Обзор методов классификации в машинном обучении с помощью Scikit-Learn

Перевод статьи [«Overview of Classification Methods in Python with Scikit-Learn»](https://stackabuse.com/overview-of-classification-methods-in-python-with-scikit-learn/)



[Free Vectors by Vecteezy](https://www.vecteezy.com/)

Для машинного обучения на Python написано очень много библиотек. Сегодня мы рассмотрим одну из самых популярных — Scikit-Learn.

Scikit-Learn упрощает процесс создания классификатора и помогает более чётко выделить концепции машинного обучения, реализуя их с помощью понятной, хорошо документированной и надёжной библиотекой.

Что такое Scikit-Learn?

[Scikit-Learn](https://scikit-learn.org/stable/) — это Python-библиотека, впервые разработанная David Cournapeau в 2007 году. В этой библиотеке находится большое количество алгоритмов для задач, связанных с классификацией и машинным обучением в целом.

Scikit-Learn базируется на библиотеке [SciPy](https://www.scipy.org/" \t "_blank), которую нужно установить перед началом работы.

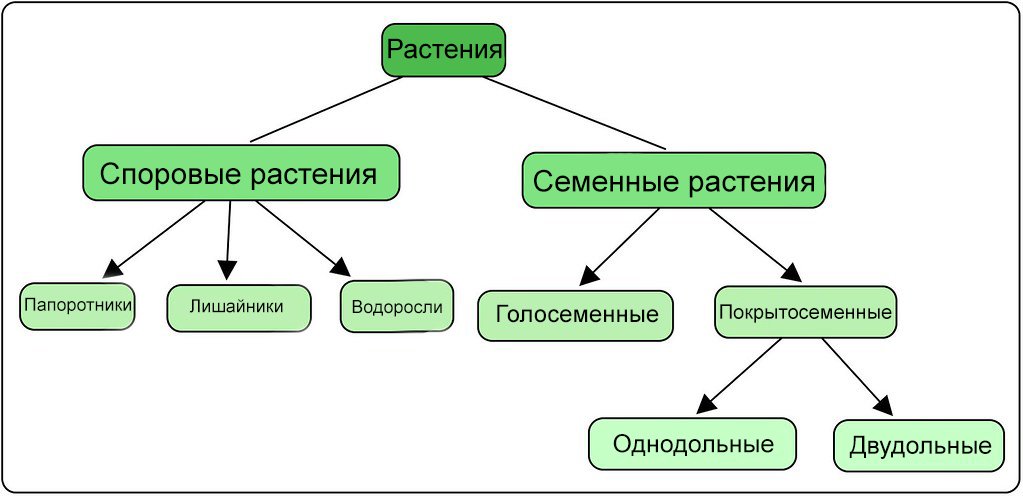
Основные термины

В системах машинного обучения или же системах нейросетей существуют входы и выходы. То, что подаётся на входы, принято называть [признаками](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%B7%D0%BD%D0%B0%D0%BA_(%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD)) (англ. *features*).

Признаки по существу являются тем же, что и переменные в научном эксперименте — они характеризуют какой-либо наблюдаемый феномен и их можно как-то количественно измерить.

Когда признаки подаются на входы системы машинного обучения, эта система пытается найти совпадения, заметить закономерность между признаками. На выходе генерируется результат этой работы.

Этот результат принято называть меткой (англ. *label*), поскольку у выходов есть некая пометка, выданная им системой, т. е. предположение (прогноз) о том, в какую категорию попадает выход после классификации.

[](https://s3.tproger.ru/uploads/2019/05/overview-classification-methods-python-scikit-learn-2-1.jpg)

В контексте машинного обучения классификация относится к [обучению с учителем](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5_%D1%81_%D1%83%D1%87%D0%B8%D1%82%D0%B5%D0%BB%D0%B5%D0%BC). Такой тип обучения подразумевает, что данные, подаваемые на входы системы, уже помечены, а важная часть признаков уже разделена на отдельные категории или классы. Поэтому сеть уже знает, какая часть входов важна, а какую часть можно самостоятельно проверить. Пример классификации — сортировка различных растений на группы, например «папоротники» и «покрытосеменные». Подобная задача может быть выполнена с помощью [Дерева Решений](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%B0_%D1%80%D0%B5%D1%88%D0%B5%D0%BD%D0%B8%D0%B9) — одного из типов классификатора в Scikit-Learn.

При обучении без учителя в систему подаются непомеченные данные, и она должна попытаться сама разделить эти данные на категории. Так как классификация относится к типу обучения с учителем, способ обучения без учителя в этой статье рассматриваться не будет.

Процесс обучения модели — это подача данных для нейросети, которая в результате должна вывести определённые шаблоны для данных. В процессе обучения модели с учителем на вход подаются признаки и метки, а при прогнозировании на вход классификатора подаются только признаки.

Принимаемые сетью данные делятся на две группы: набор данных для обучения и набор для тестирования. Не стоит проверять сеть на том же наборе данных, на которых она обучалась, т. к. модель уже будет «заточена» под этот набор.

Типы классификаторов

Scikit-Learn даёт доступ ко множеству различных алгоритмов классификации. Вот основные из них:

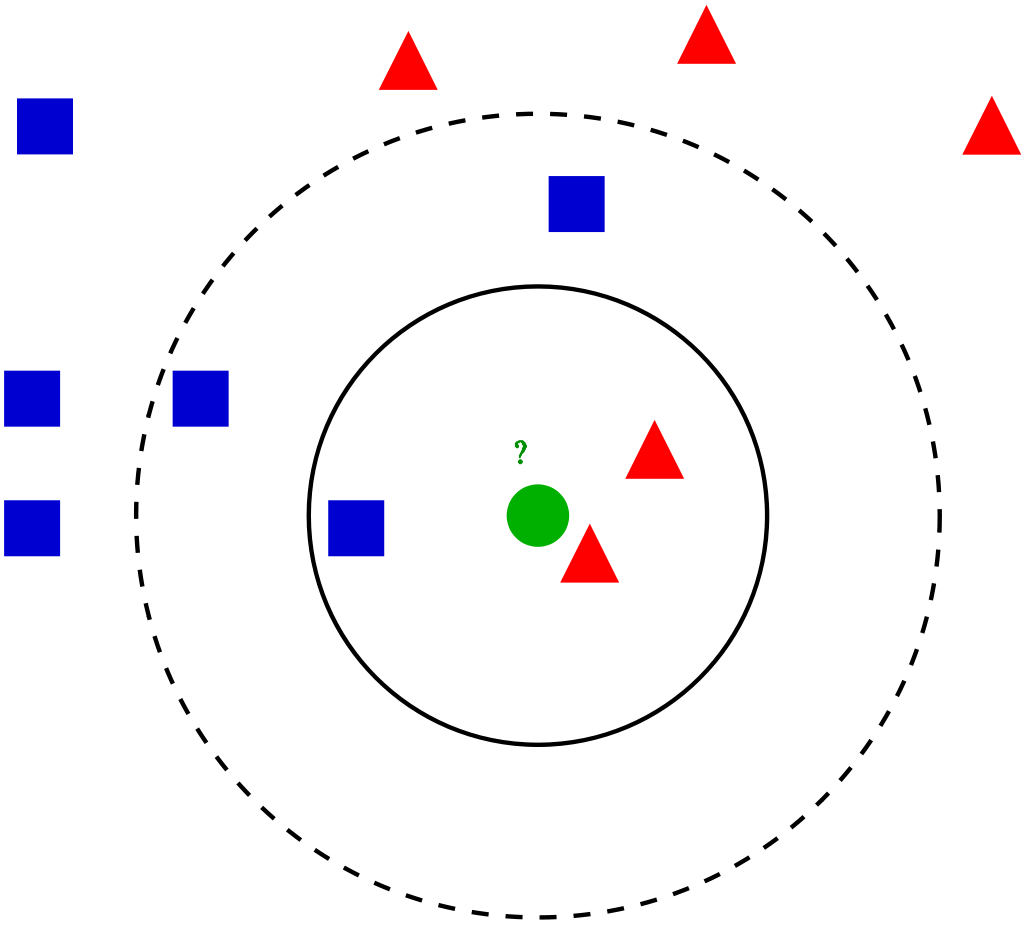
* [Метод k-ближайших соседей (K-Nearest Neighbors)](https://stackabuse.com/k-nearest-neighbors-algorithm-in-python-and-scikit-learn/);
* [Метод опорных векторов (Support Vector Machines)](https://stackabuse.com/implementing-svm-and-kernel-svm-with-pythons-scikit-learn/);
* [Классификатор дерева решений (Decision Tree Classifier)](https://stackabuse.com/decision-trees-in-python-with-scikit-learn/) / [Случайный лес (Random Forests)](https://stackabuse.com/random-forest-algorithm-with-python-and-scikit-learn/);
* [Наивный байесовский метод (Naive Bayes)](https://stackabuse.com/the-naive-bayes-algorithm-in-python-with-scikit-learn/);
* [Линейный дискриминантный анализ (Linear Discriminant Analysis)](https://stackabuse.com/implementing-lda-in-python-with-scikit-learn/);
* [Логистическая регрессия (Logistic Regression)](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D1%81%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%80%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%81%D0%B8%D1%8F);

На [сайте Scikit-Learn](https://scikit-learn.org/stable/user_guide.html) есть много литературы на тему этих алгоритмов с кратким пояснением работы каждого из них.

**[Обзор самых популярных алгоритмов машинного обучения](https://tproger.ru/translations/top-machine-learning-algorithms/)**

[tproger.ru](https://tproger.ru/translations/top-machine-learning-algorithms/)

Метод k-ближайших соседей (K-Nearest Neighbors)

[](https://s3.tproger.ru/uploads/2019/05/overview-classification-methods-python-scikit-learn-4.png)

Этот метод работает с помощью поиска кратчайшей дистанции между тестируемым объектом и ближайшими к нему классифицированным объектами из обучающего набора. Классифицируемый объект будет относится к тому классу, к которому принадлежит ближайший объект набора.

**Классификатор дерева решений (Decision Tree Classifier)**

Этот классификатор разбивает данные на всё меньшие и меньшие подмножества на основе разных критериев, т. е. у каждого подмножества своя сортирующая категория. С каждым разделением количество объектов определённого критерия уменьшается.

Классификация подойдёт к концу, когда сеть дойдёт до подмножества только с одним объектом. Если объединить несколько подобных деревьев решений, то получится так называемый *Случайный Лес* (англ. *Random Forest*).

**Наивный байесовский классификатор (Naive Bayes)**

Такой классификатор вычисляет вероятность принадлежности объекта к какому-то классу. Эта вероятность вычисляется из шанса, что какое-то событие произойдёт, с опорой на уже на произошедшие события.

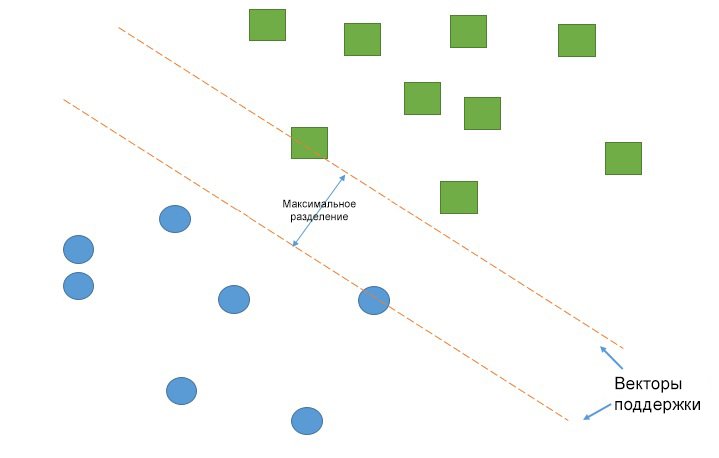
Каждый параметр классифицируемого объекта считается независимым от других параметров.

**Линейный дискриминантный анализ (Linear Discriminant Analysis)**

Этот метод работает путём уменьшения размерности набора данных, проецируя все точки данных на линию. Потом он комбинирует эти точки в классы, базируясь на их расстоянии от центральной точки.

Этот метод, как можно уже догадаться, относится к линейным алгоритмам классификации, т. е. он хорошо подходит для данных с линейной зависимостью.

**Метод опорных векторов (Support Vector Machines)**

[](https://s3.tproger.ru/uploads/2019/05/overview-classification-methods-python-scikit-learn-5-1.jpg)

Работа метода опорных векторов заключается в рисовании линии между разными кластерами точек, которые нужно сгруппировать в классы. С одной стороны линии будут точки, принадлежащие одному классу, с другой стороны — к другому классу.

Классификатор будет пытаться увеличить расстояние между рисуемыми линиями и точками на разных сторонах, чтобы увеличить свою «уверенность» определения класса. Когда все точки построены, сторона, на которую они падают — это класс, которому эти точки принадлежат.

**Логистическая регрессия (Logistic Regression)**

Логистическая регрессия выводит прогнозы о точках в бинарном масштабе — нулевом или единичном. Если значение чего-либо равно либо больше 0.5, то объект классифицируется в большую сторону (к единице). Если значение меньше 0.5 — в меньшую (к нулю).

У каждого признака есть своя метка, равная только 0 или только 1. Логистическая регрессия является линейным классификатором и поэтому используется, когда в данных прослеживается какая-то линейная зависимость.

**Примеры задач классификации**

Задача классификации — эта любая задача, где нужно определить тип объекта из двух и более существующих классов. Такие задачи могут быть разными: определение, кошка на изображении или собака, или определение качества вина на основе его кислотности и содержания алкоголя.

В зависимости от задачи классификации вы будете использовать разные типы классификаторов. Например, если классификация содержит какую-то бинарную логику, то к ней лучше всего подойдёт логистическая регрессия.

По мере накопления опыта вам будет проще выбирать подходящий тип классификатора. Однако хорошей практикой является реализация нескольких подходящих классификаторов и выбор наиболее оптимального и производительного.

**Реализация классификатора**

Первый шаг в реализации классификатора — его импорт в Python. Вот как это выглядит для логистической регрессии:

from sklearn.linear\_model import LogisticRegression

Вот импорты остальных классификаторов, рассмотренных выше:

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

Однако, это не все классификаторы, которые есть в Scikit-Learn. Про остальные можно прочитать на соответствующей странице в документации.

После этого нужно создать экземпляр классификатора. Сделать это можно создав переменную и вызвав функцию, связанную с классификатором.

logreg\_clf = LogisticRegression()

Теперь классификатор нужно обучить. Перед этим нужно «подогнать» его под тренировочные данные.

Обучающие признаки и метки помещаются в классификатор через функцию fit:

logreg\_clf.fit(features, labels)

После обучения модели данные уже можно подавать в классификатор. Это можно сделать через функцию классификатора predict, передав ей параметр (признак) для прогнозирования:

logreg\_clf.predict(test\_features)

Эти этапы (создание экземпляра, обучение и классификация) являются основными при работе с классификаторами в Scikit-Learn. Но эта библиотека может управлять не только классификаторами, но и самими данными. Чтобы разобраться в том, как данные и классификатор работают вместе над задачей классификации, нужно разобраться в процессах машинного обучения в целом.

**Процесс машинного обучения**

Процесс содержит в себе следующие этапы: подготовка данных, создание обучающих наборов, создание классификатора, обучение классификатора, составление прогнозов, оценка производительности классификатора и настройка параметров.

Во-первых, нужно подготовить набор данных для классификатора — преобразовать данные в корректную для классификации форму и обработать любые аномалии в этих данных. Отсутствие значений в данных либо любые другие отклонения — все их нужно обработать, иначе они могут негативно влиять на производительность классификатора. Этот этап называется предварительной обработкой данных (англ. *data preprocessing*).

Следующим шагом будет разделение данных на обучающие и тестовые наборы. Для этого в Scikit-Learn существует отличная функция [traintestsplit](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html" \t "_blank).

Как уже было сказано выше, классификатор должен быть создан и обучен на тренировочном наборе данных. После этих шагов модель уже может делать прогнозы. Сравнивая показания классификатора с фактически известными данными, можно делать вывод о точности классификатора.

Вероятнее всего, вам нужно будет «корректировать» параметры классификатора, пока вы не достигните желаемой точности (т. к. маловероятно, что классификатор будет соответствовать всем вашим требованиям с первого же запуска).

Ниже будет представлен пример работы машинного обучения от обработки данных и до оценки.

**Реализация образца классификации**

# Импорт всех нужных библиотек

import pandas as pd

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

Поскольку набор данных iris достаточно распространён, в Scikit-Learn он уже присутствует, достаточно лишь заложить эту команду:

sklearn.datasets.load\_iris

Тем не менее, тут ещё нужно подгрузить CSV-файл, который можно скачать [здесь](https://www.kaggle.com/uciml/iris).

Этот файл нужно поместить в ту же папку, что и Python-файл. В библиотеке [Pandas](https://pandas.pydata.org/" \t "_blank) есть функция read\_csv(), которая отлично работает с загрузкой данных.

data = pd.read\_csv('iris.csv')

# Проверяем, всё ли правильно загрузилось

print(data.head(5))

Благодаря тому, что данные уже были подготовлены, долгой предварительной обработки они не требуют. Единственное, что может понадобиться — убрать ненужные столбцы (например ID) таким образом:

data.drop('Id', axis=1, inplace=True)

Теперь нужно определить признаки и метки. С библиотекой Pandas можно легко «нарезать» таблицу и выбрать определённые строки/столбцы с помощью функции iloc():

# ".iloc" принимает row\_indexer, column\_indexer

X = data.iloc[:,:-1].values

# Теперь выделим нужный столбец

y = data['Species']

Пояснение (И.К.)

Sepal – чашелистик

Petal – лепесток

Species – виды

Код выше выбирает каждую строку и столбец, обрезав при этом последний столбец.

Выбрать признаки интересующего вас набора данных можно также передав в скобках заголовки столбцов:

# Альтернативный способ выбора нужных столбцов:

X = data.iloc['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm']

После того, как вы выбрали нужные признаки и метки, их можно разделить на тренировочные и тестовые наборы, используя функцию train\_test\_split():

# test\_size показывает, какой объем данных нужно выделить для тестового набора

# Random\_state — просто сид для случайной генерации

# Этот параметр можно использовать для воссоздания определённого результата:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=27)

Чтобы убедиться в правильности обработки данных, используйте:

print(X\_train)

print(y\_train)

Теперь можно создавать экземпляр классификатора, например метод опорных векторов и метод k-ближайших соседей:

SVC\_model = svm.SVC()

# В KNN-модели нужно указать параметр n\_neighbors

# Это число точек, на которое будет смотреть

# классификатор, чтобы определить, к какому классу принадлежит новая точка

KNN\_model = KNeighborsClassifier(n\_neighbors=5)

Теперь нужно обучить эти два классификатора:

SVC\_model.fit(X\_train, y\_train)

KNN\_model.fit(X\_train, y\_train)

Эти команды обучили модели и теперь классификаторы могут делать прогнозы и сохранять результат в какую-либо переменную.

SVC\_prediction = SVC\_model.predict(X\_test)

KNN\_prediction = KNN\_model.predict(X\_test)

Теперь пришло время оценить точности классификатора. Существует несколько способов это сделать.

Нужно передать показания прогноза относительно фактически верных меток, значения которых были сохранены ранее.

# Оценка точности — простейший вариант оценки работы классификатора

print(accuracy\_score(SVC\_prediction, y\_test))

print(accuracy\_score(KNN\_prediction, y\_test))

# Но матрица неточности и отчёт о классификации дадут больше информации о производительности

print(confusion\_matrix(SVC\_prediction, y\_test))

print(classification\_report(KNN\_prediction, y\_test))

Вот, к примеру, результат полученных метрик:

SVC accuracy: 0.9333333333333333

KNN accuracy: 0.9666666666666667

Поначалу кажется, что KNN работает точнее. Вот матрица неточностей для SVC:

[[ 7 0 0]

[ 0 10 1]

[ 0 1 11]]

Количество правильных прогнозов идёт с верхнего левого угла в нижний правый. Вот для сравнения метрики классификации для KNN:

precision recall f1-score support

Iris-setosa 1.00 1.00 1.00 7

Iris-versicolor 0.91 0.91 0.91 11

Iris-virginica 0.92 0.92 0.92 12

micro avg 0.93 0.93 0.93 30

macro avg 0.94 0.94 0.94 30

weighted avg 0.93 0.93 0.93 30

AVERAGE – среднее значение (И.К.)

**Оценка классификатора**

Когда дело доходит до оценки точности классификатора, есть несколько вариантов.

**Точность классификации**

Точность классификации измерять проще всего, и поэтому этот параметр чаще всего используется. Значение точности — это число правильных прогнозов, делённое на число всех прогнозов или, проще говоря, отношение правильных прогнозов ко всем.

Хоть этот показатель и может быстро дать вам явное представление о производительности классификатора, его лучше использовать, когда каждый класс имеет хотя бы примерно одинаковое количество примеров. Так как такое будет случаться редко, рекомендуется использовать другие показатели классификации.

**Логарифмические потери**

Значение [Логарифмических Потерь](https://datawookie.netlify.com/blog/2015/12/making-sense-of-logarithmic-loss/) (англ. *Logarithmic Loss*) — или просто *логлосс* — показывает, насколько классификатор «уверен» в своём прогнозе. Логлосс возвращает вероятность принадлежности объекта к тому или иному классу, суммируя их, чтобы дать общее представление об «уверенности» классификатора.

Этот показатель лежит в промежутке от 0 до 1 — «совсем не уверен» и «полностью уверен» соответственно. Логлосс сильно падает, когда классификатор сильно «уверен» в неправильном ответе.

**Площадь ROC-кривой (AUC)**

Такой показатель используется только при бинарной классификации. Площадь под ROC-кривой представляет способность классификатора различать подходящие и не подходящие какому-либо классу объекты.

Значение 1.0: вся область, попадающая под кривую, представляет собой идеальный классификатор. Следовательно, 0.5 означает, что точность классификатора соответствует случайности. Кривая рассчитывается с учётом точности и специфичности модели. Подробнее о расчётах можно прочитать [здесь](https://stackabuse.com/understanding-roc-curves-with-python/).

**Матрица неточностей**

*Матрица неточностей* (англ. *Confusion Matrix*) — это таблица или диаграмма, показывающая точность прогнозирования классификатора в отношении двух и более классов. Прогнозы классификатора находятся на оси X, а результат (точность) — на оси Y.

Ячейки таблицы заполняются количеством прогнозов классификатора. Правильные прогнозы идут по диагонали от верхнего левого угла в нижний правый. Про это можно почитать в [данной статье](https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html).

**Отчёт о классификации**

В библиотеке Scikit-Learn уже встроена возможность создавать отчёты о производительности классификатора. Эти отчёты дают интуитивно понятное представление о работе модели.

**Заключение**

Чтобы лучше вникнуть в работу с Scikit-Learn, неплохо было бы узнать больше о [работе различных методов классификации](https://stackabuse.com/classification-in-python-with-scikit-learn-and-pandas/). После этого стоит лучше узнать о замере производительности классификаторов. Однако понимание многих нюансов в классификации приходит только со временем.