседей легко интерпретировать, на практике он не часто используется из-за скорости вычислений и его неспособности обрабатывать большое количество признаков. Метод, который мы обсудим ниже, лишен этих недостатков.

Код к лабораторной работе:

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


X, y = mglearn.datasets.make_forge()
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.legend(["Класс 0", "Класс 1"], loc=4)
plt.xlabel("Первый признак")
plt.ylabel("Второй признак")
print("форма массива X: {}".format(X.shape))
plt.show()


X, y = mglearn.datasets.make_wave(n_samples=40)
plt.plot(X, y, 'o')
plt.ylim(-3, 3)
plt.xlabel("Признак")
plt.ylabel("Целевая переменная")
plt.show()

from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
print("Ключи cancer(): \n{}".format(cancer.keys()))

print("Форма массива data для набора cancer: {}".format(cancer.data.shape))

print("Количество примеров для каждого класса:\n{}".format(
{n: v for n, v in zip(cancer.target_names, np.bincount(cancer.target))}))

print("Количество примеров для каждого класса:\n{}".format(
{n: v for n, v in zip(cancer.target_names, np.bincount(cancer.target))}))

from sklearn.datasets import load_boston
boston = load_boston()
print("форма массива data для набора boston: {}".format(boston.data.shape))

X, y = mglearn.datasets.load_extended_boston()
print("форма массива X: {}".format(X.shape))

mglearn.plots.plot_knn_classification(n_neighbors=1)
plt.show()

mglearn.plots.plot_knn_classification(n_neighbors=3)
plt.show()

from sklearn.model_selection import train_test_split
```

```
X, y = mglearn.datasets.make_forge()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)

clf.fit(X_train, y_train)

print("Прогнозы на тестовом наборе: {}".format(clf.predict(X_test)))

print("Правильность на тестовом наборе: {:.2f}".format(clf.score(X_test, y_test)))


fig, axes = plt.subplots(1, 3, figsize=(10, 3))
for n_neighbors, ax in zip([1, 3, 9], axes):
    clf = KNeighborsClassifier(n_neighbors=n_neighbors).fit(X, y)
    mglearn.plots.plot_2d_separator(clf, X, fill=True, eps=0.5, ax=ax, alpha=.4)
    mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)
    ax.set_title("количество соседей:{}".format(n_neighbors))
    ax.set_xlabel("признак 0")
    ax.set_ylabel("признак 1")
axes[0].legend(loc=3)
plt.show()

from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=66)

training_accuracy = []
test_accuracy = []

neighbors_settings = range(1, 11)

for n_neighbors in neighbors_settings:
    clf = KNeighborsClassifier(n_neighbors=n_neighbors)
    clf.fit(X_train, y_train)
    training_accuracy.append(clf.score(X_train, y_train))
    test_accuracy.append(clf.score(X_test, y_test))

plt.plot(neighbors_settings, training_accuracy, label="правильность на обучающем наборе")
plt.plot(neighbors_settings, test_accuracy, label="правильность на тестовом наборе")
plt.ylabel("Правильность")
plt.xlabel("количество соседей")
plt.legend()
plt.show()

mglearn.plots.plot_knn_regression(n_neighbors=1)
plt.show()
```

```
mglearn.plots.plot_knn_regression(n_neighbors=3)
plt.show()

from sklearn.neighbors import KNeighborsRegressor
X, y = mglearn.datasets.make_wave(n_samples=40)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

reg = KNeighborsRegressor(n_neighbors=3)
reg.fit(X_train, y_train)

print("Прогнозы для тестового набора:\n{}".format(reg.predict(X_test)))

print("R^2 на тестовом наборе: {:.2f}".format(reg.score(X_test, y_test)))

fig, axes = plt.subplots(1, 3, figsize=(15, 4))
line = np.linspace(-3, 3, 1000).reshape(-1, 1)

for n_neighbors, ax in zip([1, 3, 9], axes):
    reg = KNeighborsRegressor(n_neighbors=n_neighbors)
    reg.fit(X_train, y_train)
    ax.plot(line, reg.predict(line))
    ax.plot(X_train, y_train, '^', c=mglearn.cm2(0), markersize=8)
    ax.plot(X_test, y_test, 'v', c=mglearn.cm2(1), markersize=8)
    ax.set_title(
        "{} neighbor(s)\n train score: {:.2f} test score: {:.2f}".format(
            n_neighbors, reg.score(X_train, y_train), reg.score(X_test, y_test)))
    ax.set_xlabel("Признак")
    ax.set_ylabel("Целевая переменная")
axes[0].legend(["Прогнозы модели", "Обучающие данные/ответы",
    "Тестовые данные/ответы"], loc="best")
plt.show()
```