# Лабораторная работа №1

## 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. 1.1).

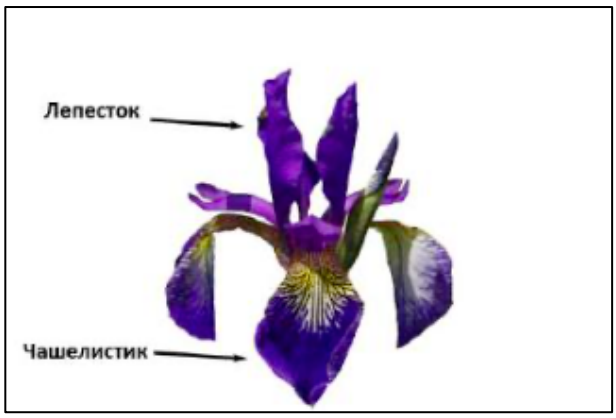


Рисунок 1.1 Структура цветка ириса

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.

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)*.

## Загружаем данные

Данные, которые мы будем использовать для этого примера, – это набор данных Iris, классический набор данных в машинном обучении и статистике. Он уже включен в модуль ***datasets*** библиотеки ***scikit-learn***. Мы можем загрузить его, вызвав функцию ***load_iris***:

```
iris_dataset = load_iris()
```

Объект *iris*, возвращаемый *load_iris*, является объектом Bunch, который очень похож на словарь. Он содержит ключи и значения:

```
print("Ключи iris_dataset: \n{}".format(iris_dataset.keys()))
```

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

Значение ключа DESCR – это краткое описание набора данных. Здесь мы покажем начало описания (оставшуюся часть описания вы можете посмотреть самостоятельно):

```
print(iris_dataset['DESCR'][:193] + "\n...")
```

```
Iris Plants Database
====================
Notes
----
Data Set Characteristics:
:Number of Instances: 150 (50 in each of three classes)
:Number of Attributes: 4 numeric, predictive att
...
----
```

Значение ***target_names*** – это массив строк, содержащий сорта цветов, которые мы хотим предсказать:

```
print("Названия ответов: {}".format(iris_dataset['target_names']))
```

```
Названия ответов: ['setosa' 'versicolor' 'virginica']
```

Значение ***feature_names*** – это список строк с описанием каждого признака:

```
print(
   "Названия признаков:\n{}".format(iris_dataset['feature_names'])
)
```

```
Названия признаков:
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
 'petal width (cm)']
```

Сами данные записаны в массивах ***target*** и ***data***. ***data*** – массив NumPy, который содержит количественные измерения длины чашелистиков, ширины чашелистиков, длины лепестков и ширины лепестков:

```
print("Тип массива data: {}".format(type(iris_dataset['data'])))
```

```
Тип массива data: <class 'numpy.ndarray'>
```

Строки в массиве ***data*** соответствуют цветам ириса, а столбцы представляют собой четыре признака, которые были измерены для каждого цветка:

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

```
Форма массива data: (150, 4)
```

Мы видим, что массив содержит измерения для 150 различных цветов по 4 признакам. Вспомним, что в машинном обучении отдельные элементы называются ***примерами*** *(samples)*, а их свойства – ***характеристиками*** или ***признаками*** *(feature)*. ***Форма*** *(shape)* массива данных определяется количеством примеров, умноженным на количество признаков. Это является общепринятым соглашением в **scikit-learn**, и ваши данные всегда будут представлены в этой форме. Ниже приведены значения признаков для первых пяти примеров:

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

```
Первые пять строк массива data:
[
 [ 5.1 3.5 1.4 0.2]
 [ 4.9 3.0 1.4 0.2]
 [ 4.7 3.2 1.3 0.2]
 [ 4.6 3.1 1.5 0.2]
 [ 5.0 3.6 1.4 0.2]
]
```

Взглянув на эти данные, мы видим, что все пять цветов имеют ширину лепестка 0.2 см и первый цветок имеет самую большую длину чашелистика, 5.1 см.

Массив ***target*** содержит сорта уже измеренных цветов, тоже записанные в виде массива NumPy:

```
print("Тип массива target: {}".format(type(iris_dataset['target'])))
```

```
Тип массива target: <class 'numpy.ndarray'>
```

***target*** представляет собой одномерный массив, по одному элементу для каждого цветка:

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

```
Форма массива target: (150,)
```

Сорта кодируются как целые числа от 0 до 2:

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

```
Ответы:[
000000000000000000000
000000000000000000000
000000000011111111111
111111111111111111111
111111111111111111111
222222222222222222222
222222222222222222222
2222222222
]
```

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

## Метрики эффективности: обучающий и тестовый наборы

На основе этих данных нам нужно построить модель машинного обучения, которая предскажет сорта ириса для нового набора измерений. Но прежде, чем мы применить нашу модель к новому набору, мы должны убедиться в том, что модель на самом деле работает и ее прогнозам можно доверять.

К сожалению, для оценки качества модели мы не можем использовать данные, которые были взяты нами для построения модели. Это обусловлено тем, что наша модель просто запомнит весь обучающий набор и поэтому она всегда будет предсказывать правильную метку для любой точки данных в обучающем наборе. Это «запоминание» ничего не говорит нам об

обобщающей способности модели (другими словами, мы не знаем, будет ли эта модель так же хорошо работать на новых данных).

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 к***онтрольным 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).

Давайте вызовем функцию ***train_test_split*** для наших данных и зададим обучающие данные, обучающие метки, тестовые данные, тестовые метки, используя вышеупомянутые буквы:

```
X_train, X_test, y_train, y_test = train_test_scipylit(
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.

Выводом функции ***train_test_split*** являются ***X_train***, ***X_test***, ***y_train*** и ***y_test***, которые все являются массивами Numpy. **X_train** содержит 75% строк набора данных, а ***X_test*** содержит оставшиеся 25%:

```
print("форма массива X_train: {}".format(X_train.shape))
print("форма массива y_train: {}".format(y_train.shape))

print("форма массива X_test: {}".format(X_test.shape))
print("форма массива y_test: {}".format(y_test.shape))
```

```
форма массива X_train: (112, 4)
форма массива y_train: (112,)
форма массива X_test: (38, 4)
форма массива y_test: (38,)
```

## Просмотр данных

Перед тем как строить модель машинного обучения, неплохо было бы исследовать данные, чтобы понять, можно ли легко решить поставленную задачу без машинного обучения или содержится ли нужная информация в данных.

Кроме того, исследование данных – это хороший способ обнаружить аномалии и особенности. Например, вполне возможно, что некоторые из ваших ирисов измерены в дюймах, а не в сантиметрах. В реальном мире нестыковки в данных и неожиданности очень распространены.

Один из лучших способов исследовать данные – визуализировать их. Это можно сделать, используя ***диаграмму рассеяния*** *(scatter plot)*. В диаграмме рассеяния один признак откладывается по оси x, а другой признак – по оси y, каждому наблюдению соответствует точка. К сожалению, экран компьютера имеют только два измерения, что позволяет разместить на графике только два (или, возможно, три) признака одновременно. Таким образом, трудно разместить на графике наборы данных с более чем тремя признаками. Один из способов решения этой проблемы – построить ***матрицу диаграмм рассеяния*** *(scatterplot matrix)* или парные ***диаграммы рассеяния*** *(pair plots)*, на которых будут изображены все возможные пары признаков. Если у вас есть небольшое число признаков, например, четыре, как здесь, то использование матрицы диаграмм рассеяния будет вполне разумным. Однако, вы должны помнить, что матрица диаграмм рассеяния не показывает взаимодействие между всеми признаками сразу, поэтому некоторые интересные аспекты данных не будут выявлены с помощью этих графиков.

Рис. 1.2 представляет собой матрицу диаграмм рассеяния для признаков обучающего набора. Точки данных окрашены в соответствии с сортами ириса, к которым они относятся. Чтобы построить диаграммы, мы сначала преобразовываем массив **NumPy** в ***DataFrame*** *(основной тип данных в библиотеке pandas)*. В **pandas** есть функция для создания парных диаграмм рассеяния под названием ***scatter_matrix***. По диагонали этой матрицы располагаются гистограммы каждого признака:

```
# 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
)
pyplot.show()
```

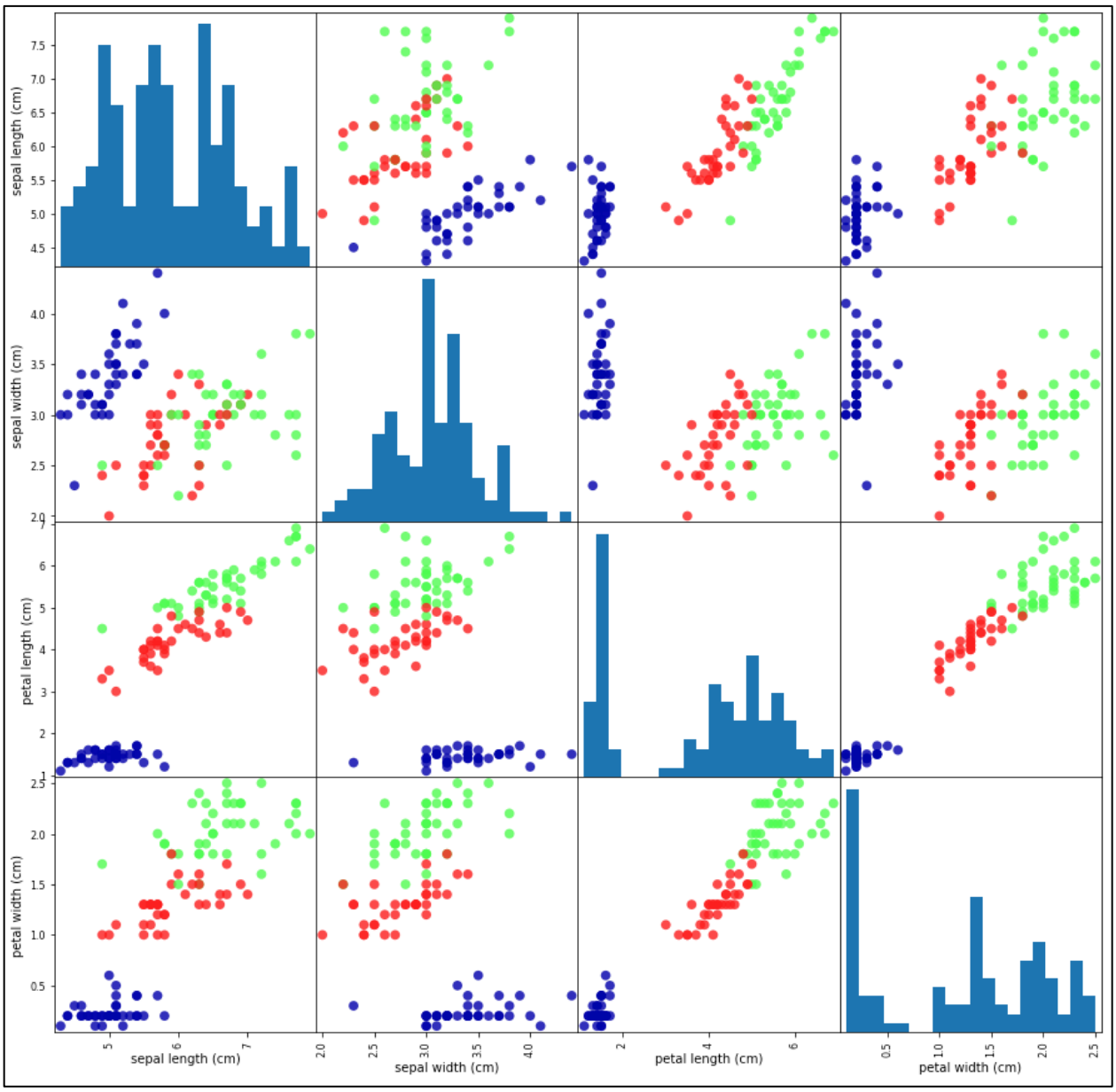


Рисунок 1.2 Матрица диаграмм рассеяния для набора данных Iris, цвет точек данных определяется метками классов

Взглянув на график, мы можем увидеть, что, похоже, измерения чашелистиков и лепестков позволяют относительно хорошо разделить три класса. Это означает, что модель машинного обучения, вероятно, сможет научиться разделять их.

Код к лабораторной работе:

```
import sys
import pandas as pds
import mglearn
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
import IPython
import sklearn

from sklearn.datasets import load_iris
iris_dataset = load_iris()

print("Ключи iris_dataset: \n{}".format(iris_dataset.keys()))

print(iris_dataset['DESCR'][:193] + "\n...")

print("Названия ответов: {}".format(iris_dataset['target_names']))

print("Названия признаков: \n{}".format(iris_dataset['feature_names']))

print("Тип массива data: {}".format(type(iris_dataset['data'])))

print("Форма массива data: {}".format(iris_dataset['data'].shape))
print("Первые пять строк массива data:\n{}".format(iris_dataset['data'][:5]))
print("Тип массива target: {}".format(type(iris_dataset['target'])))
print("Форма массива target: {}".format(iris_dataset['target'].shape))

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

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)

print("форма массива X_train: {}".format(X_train.shape))
print("форма массива y_train: {}".format(y_train.shape))

print("форма массива X_test: {}".format(X_test.shape))
print("форма массива y_test: {}".format(y_test.shape))

iris_dataframe = pds.DataFrame(X_train, columns=iris_dataset.feature_names)

from pandas.plotting import scatter_matrix

grr = scatter_matrix(iris_dataframe, c=y_train, figsize=(15, 15), marker='o',
        hist_kwds={'bins': 20}, s=60, alpha=.8, cmap=mglearn.cm3)

plt.show()
```