Первый пример: классификация сортов ириса

В этом разделе мы рассмотрим простой пример применения машинного обучения и построим нашу первую модель. В процессе изложения материала мы познакомим вас с некоторыми основными принципами и терминами.

Предположим, что ботаник-любитель хочет классифицировать сорта ирисов, которые он собрал. Он измерил в сантиметрах некоторые характеристики ирисов: длину и ширину лепестков, а также длину и ширину чашелистиков (см. рис. 1.2).

Кроме того, у него есть измерения этих же характеристик ирисов, которые ранее позволили опытному эксперту отнести их к сортам *setosa*, *versicolor* и *virginica*. Относительно этих ирисов ботаник-любитель уверенно может сказать, к какому сорту принадлежит каждый ирис. Давайте предположим, что перечисленные сорта являются единственными сортами, которые ботаник-любитель может встретить в дикой природе.

Наша цель заключается в построении модели машинного обучения, которая сможет обучиться на основе характеристик ирисов, уже классифицированных по сортам, и затем предскажет сорт для нового цветка ириса.

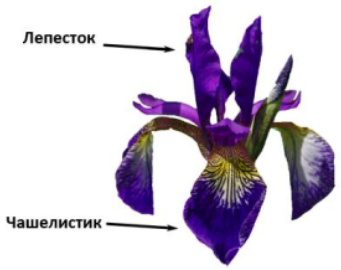

Рис. 1.2 Структура цветка ириса

Поскольку у нас есть примеры, по которых мы уже знаем правильные сорта ириса, решаемая задача является задачей обучения с учителем. В этой задаче нам нужно спрогнозировать один из сортов ириса. Это пример задачи *классификации* (*classification*). Возможные ответы (различные сорта ириса) называются *классами* (*classes*). Каждый ирис в наборе данных принадлежит к одному из трех классов, таким образом решаемая задача является задачей трехклассовой классификации.

Ответом для отдельной точки данных (ириса) является тот или иной сорт этого цветка. Сорт, к которому принадлежит цветок (конкретная точка данных), называется *меткой* (*label*).

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

```
In[10]:
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

```
In[16]:
print("Форма массива data: {}".format(iris_dataset['data'].shape))

Out[16]:
Форма массива data: (150, 4)


In[17]:
print("Первые пять строк массива data:\n{}".format(iris_dataset['data'][:5]))

Out[17]:
Первые пять строк массива data:
[[ 5.1  3.5  1.4  0.2]
 [ 4.9  3.   1.4  0.2]
 [ 4.7  3.2  1.3  0.2]
 [ 4.6  3.1  1.5  0.2]
 [ 5.   3.6  1.4  0.2]]


In[19]:
print("Форма массива target: {}".format(iris_dataset['target'].shape))

Out[19]:
Форма массива target: (150,)


In[20]:
print("Ответы:\n{}".format(iris_dataset['target']))

Out[20]:

Ответы:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Значения чисел задаются массивом `iris['target_names']`: 0 – *setosa*, 1 – *versicolor*, а 2 – *virginica*.

Для оценки эффективности модели, мы предъявляем ей новые размеченные данные (размеченные данные, которые она не видела раньше). Обычно это делается путем разбиения собранных размеченных данных (в данном случае 150 цветов) на две части. Одна часть данных используется для построения нашей модели машинного обучения и называется *обучающими данными* (*training data*) или *обучающим набором* (*training set*). Остальные данные будут использованы для оценки качества модели, их называют *тестовыми данными* (*test data*), *тестовым набором* (*test set*) или *контрольным набором* (*hold-out set*).

В библиотеке `scikit-learn` есть функция `train_test_split`, которая перемешивает набор данных и разбивает его на две части. Эта функция отбирает в обучающий набор 75% строк данных с соответствующими метками. Оставшиеся 25% данных с метками объявляются тестовым набором. Вопрос о том, сколько данных отбирать в обучающий и тестовый наборы, является дискуссионным, однако использование тестового набора, содержащего 25% данных, является хорошим правилом.

В `scikit-learn` данные, как правило, обозначаются заглавной `X`, тогда как метки обозначаются строчной `y`. Это навеяно стандартной математической формулой $f(x)=y$, где x является аргументом функции, а y – выводом. В соответствии с некоторыми математическими соглашениями мы используем заглавную `X`, потому что данные представляют собой двумерный массив (матрицу) и строчную `y`, потому что целевая переменная – это одномерный массив (вектор).

```
In[21]:
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset['data'], iris_dataset['target'], random_state=0)
```

Перед разбиением функция `train_test_split` перемешивает набор данных с помощью генератора псевдослучайных чисел. Если мы просто возьмем последние 25% наблюдений в качестве тестового набора, все точки данных будет иметь метку 2, поскольку все точки данных отсортированы по меткам (смотрите вывод для `iris['target']`, показанный ранее). Используя тестовый набор, содержащий только один из трех классов, вы не сможете объективно судить об обобщающей способности модели, таким образом, мы перемешиваем наши данные, чтобы тестовые данные содержали все три класса.

Чтобы в точности повторно воспроизвести полученный результат, мы воспользуемся генератором псевдослучайных чисел с фиксированным стартовым значением, которое задается с помощью параметра `random_state`. Это позволит сделать результат воспроизводим, поэтому вышеприведенный программный код будет генерировать один и тот же результат. Мы всегда будем задавать `random_state` при использовании рандомизированных процедур в этой книге.

```
In[24]:
# создаем dataframe из данных в массиве X_train
# маркируем столбцы, используя строки в iris_dataset.feature_names
iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)
# создаем матрицу рассеяния из dataframe, цвет точек задаем с помощью y_train
grr = pd.scatter_matrix(iris_dataframe, c=y_train, figsize=(15, 15), marker='o',
                        hist_kwds={'bins': 20}, s=60, alpha=.8, cmap=mglearn.cm3)
```

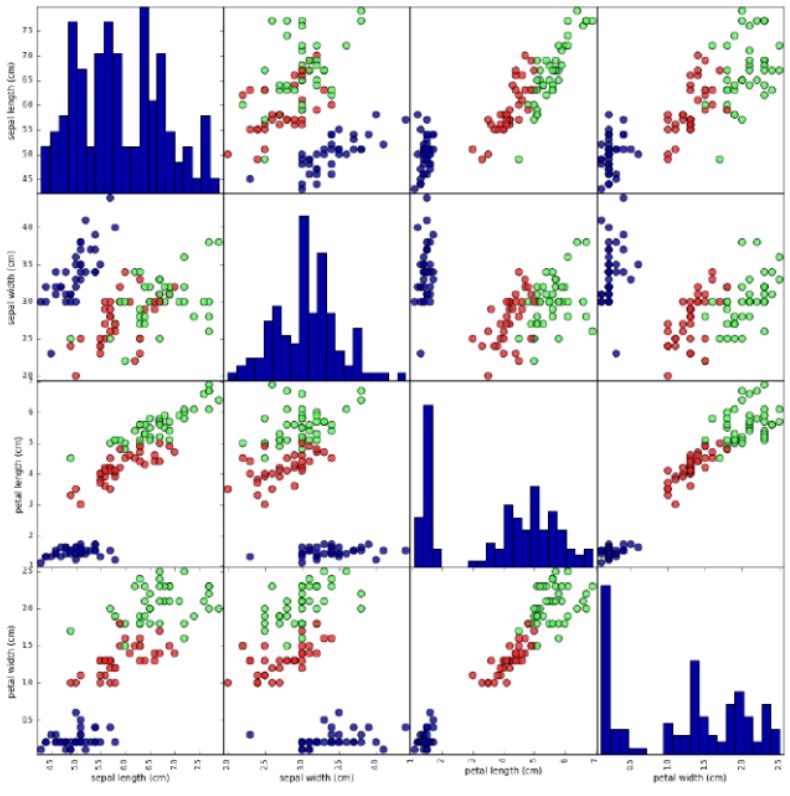

Рис. 1.3 Матрица диаграмм рассеяния для набора данных Iris, цвет точек данных определяется метками классов

В `scikit-learn` все модели машинного обучения реализованы в собственных классах, называемых классами `Estimator`. Алгоритм классификации на основе метода k ближайших соседей реализован в классификаторе `KNeighborsClassifier` модуля `neighbors`. Прежде чем использовать эту модель, нам нужно создать объект-экземпляр класса. Это произойдет, когда мы зададим параметры модели. Самым важным параметром `KNeighborsClassifier` является количество соседей, которые мы установим равным 1:

```
In[25]:
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

Для построения модели на обучающем наборе, мы вызываем метод `fit` объекта `knn`, который принимает в качестве аргументов массив NumPy `X_train`, содержащий обучающие данные, и массив NumPy `y_train`, соответствующий обучающим меткам:

```
In[26]:
knn.fit(X_train, y_train)

Out[26]:
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

```
In[27]:
X_new = np.array([[5, 2.9, 1, 0.2]])
print("форма массива X_new: {}".format(X_new.shape))

Out[27]:
форма массива X_new: (1, 4)
```

```
In[28]:
prediction = knn.predict(X_new)
print("Прогноз: {}".format(prediction))
print("Спрогнозированная метка: {}".format(
       iris_dataset['target_names'][prediction]))

Out[28]:
Прогноз: [0]
Спрогнозированная метка: ['setosa']
```

Таким образом, мы можем сделать прогноз для каждого ириса в тестовом наборе и сравнить его с фактической меткой (уже известным сортом). Мы можем оценить качество модели, вычислив *правильность* (*accuracy*) – процент цветов, для которых модель правильно спрогнозировала сорта:

```
In[29]:
y_pred = knn.predict(X_test)
print("Прогнозы для тестового набора:\n {}".format(y_pred))

Out[29]:
Прогнозы для тестового набора:
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0 2]
```

```
In[30]:
print("Правильность на тестовом наборе: {:.2f}".format(np.mean(y_pred == y_test)))

Out[30]:
Правильность на тестовом наборе: 0.97
```

```
In[31]:
print("Правильность на тестовом наборе: {:.2f}".format(knn.score(X_test, y_test)))

Out[31]:
Правильность на тестовом наборе: 0.97
```

Ниже приводится краткое изложение программного кода, необходимого для всей процедуры обучения и оценки модели:

```
In[32]:
X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset['data'], iris_dataset['target'], random_state=0)

knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)


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

Out[32]:
Правильность на тестовом наборе: 0.97
```

Этот фрагмент содержит базовый код, необходимый для применения любого алгоритма машинного обучения с помощью `scikit-learn`. Методы `fit`, `predict` и `score` являются общими для моделей