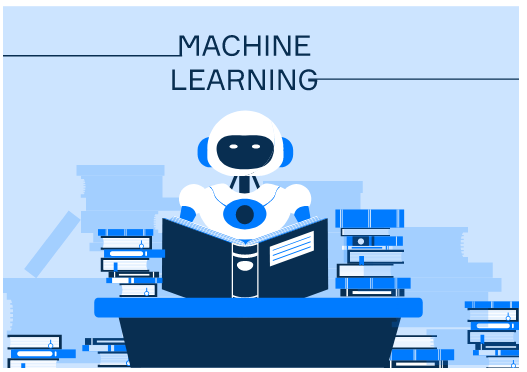
## Theory: Intro to Machine Learning

When you open your mailbox, you do not see that much of spam these days as they are automatically filtered in a separate folder. If you receive a document in an unknown language, you can easily translate it to any other language you understand in just one click. Your bank blocks your credit card based on some suspicious activity before you even realize that it was stolen. Your favorite streaming service always suggests the movies you fond of, and the special offers you get from your favorite stores are always on point.

What do all of these, and many other technologies share in common besides making our lives way easier? Right, all of them are powered by Machine Learning (ML), a subfield of Artificial Intelligence (AI), which is booming nowadays.

This topic will introduce you to the world of machine learning.



**What is machine learning?**

You might have heard the term **machine learning** a lot, but what does it actually mean? Well, the goal of ML is to create algorithms that can learn from past experiences and transfer this knowledge to new, unseen cases. Let's consider an example.

Suppose we want to create an ML-based spam filtering system. Then we will need to collect some emails received in the past, both informative and those marked as spam, and introduce them to the algorithm along with their labels. The algorithm, in turn, will try to learn how to distinguish between the two types of mails. Once the learning process is over, our model will be able to analyze new incoming emails and filter out the spam ones.

The key point here is that we do not teach the algorithm *how* the two types of emails differ from each other, we just show some examples from the past and let the algorithm figure it out on its own. Cool, right?

Note that the notion of spam may be different for different users of the same mail client. Indeed, while an email about machine learning summer school can be very interesting to you, someone studying medieval music will probably consider it spam. So if you apply the same ML algorithm to different data sets (for example, sets of emails from you and your friend), you will end up with completely different spam filters.

ML can be applied to solve many different problems. Roughly speaking, there are two main ML settings, namely **supervised** and **unsupervised learning**.

## Supervised learning

In the **supervised learning**, our goal is to learn to predict some **target** attributes from the values of other attributes, often called **features**.

If the target can take on just a few distinct values, the problem is referred to as **classification**. Spam filtering described above is a typical example of a **binary classification** problem. Each email belongs to either of the two categories: spam or regular. In a more general case, there can be more possible classes, and the problem is then called **multi-class classification**. For example, we might want to train an ML model that recognizes hand-written digits. Then, each image of a digit must be associated with one of the 10 classes, from 0 to 9.

Another example of a classification problem is **multi-label classification**. In this setting, the model is assigning not one but multiple binary labels to each example. A typical example of a multi-label classification is text categorization. There is a large number of pre-defined topics (for instance, politics, economy, sports, culture, hobby, ...), and each text can cover several of them (for instance, politics and economy, hobby and sports, and so on). The task is then to predict the correct topics for every text.

If the target attribute of the model is numerical, the problem is referred to as **regression**. An example of a regression problem would be predicting the yearly income of a person based on their education, occupation, background, and other points, or predicting real-estate prices based on the location and size of the property.

## Unsupervised learning

Another machine learning setting is **unsupervised learning**. The input data contains no information about the property that we want to predict.

A typical example of an unsupervised ML algorithm is **clustering**. Its goal is to group examples from the training data into so-called clusters, or groups, based on how similar they are. Clustering is often used in market research in order to identify similar groups of consumers based on their purchasing behavior. In bioinformatics, clustering helps categorize genes with similar functionalities and gain insight into structures inherent to populations.

Unsupervised techniques are also commonly used to solve the so-called **anomaly detection** task, the goal of which is to automatically detect suspicious events that are significantly different from the rest of the data. Anomaly detection techniques are widely used by banks to detect fraudulent transactions, in aviation, health monitoring systems, and so on.

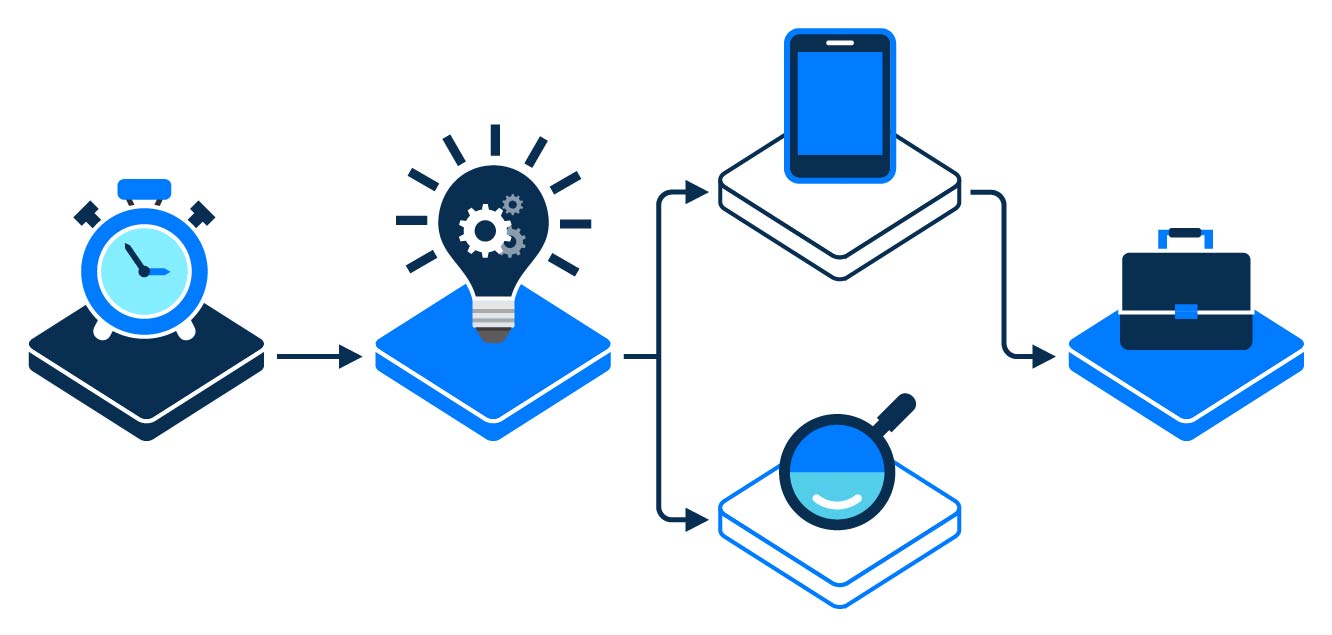
## Conclusions

* Machine learning algorithms are able to learn from collected data and apply the knowledge to the unknown cases.
* Supervised learning refers to the setting where we want to learn to predict the value of a specific target variable from the data.
* In unsupervised learning, there is no target attribute to predict. The goal is to explore the hidden structure of the data.

## Theory: Computer algorithms

**Everyday algorithms**

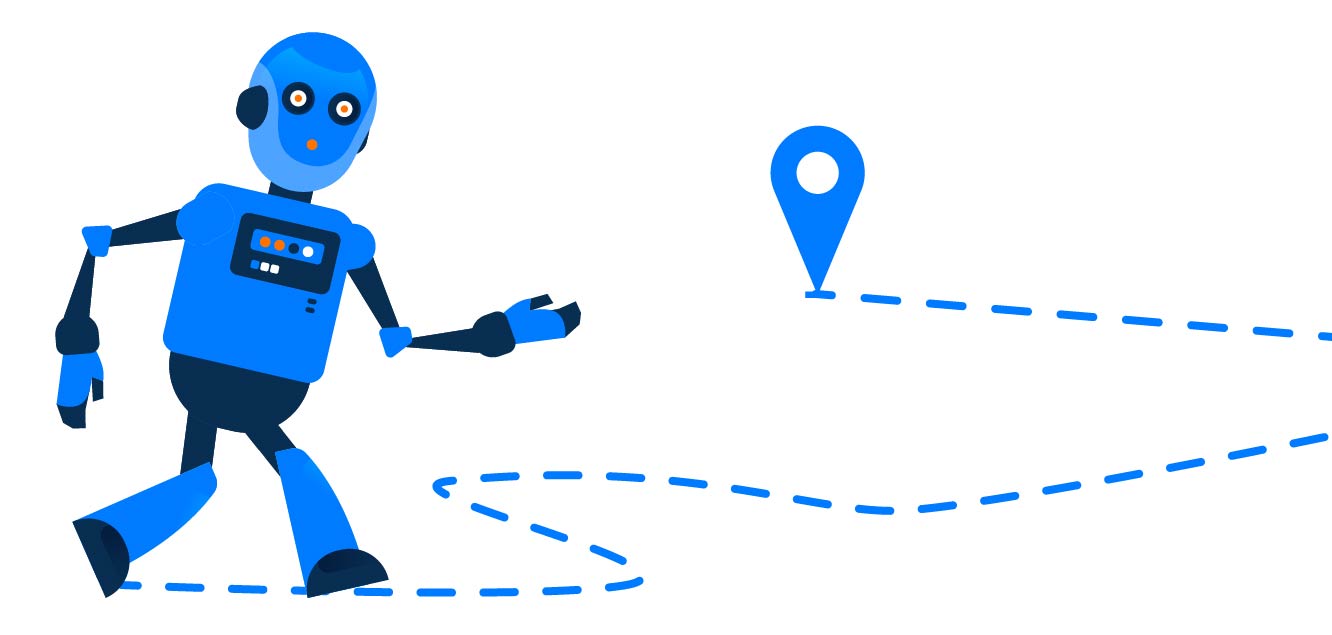
You have probably heard something about **algorithms** in real life. Simply put, it is a step-by-step sequence of actions you need to perform to achieve a useful result. It can be an algorithm for cooking a sandwich described by a recipe or an algorithm for getting dressed according to today's weather and your mood.



Among all algorithms, there is one special group called **computer algorithms.** These are ones that are usually created for and utilized by computers. In this topic, we will discuss in detail what computer algorithms are and will explain why it is important to learn them.

**Algorithms for computers**

Computer algorithms are everywhere around us. Your smartphone may guide you through a city from one point to another using a certain algorithm. Other algorithms can control the behavior of your enemies in a computer game. Services like Google or Yahoo apply sophisticated algorithms to provide you with the most relevant results when you use them to search for information on the Web. Algorithms are also used to calculate the trajectory of rockets and they even help doctors to determine diagnoses correctly.



An important difference between real-life and computer algorithms is that a computer cannot guess what we want to do. If something goes wrong or an algorithm is not clear, a human can adjust the algorithm based on their experience. Computers cannot do the same. Thus, a computer algorithm must be described precisely and unambiguously.

**Programs and algorithms**

As you may know, a program is a sequence of instructions to perform some tasks on a computer. The difference between programs and algorithms is that programs are written using a specific programming language while algorithms are usually described at a higher level than programming language statements. In other words, an algorithm is like an abstract schema, and a program can be its implementation.

All this also means that algorithms are language-agnostic: one algorithm can be implemented using different programming languages. For example, you may use Java, Python, Kotlin, or other languages to implement the same algorithm.

Programming languages usually contain implementations of some basic algorithms for solving typical problems. These algorithms are provided in standard libraries and software developers can reuse them instead of implementing a new one each time. However, to be able to use such algorithms correctly and efficiently, and to be able to understand how other developers use them, it is important to learn these basic algorithms and get familiar with how they work under the hood.

Algorithms from standard libraries cannot cover all possible problems developers can encounter. Thus, sometimes you will need to implement a solution for a problem yourself from scratch. This is another reason why it is important to learn algorithms: you need to know which one when to apply and how to implement it efficiently.

**Summary**

An algorithm is a sequence of actions you need to perform to achieve a useful result. An important group of algorithms are computer algorithms: ones created for and utilized by computers. There are several reasons why it is important to learn computer algorithms:

* software developers often encounter tasks of the same type while working on different projects. For such typical tasks, programming languages provide ready-to-use algorithms in standard libraries. To utilize these algorithms efficiently, you need to understand how they work under the hood.
* sometimes you may encounter a problem that is impossible to solve using algorithms from standard libraries. In such cases, you need to implement an algorithm yourself. To be able to do that, you need to know basic algorithmic approaches, their pros and cons, and which one to apply in a particular case.
* often you need not only to write the code yourself, but also to read the code written by other developers. If you want to understand the algorithms they might likely use, you need to know basic algorithms and algorithmic approaches as well.
* implementing algorithms might help you to improve your programming skills.

We believe that there are other reasons why learning algorithms is worth it. If you have some, don't hesitate to write them in comments :)

Good luck with learning algorithms!

TEST 1

Many modern smartphones automatically identify faces on the photos we take, and it is possible to quickly look up pictures with a particular person.

Is grouping photos with the same people on them a supervised or an unsupervised machine learning problem?

A: Unsupervised

TEST 2

Select all the statements that are true in the supervised learning setting but wrong for unsupervised learning.

Report a typo

Select one or more options from the list

Data should contain values of the target attribute.

The goal is to find previously undetected patterns in a dataset with no pre-existing labels.

Quality of a model can be evaluated by comparing the model's predictions with actual values of the target.

A model is a hard-coded solution of the problem.

One needs data to train a model.

Imagine that you are developing an activity tracking app that monitors heart rate, daily step count, calories burned, and so on. Each month, every user gets the report with an overview of their activities and some summary statistics.

You want to make the report more interesting. Your idea is to use machine learning to automatically highlight days when the user activity was abnormally high or low compared to the other days. What kind of problem is it?

Report a typo

Select one option from the list

Classification

Anomaly detection

Clustering

It's not a machine learning problem

**Correct.**

Which of the following real-world problems can be solved by classification algorithms?

Remember that in the task of classification the target is a set of few distinct values.

Report a typo

Select one or more options from the list

Predicting if a customer will buy a certain product or not based on the history of purchases.

Predicting a student's final exam score based on the scores for intermediate assignments.

Group pictures with similar dogs on them together.

Predict which celebrity is on a picture.

Predicting if a person has a certain disease based on laboratory test results.

**Correct.**

## Theory: Typical ML pipeline

11 minutes 1 / 4 problems solved

**241** users solved this topic. Latest completion was **10 minutes ago**.

You might wonder about what machine learning specialists do as part of their job. Of course, different projects mean different tasks but there are some very common steps. In this topic, we will try to highlight them so that you could have a better idea of typical machine learning tasks. This will also help you understand what it takes to become a data scientist.

## Data collection

Machine learning is impossible without data. When developing a new machine learning algorithm, experts often use publicly available datasets to benchmark your method and compare it to the ones that are already developed. You can find them on the famous [UCI repository](https://archive.ics.uci.edu/ml/index.php), as well as on [Kaggle](https://www.kaggle.com/datasets), the largest ML competition platform.

If you are working on a specific problem, let's say, in consultancy, your client (for instance, some company) can already provide you with some data they have which is relevant to the problem they want you to solve. The data can come, for instance, as Excel spreadsheets. Alternatively, you may be given access to a database from where you can load all the necessary data using SQL queries.

However, for some tasks, you might need to collect the data yourself. This can be the case, for example, if you are working on a problem or a particular application no one has worked on before. Data collection can include web scraping, which is automatically extracting and parsing the content of certain web pages, and manually labeling the data.

## Data preprocessing

Whether you use available data or collect it yourself, the datasets you end up with can be very messy. Sometimes, there can be a lot of missing values that you might need to fill in somehow. Some values can be simply wrong (imagine someone made a typo when filling in a spreadsheet and inserted 100 instead of 10.0). It is also quite common that the data is coming from different sources. In this case, it is likely that the format is different (for instance, different measure units, date formats, currencies, and so on, used in different files).

So typically the first step in any machine learning project is the data preprocessing. It includes joining data from different sources, dealing with missing values, and so on.

## Exploratory data analysis

Once the data is ready to use, it would be a good idea to take a closer look at it before starting the actual modeling part. This step is generally called exploratory data analysis (EDA). Usually, it involves making some plots and calculating some basic statistics on your data.

EDA is a crucial step as it helps you get to know your data better and identify possible problems with it that might have been left unnoticed at the preprocessing step. Besides, at this step, you gain more insights about the data and the events you will be trying to model. This is the time to test some assumptions about the data that you might have and get some ideas on which approaches can be the best to tackle the problem.

## Model selection

Now that you know your data well, you can finally start the modeling part! This is typically an iterative process — you start with training an ML model that you believe will do well on the task you are trying to solve. Then, you evaluate the model's performance and carefully investigate it, whether the model performs as expected, whether it has any pattern in the mistakes it makes, and so on. If so, it is also a good idea to devise a method to fix it.

After that, you may want to make the necessary adjustments, train a new model, analyze its performance, and repeat, until you are happy with the model you have.

## Deploying your model

Even if you built the greatest ML model in the world, it is of little use if it cannot be used by anyone else than you, or if the results you are getting cannot be reproduced.

To put it simply, at this final stage you need to make sure that your code can be run on any machine, that your implementation is robust (that is, it does not produce unexpected errors), efficient, and scales well.

The process of making your models available in production environments is called **deployment**. For example, a company you are working for can be interested, for instance, in integrating your ML solution into the software they are already using, so you will need to deploy your model so that it can provide predictions to other software systems.

It is typically Machine learning engineers who implement the built model into the production, but in a smaller company, you can be responsible for both developing and deploying ML solutions.

## Do I have to know all of this?

Above, we have described the most typical steps in a data scientist's job. It seems like you need to know a lot of things, right?

In some companies (typically, smaller ones), you might be expected to perform all of these tasks by yourself. In others, the roles can be spread across different people; data engineers responsible for data preprocessing, some do the modeling part, the others implement the solution efficiently and deploy it.

You might have noticed that not all of the steps mentioned above are immediately related to machine learning itself but rather to data or software engineering. This is true. Also, ironically, many data scientists report that most of their time is spent exactly on such engineering tasks rather than on pure machine learning (see, for instance, [this 2020 Datanami survey](https://www.datanami.com/2020/07/06/data-prep-still-dominates-data-scientists-time-survey-finds/)).

## Conclusions

* Machine learning experts typically deal with very diverse tasks at their job.
* Every ML project is different, but the most common steps are data loading and preprocessing, exploratory data analysis, modeling, and deployment.
* Depending on your job, you can be expected to perform all of these tasks, or the task can be divided among the team.

## Theory: Introduction to pandas

In practice, data is often stored in the form of a table, for example, an Excel spreadsheet, a CSV file, or an SQL database. Imagine that you need to analyze this tabular data and get some useful insights from it. Let's think of the task you will need to perform.

First, you will need to load data from different formats preserving its tabular structure and probably join several tables together. Then, to perform the actual data analysis, you will definitely want to access different columns, rows, and cells of the tables, compute some overall statistics, create pivot tables and maybe even make basic plots. Is there a tool in Python that combines all these functionalities? And the answer is yes!

This topic will introduce you to the pandas, a powerful open-source library for data manipulation and analysis. You will install the library and get an idea of its main functionality.



## Installing pandas

pandas is not included in the standard Python library, so you might need to install it separately, for example, using pip. Type the following in your command line:

pip install pandas

Note that pandas is also dependent on other libraries (for example, numpy for vector computations) that you need to be installed as well. Besides, there are many optional dependencies. For instance, if you want to use pandas data visualization functionality, you need to install matplotlib, a plotting library in Python. You install those libraries using pip, too.

Once the installation is complete, you will be able to import it in your code. Since pandas is quite a long name, it is commonly abbreviated and imported as pd:

import pandas as pd

Note that new versions of pandas are released once in a while, with new functionality and fixed bugs. So it would be a good idea to keep an eye on the updates. You can easily upgrade the version of pandas on your machine with the command: pip install --upgrade pandas.

## Data structures in pandas

The two data structures of pandas are Series (1D) and DataFrame (2D). You will get more familiar with them in the dedicated topics, but an overview is given below.

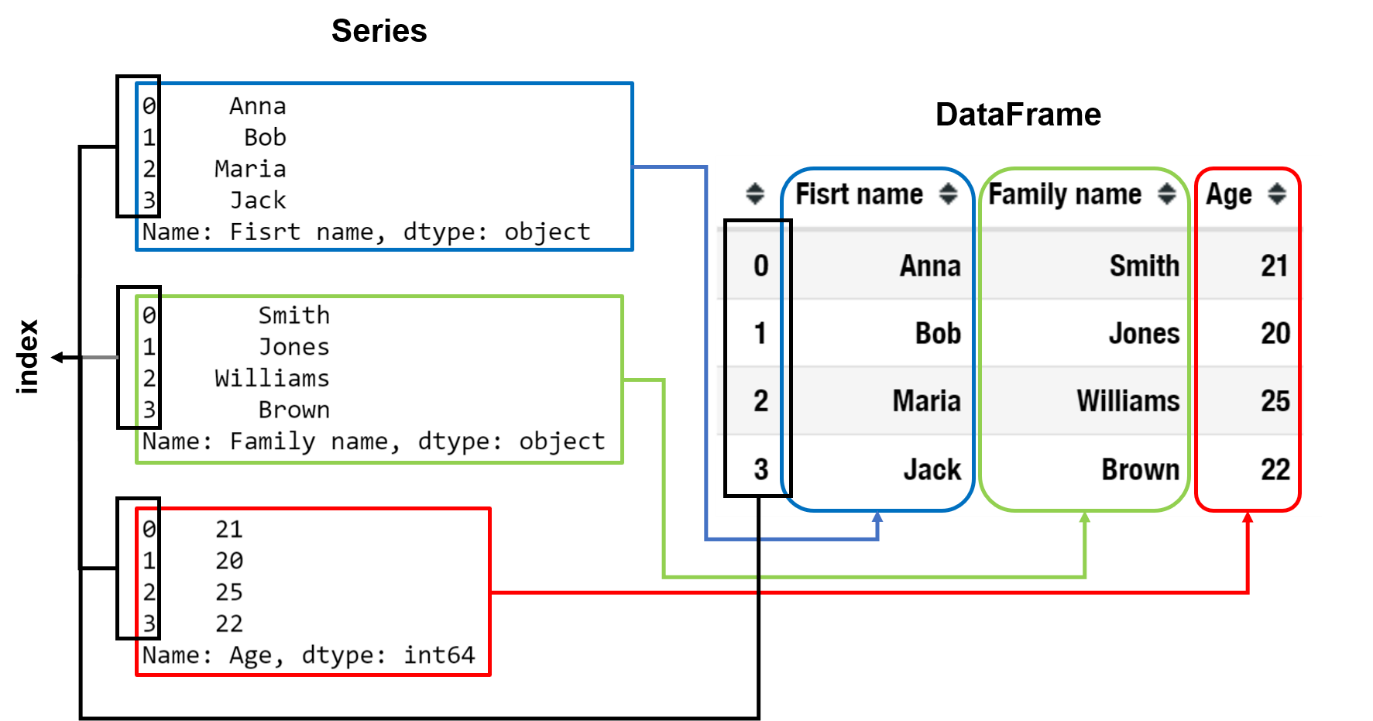
Series is a one-dimensional array that stores elements of the same data type.

Each element stored in a Series is associated with a label called **index**. By default, this index is just the sequence 0, 1, 2, ... . However, any custom values can be used. For example, when analyzing time series, timestamps are typically set as indexes.

Indexes, as well as automatic and explicit data alignment based on them, are the core of pandas.

DataFrame, in turn, is a two-dimensional data structure that is used to represent tabular data with columns of potentially different data types. You can see DataFrame as a table each column of which is a Series object. In other words, the DataFrame is a container for Series, while Series is a container for scalars.

This is illustrated with an example below. Three Series objects store names, surnames, and ages of students respectively, while the DataFrame combines this information in a single table.



Note that each row in the DataFrame is also associated with an index. It is worth mentioning that even though DataFrame is a 2D data structure, one can still use it to represent higher-dimensional data by using so-called **multi-index**, that is, assigning several indexes, or labels, to each row. However, this is rarely used in practice.

Tables represented as DataFrame slightly differ from spreadsheets to optimize computations. Series (= columns of a DataFrame) can store data of one type only to perform operations on them faster. It is also not possible to add or remove elements from a Series, which helps pandas efficiently store Series objects in memory.

## What is inside pandas?

As stated in its guidelines, pandas aims to become 'the most powerful and flexible open-source data analysis/manipulation tool available in any language'.

The name pandas is derived from **'panel data'**, a term that is used in statistics and econometrics to refer to the data sets containing observations over multiple time periods for the same individuals. This library will be helpful if you are working with tabular data, such as data stored in spreadsheets or databases. Apart from that, pandas offers great support for time series and provides extensive functionality for working with dates, times, and time-indexed data.

With the help of pandas, one can easily perform the most typical data processing steps.

In particular, the package makes it convenient to load and save data, as it supports out-of-the-box integration with many commonly-used tabular formats such as .csv, .xlsx, as well as SQL databases.

Let's look at what we can get from pandas:

* Intuitive merging and joining data sets allows one to easily combine data from different sources, while flexible reshaping tools help construct pivot tables.
* Missing values in the data are represented as NaN and can be easily handled, for example, replaced by some value, using built-in functionality.
* Besides, if you have matplotlib, a Python plotting library, installed, then you can use pandas built-in plotting functionality to make a basic plot from your data to better understand it.
* Get basic statistical information about our data literally with one line of code.
* Integrate it with other libraries for machine learning like sklearn.

Finally, pandas is open-source software, which makes it very popular in both academic and commercial domains.

## Conclusions

* pandas is a flexible tool for data analysis.
* pandas is a perfect tool to work with heterogeneous data.
* Two data structures in pandas are Series (1D) and DataFrame (2D).
* Series stores values of the same data type, while columns in a DataFrame can be of different types.

## Theory: Series

17 minutes 0 / 4 problems solved

**198** users solved this topic. Latest completion was **about 2 hours ago**.

As you already know, pandas is a popular Python library for data manipulation. This topic will introduce you to Series, a basic one-dimensional data structure in pandas.

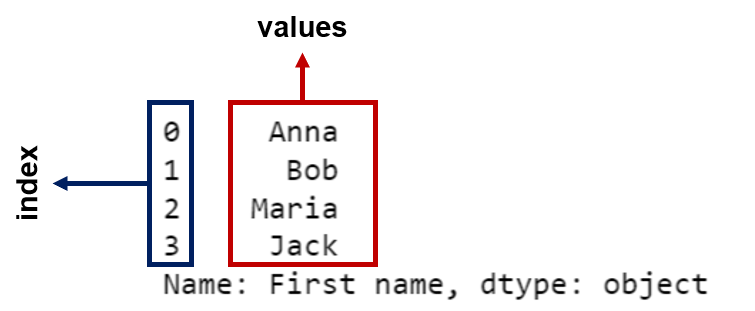
Series is a building block to the 2D data structure in pandas, DataFrame. The latter is the one you will use the most in practice. So, while it is important to have some idea of Series, in-depth knowledge of its functionality is not necessary.

Before we start, do not forget that you need to import pandas to be able to use all of its functionality, including Series. Note that traditionally, the name of the library is abbreviated as pd in the import statement:

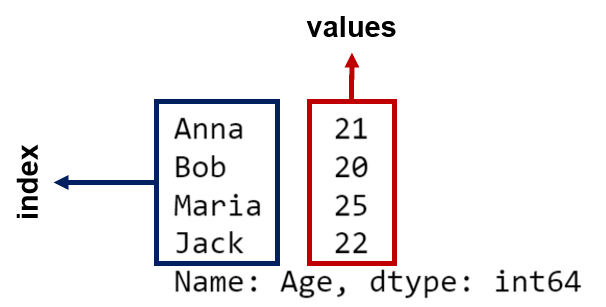
import pandas as pd

## What is Series

Series is a one-dimensional array. For example, here is a Series that stores names of the students on a machine learning class:



You can notice that each element stored in a Series is associated with a label called **index**. By default, this index is just the sequence 0, 1, 2, ... . However, any custom values can be used. For example, we can store ages of the students in a Series, and set students' names as row identifiers so that we know which student the age corresponds to:



Cool, but how do you create such a Series object? There are several ways to do so.

## Converting other data structures to Series

If the data is already stored in some other data structure, you can easily convert it to Series as shown below:

ages\_list = [21, 20, 25, 22]  
names\_list = ['Anna', 'Bob', 'Maria', 'Jack']  
  
ages\_series = pd.Series(ages\_list, index=names\_list, name='Age')  
print(ages\_series)  
  
# Anna     21  
# Bob      20  
# Maria    25  
# Jack     22  
# Name: Age, dtype: int64

Here, we convert a list of students' ages ages\_list into a Series object ages\_series. We also assign a custom index, students' names stored in the list names\_list, using the index keyword. If we fail to provide the values for the index, values 0, 1, 2, 3 would be assigned. Finally, we can also give our Series a name by specifying the optional name parameter.

Similarly, you can convert a Python dictionary into a Series object. Note that dictionary keys (students' names in the example below) will automatically become the indexes in the new Series. Cool, right?

student\_ages\_dict = {'Anna': 21, 'Bob': 20, 'Maria': 25, 'Jack': 22}  
  
ages\_series = pd.Series(student\_ages\_dict, name='Ages')  
print(ages\_series)  
  
# Anna     21  
# Bob      20  
# Maria    25  
# Jack     22  
# Name: Ages, dtype: int64

You can always change index later by modifying the index attribute of the Series:

ages\_series.index = ['A', 'B', 'M', 'J']  
print(ages\_series)  
  
# A    21  
# B    20  
# M    25  
# J    22  
# Name: Ages, dtype: int64

Of course, the length of the data and index you provide should coincide. Otherwise, an error will occur.

## Modifying a Series object

Series is **value-mutable;** you can easily change the values stored in a Series, for example by accessing them by index. To illustrate this, let's update Jack's age:

ages\_series['Jack'] = 23  
print(ages\_series)  
  
# Anna     21  
# Bob      20  
# Maria    25  
# Jack     23  
# Name: Ages, dtype: int64

However, a Series object is **size-immutable**, once it's created, no elements can be added to or removed from it. It is made on purpose, to efficiently store Series in memory.

But what should you do if you need to add or drop some values to/from a Series? No worries, appending, and removing elements is still possible. However, the result of these operations will be a new Series object.

For example, let's remove Maria's record from our Series. This can be done with the drop method.:

new\_ages\_series = ages\_series.drop(index='Maria')  
print(new\_ages\_series)  
  
# Anna    21  
# Bob     20  
# Jack    23  
# Name: Ages, dtype: int64

Note that the original Series remained unchanged:

print(ages\_series)  
  
# Anna     21  
# Bob      20  
# Maria    25  
# Jack     23  
# Name: Ages, dtype: int64

If you want the returned Series to be automatically assigned to the original one, you can specify the optional inplace parameter to be True :

ages\_series.drop(index='Maria', inplace=True)  
print(ages\_series)  
  
# Anna    21  
# Bob     20  
# Jack    23  
# Name: Ages, dtype: int64

To add new records to Series, one can explicitly specify the value for the new index. Let's add Maria's record back to the ages\_series:

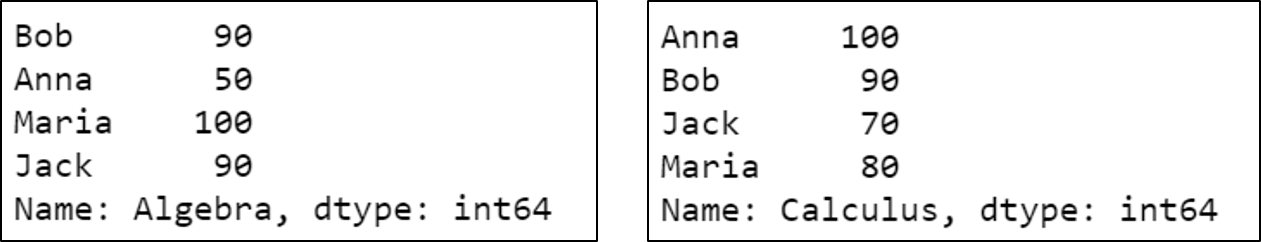
ages\_series['Maria'] = 25   
print(ages\_series)  
  
# Anna     21  
# Bob      20  
# Jack     23  
# Maria    25  
# Name: Ages, dtype: int64

Note that this syntax is introduced for convenience. Since Series is size-immutable, behind the scenes, a new Series with the newly added element will be created and automatically assigned to the original one.

## Operations on Series

The key feature of pandas is that operations between several Series automatically align the data based on the index.

Let's imagine we have two Series, algebra and calculus, containing students' exam results for Algebra and Calculus courses respectively:

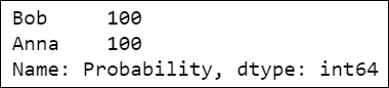


Suppose we want to compute the average score students got for the two exams.

average = 0.5\*(algebra + calculus)  
print(average)  
  
# Anna     75.0  
# Bob      90.0  
# Jack     80.0  
# Maria    90.0  
# dtype: float64

Note that the order of the students in the two Series, algebra and calculus, is different, and yet the averages are computed correctly. Very convenient!

Alright, but what happens if some indexes from one Series don't exist in the other one? Let's imagine there was also the third exam on Probability, but only Anna and Bob took it. The results are stored in the probability Series:



What will happen if we try to compute the average of the three Series now?

average = 0.33\*(algebra + calculus + probability)  
print(average)  
  
# Anna     82.5  
# Bob      92.4  
# Jack      NaN  
# Maria     NaN  
# dtype: float64

As you can see, the result of an operation between unaligned Series has the **union** of the indexes involved. If a label is not found in one of the operands, the result will be marked as a missing value NaN.

Writing code without doing any explicit data alignment gives a lot of flexibility in data analysis. The integrated data alignment in pandas is what sets the library apart from the majority of related tools for working with labeled data.

## Conclusions

* Series is a one-dimensional data structure in pandas.
* Series store values along with their labels called indexes.
* Series is value-mutable but size-immutable: one can modify values stored in it, but cannot add new values to or remove values from it.
* When performing operations on several Series objects, they are automatically aligned on the index.

<https://hyperskill.org/learn/step/10421>

This topic is an introduction to scikit-learn, arguably the most popular ML library in Python.



## Installing scikit-learn

Since scikit-learn is not a part of the Python standard library, you might need to install it on your machine before you can start using it. One way to do so is by using pip. Run the following command in the command line:

pip install scikit-learn

Note that the scikit-learn requires such Python libraries as numpy and scipy. If you have not installed them yet, depending on your operating system, these libraries will be either automatically installed along with scikit-learn, or will need to be installed manually beforehand. This can be easily done with pip as well. Alternatively, you can automatically install scikit-learn along with all its dependencies by running the following command:

pip install scikit-learn[alldeps]

Also, note that scikit-learn is constantly being updated, so it is a good idea to check for new versions every now and then. To upgrade the current version installed on your machine to the latest release with pip, run

pip install --user --upgrade scikit-learn

Once the library is installed, you can import it in your code and start using it. Importing the whole scikit-learn is typically overkill. Normally, you will be importing only the modules you are intending to use. The typical import statement will look like this:

from sklearn.module\_of\_interest import functionality\_of\_interest

To import the whole module, use \* :

from sklearn.module\_of\_interest import \*

You might have noticed that we use the name sklearn, not scikit-learn, in the import statements above. Well, the former is just a conventional abbreviation of the name of the package. You can use either to install the package, but only the abbreviated version can be used for importing it. In our text, we will use the short sklearn from now on, since this is how most users refer to the library in practice.

## What's good about sklearn?

There are a number of things that make sklearn this popular. Here are some of them.

To begin with, sklearn can be used for various tasks within the data mining process, such as data pre-processing, training machine learning models, and model selection. It is really convenient for machine learning practitioners to have this diverse functionality in one place. Besides, sklearn integrates well with other popular Python libraries useful for data analysis, for example, pandas for storing and manipulating data, numpy for vector computations and matplotlib and seaborn for data visualization.

Another advantage of sklearn is that this library arguably features all widely-used algorithms for the most common machine learning problems such as classification, regression, clustering, etc. Whichever algorithm you want to use with your data, most likely, you will find it inside sklearn. Convenient, right?

Part of its success is also due to a simple intuitive interface. Fitting any ML model in sklearn is very simple, even if you know almost nothing about machine learning. Moreover, the interface is consistent, so your code will look more or less the same, whether you are solving a classification, regression, or a clustering problem. The same goes for data preprocessing techniques.

Apart from that, sklearn has an excellent [user guide](https://scikit-learn.org/stable/user_guide.html) abound with explanations and examples.

And, finally, sklearn is an open-source library, meaning that everyone can view the [source code](https://github.com/scikit-learn/scikit-learn) and contribute to it. It also means that sklearn is free to use for both personal and commercial purposes, which is why many companies chose it over other expensive data analysis software.

## Is sklearn really so perfect?

By now you are probably already convinced that sklearn is a wonderful tool. Well, you should keep in mind that the library unfortunately also has some considerable weaknesses.

The major downside is speed. The sklearn implementation of the machine learning algorithms cannot compete in speed with their lower-level implementations (for example, in C++). So, if speed is an important factor for your project, or if you are working with really big datasets, you may want to use other tools. However, sklearn could still be useful for quick proof of concept on a smaller portion of the data.

Besides, you should be aware of the fact that sklearn code might contain bugs. That is natural. After all, this library is created by people, not robots. Be critical about the results you get and make sure everything works as expected. A good way to check this is to come up with some toy examples where the results of the analysis are known to you, and run your code on them. If you think you found a bug, you can [report it using the Bug Tracker](https://scikit-learn.org/stable/developers/contributing.html#submitting-a-bug-report-or-a-feature-request), so that sklearn contributors could fix it, or even fix it yourself.

Also, always study the documentation carefully. The default behavior of some methods in sklearn implementation can be different from what you expect.

## Conclusions

* scikit-learn is a Python library for data mining and machine learning.
* sklearn is short for scikit-learn and should be used to import the package.
* It has rich functionality and is quite easy to use.
* It's a free, open-source library.
* The scikit-learn implementation may contain bugs.

[**https://hyperskill.org/learn/step/10430**](https://hyperskill.org/learn/step/10430)

## Theory: DataFrame

You are already familiar with Series, a one-dimensional data structure in pandas. In this topic, you will learn about another key pandas data structure, the DataFrame.

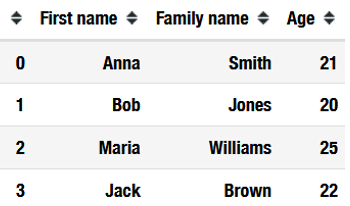
Don't forget to import the pandas library:

import pandas as pd

**What is DataFrame**

DataFrame is a table with columns. Each element of a Series, each row of a DataFrame is labeled with an index.

Here is an example of a DataFrame object students that stores information about four students:



This DataFrame has three columns, namely 'First name', 'Family name', and 'Age'. The four rows are labeled with indexes 0, 1, 2, 3.

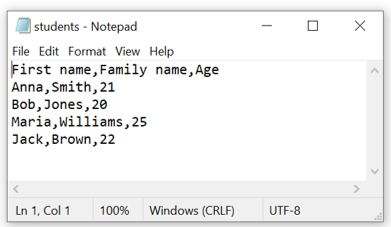
Alright, so how to create it?

**Creating a DataFrame: reading data from a file**

Often you want to use the data from a file that is stored on your computer. pandas has functions that allow you to do it.

One of the most popular text formats is .csv, which stands for comma-separated values. This format can store tabular data; each row in a file represents a row in a table, and values corresponding to different columns are separated by commas.

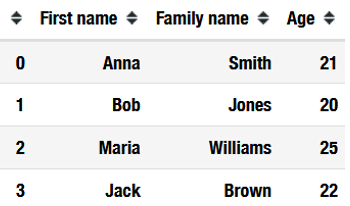
Suppose the data about the students is stored in a students.csv file:



To transfer a students DataFrame, you can use a read\_csv() function from pandas. This function takes the path to the file and some additional arguments that can be helpful to read the data correctly.

If we want to read the file as it is, we can simply write:

students = pd.read\_csv('students.csv')  
students

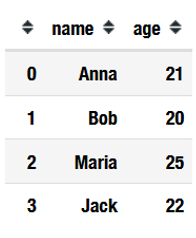


We won't list all additional parameters that read\_csv can take here, but here are the most essential ones:

* *sep* — delimiter that is used with (default ',').
* *header* — row number that stores the column headers. By default, pandas tries to infer them from the first row.
* *names —* a list of column names. If you want to use other column names, set *header=0* and pass a list of new column names with *names*.
* *index\_col* — columns in your file that are used as row labels of the DataFrame. It's set to None by default and the row numbers are used as indexes.
* *usecol —* a list of column numbers or column names to be read. By default, the dataframe reads every column.

Let's read the same file again, but this time we only use the first and the last column, giving them different names:

students = pd.read\_csv('students.csv', usecols=[0,2],header=0, names=['name', 'age'])  
students



You can use the read\_excel() function to read the data from a spreadsheet. It has a similar interface but it reads .xlsx files. To read a JSON file, use read\_json() instead.

**Creating a DataFrame from other data structures**

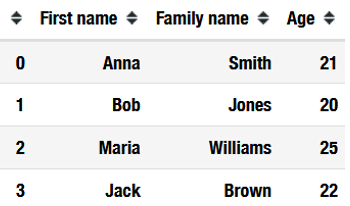
It's also possible to convert other data structures, e.g. dictionaries, lists, or numpy arrays, to a DataFrame object. You need to pass the data to the DataFrame constructor.

For instance, suppose you have a nested list containing information about students:

students\_list = [['Anna', 'Smith', 21],  
                 ['Bob', 'Jones', 20],  
                 ['Maria', 'Williams', 25],  
                 ['Jack', 'Brown', 22]]

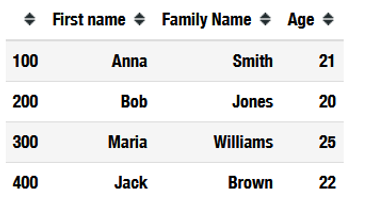
We can easily turn it into a DataFrame:

students = pd.DataFrame(students\_list, columns = ['First name', 'Family Name', 'Age'])  
students



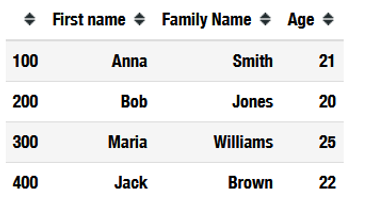
We could additionally specify the index instead of the default 0, 1, 2, ... with the index argument. Let's try that:

students\_number = [100, 200, 300, 400]  
students = pd.DataFrame(students\_list,   
                        columns = ['First name', 'Family Name', 'Age'],  
                        index = students\_number)  
students



Creating a DataFrame from a nested dictionary, index and column names will be automatically inferred from the dictionary keys. Take a look at the example:

# This is a nested dictionary representing the students table  
students\_dict = {'First name': {100: 'Anna',   
                                200: 'Bob',   
                                300: 'Maria',  
                                400: 'Jack'},  
                   
                 'Family name': {100: 'Smith',   
                                 200: 'Jones',  
                                 300: 'Williams',  
                                 400: 'Brown'},  
                 'Age': {100: 21,   
                         200: 20,   
                         300: 25,  
                         400: 22}}  
  
students = pd.DataFrame(students\_dict)  
students



**First glance at the data**

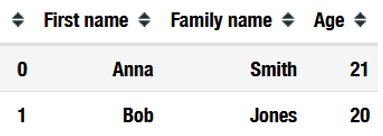
Imagine that you've just loaded your data into a DataFrame and you can't wait to start exploring it.

To check how many rows and columns a frame has, you can access the shape attribute. It contains a tuple with two values, the dimensions along the two axes. For example, in our students DataFrame, there're four rows and three columns:

students.shape  
# (4, 3)

You might also want to take a look at your data. The DataFrame may be too large to print it out. In this case, use head() and tail() methods. They will print the first or the last five rows of the DataFrame respectively. If you want a different number of rows displayed, just specify it in the brackets. Let's print out just 2 first rows:

students.head(2)

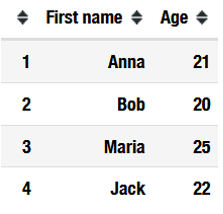


You can also access each of the DataFrame's columns separately by putting the name of the column in the square brackets after the name of the DataFrame. Note that each column of a DataFrame is a Series:

students['Age']  
# 0    21  
# 1    20  
# 2    25  
# 3    22  
# Name: Age, dtype: int64

If you need to access several columns at once, just put their names on a list. Let's take a look at the first and last columns only. Note that a resulting table is a DataFrame object:

students[['First name', 'Age']]



Note that if you want to get a single column from a DataFrame as another DataFrame object but not Series, you should put the name of the columns in double square brackets:

students[['Age']]



If you need to access the data in a particular column itself without the indexes, you can use the values attribute. Then, you'll get a NumPy array instead of a Series or a DataFrame:

students['Age'].values  
# array([21, 20, 25, 22], dtype=int64)  
  
students[['First name', 'Age']].values  
# array([['Anna', 21],  
#        ['Bob', 20],  
#        ['Maria', 25],  
#        ['Jack', 22]], dtype=object)

**Saving a DataFrame to a file**

Once you are done with a DataFrame, you can easily save it to a file on your computer. Just like with reading data from different file formats, pandas implements methods to save the DataFrame in various formats: to\_csv, to\_excel and to\_json. They are alike so let's write a table in a .csv file.

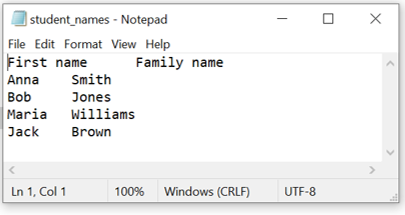
to\_csv() method can take a lot of arguments, but the most important ones are the following:

* path to the file where the DataFrame should be stored.
* *sep —* delimiter to use (default ',')
* *header —* stores the column names (default True). You can also pass a list of column names different than the ones that the DataFrame has.
* *index —* whether to write index (default True)
* *columns* — columns to write. By default, all columns are used, but you can pass a list of column names to use only part of them.

If we want to write the first and the second columns of the students DataFrame to the *student\_names.csv* file, without index and with tabulation as a delimiter. This can be done as follows:

students.to\_csv('student\_names.csv', sep='\t', columns=['First name', 'Family name'], index=False)

Here is the resulting file:



**Conclusions**

* DataFrame is a two-dimensional data structure. It's useful to store tabular data with columns of different data types.
* Row names in a DataFrame are called indexes.
* Each column of a DataFrame is a Series.
* A DataFrame can be created by reading data from a file (e.g., .csv), or by converting other data structures into a DataFrame.
* head() and tail() methods allow one to see the first and the last couple of rows of a DataFrame.

## Theory: Introduction to classification

16 minutes 0 / 4 problems solved

**105** users solved this topic. Latest completion was **1 day ago**.

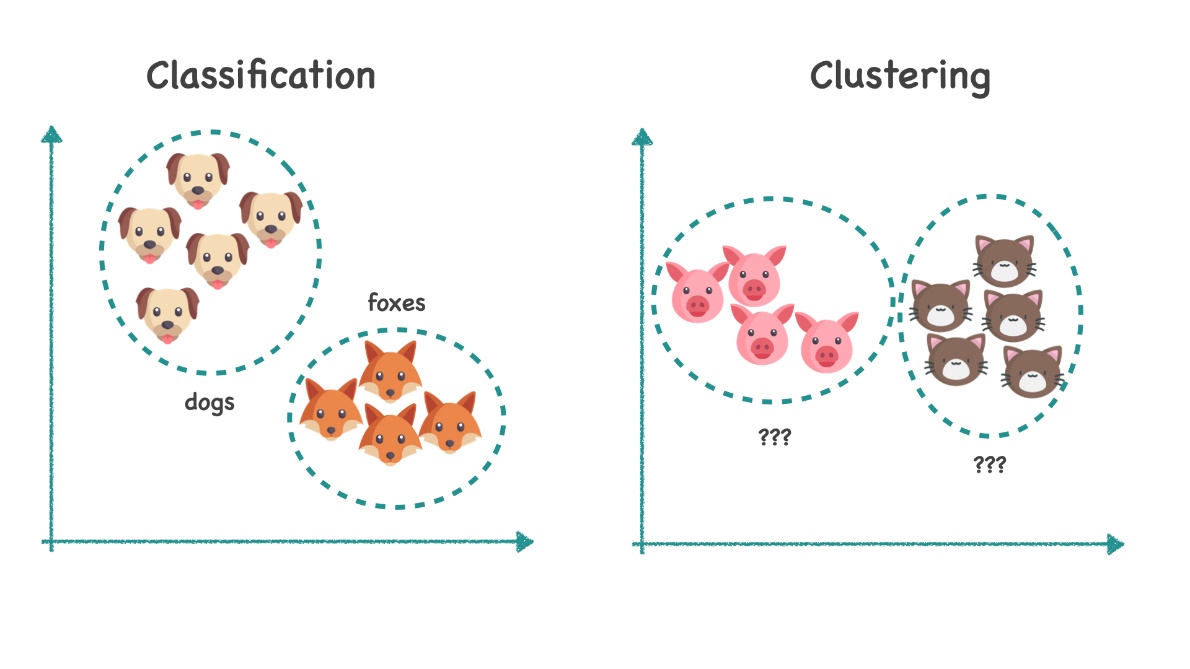
Classification can be helpful, the primary technique here is to place a label for each sample. In terms of machine learning, the classification problem can be formulated to predict finite and descriptive sets. Basically, the classification is a supervised learning task of assigning a label to a sample.

## Classification in real life

Classification is what we do every day in our heads. When we go to a supermarket to buy a watermelon, the main task is to determine whether a watermelon is good or not (a binary classification) based on tapping, weight, aroma, and sound.

Some tasks are more complex and ambiguous. In some cases, classes can overlap; when choosing a movie for the evening, one can choose a movie that is both comedy and fantasy. It is also worth mentioning that the data structure could be nested. This is a situation when classes are arranged in a hierarchy and are nested relative to each other. This is also a classification task, for example, nested word hierarchies — "class" in "classification".

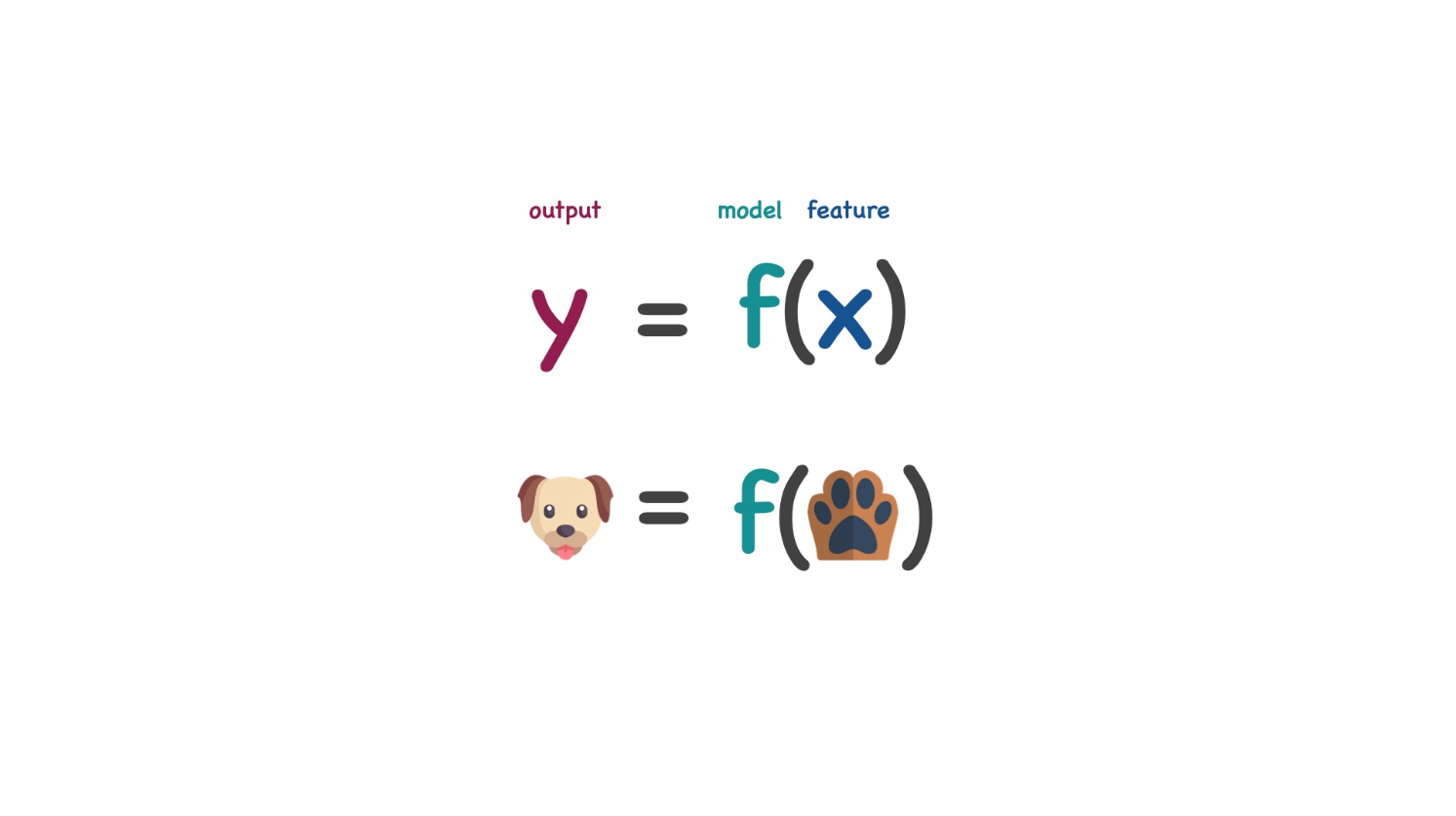
A quick note on the difference between classification and clustering tasks. Classification implies that we already know the labels for some of the data. Clustering is a search for structure within the data without knowing the exact answer.



Mostly you will have to deal with **binary classifications**, **multiclass classifications**, and **multilabel classifications**. Binary classification implies a problem where a label is binary: yes or no, cat or dog, give a loan or request additional verification. Multiclass classification solves a problem with many classes, but each sample can only be in one class. For example, "apples, pears or plums".Multilabel classification assigns several labels at once, this task is very popular in text classification: drama, history, and comedy can be genres of one single book.

## The formal explanation

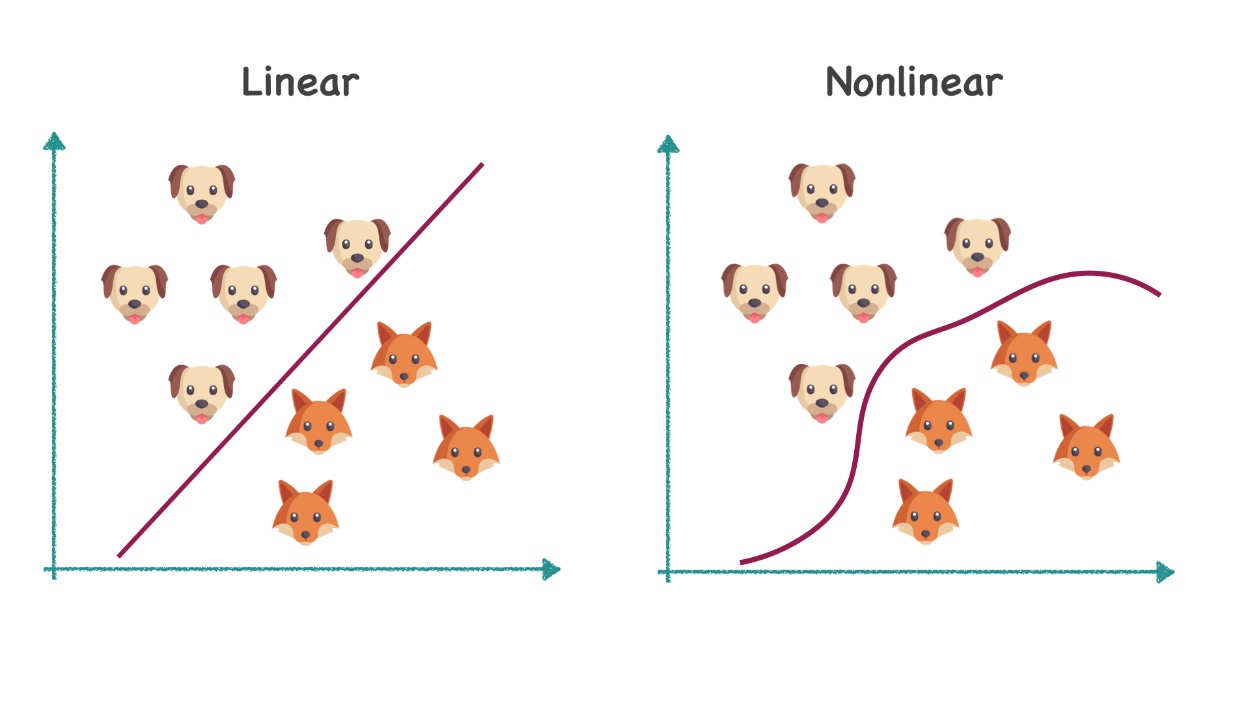
In general, any task of supervised learning can be written like this:



f(x)f(x)f(x) is the pre-selected model, where xxx is input data, yyy output data. Our challenge as a machine learning engineer is to find parameters to the model θ\thetaθ that minimize the **loss or error function**. For example, the model should minimize the number of incorrectly classified pairs in the training set. Choosing the right model, the actual error, and optimizing the model parameters is the art of machine learning.

## Basic ML algorithms for classification

We can divide all algorithms into linear and nonlinear. In the context of machine learning, linear models create a multidimensional **dividing plane** between classes. In contrast, nonlinear models create some **complex surfaces**.



How to choose the right training model so that it could produce results quickly and efficiently? First of all, it depends on the data. Features can be dependent or independent, linearly divided or divided by curves of a more complex order. So, it is so essential to understand the context of the problem before solving it. Knowing this will allow you to choose the best model.

First of all, let's give a quick overview of **linear models.**

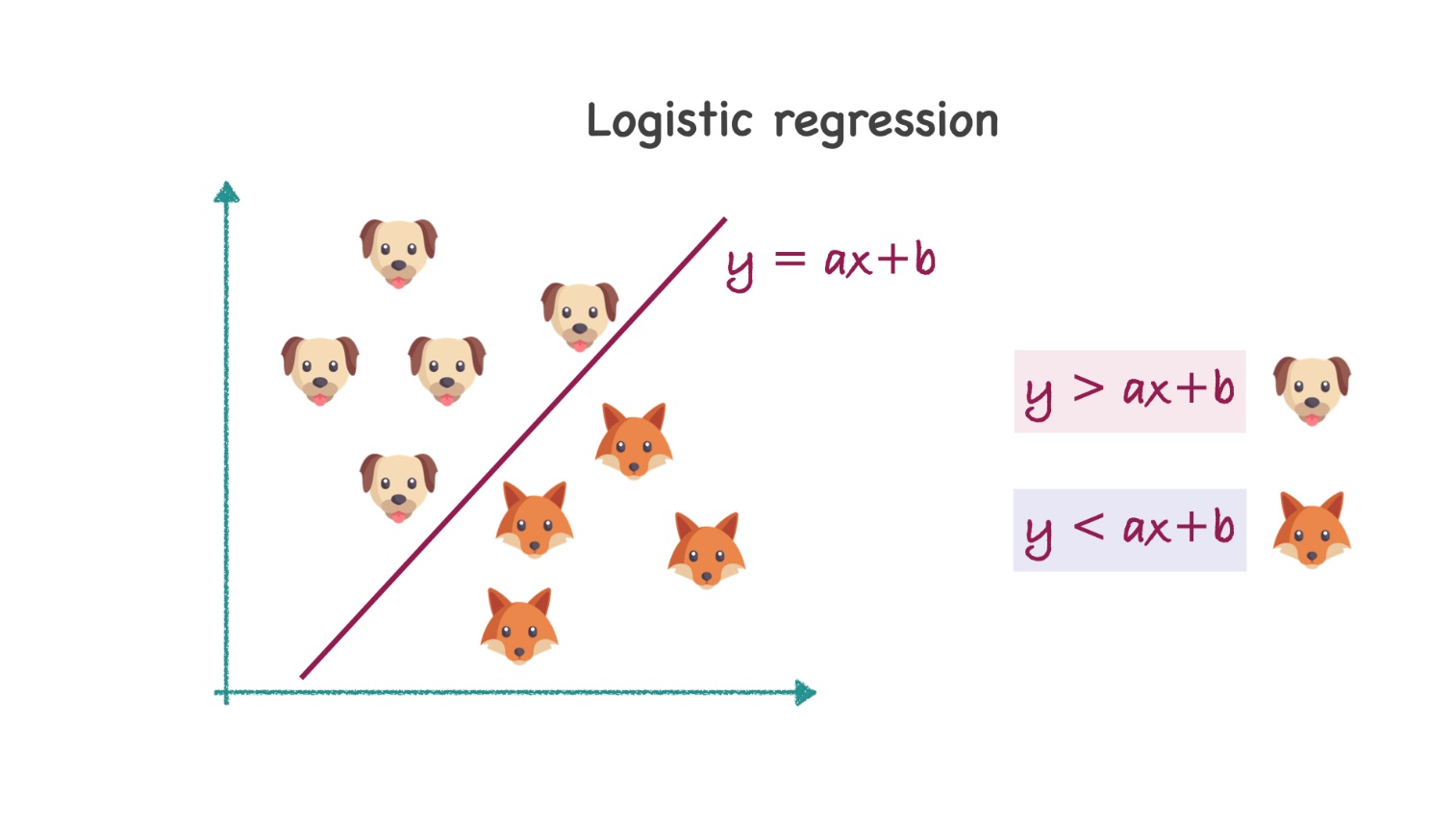
Naïve Bayes is one of the earliest and oldest email spam search models that use the Bayes rule:

P(y∣x⃗)=P(y)P(x⃗∣y)P(x⃗)P(y|\vec x) = \frac{P(y)P(\vec x|y)}{P(\vec x)}P(y∣x

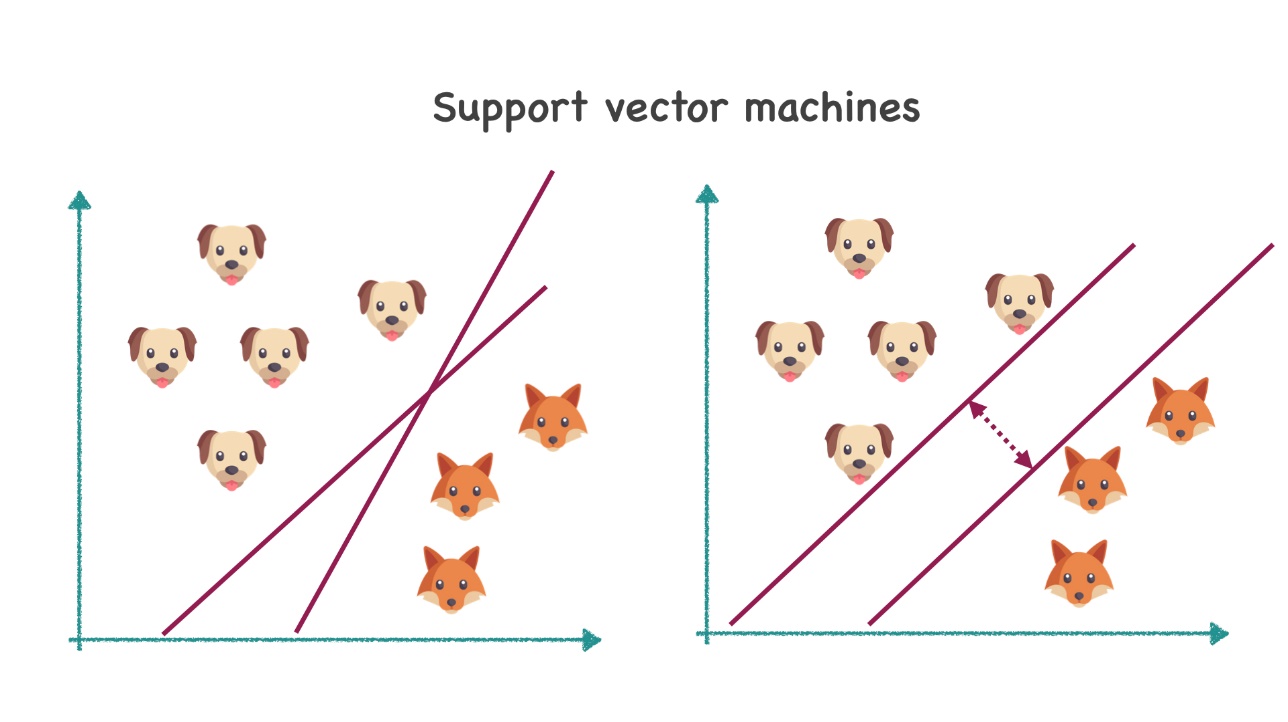
)=P(x)P(y)P(x

∣y)​

This algorithm considers each feature as independent from other features but dependant on the output. Based on the formula, the algorithm predicts the probability of class.  
  
Logistic regression is one of the main regression-based machine learning algorithms. Its main idea is that there is some linear dividing surface, on the opposite sides of which we have two different classes. The main task is to find the weight of each feature that would define this surface.



The Support Vector Machines (SVM) algorithm asks a more fundamental question: which of the lines is the most optimal in the figure?



The most optimal solution here would be the hyperplane that maximizes the margin between the two classes.

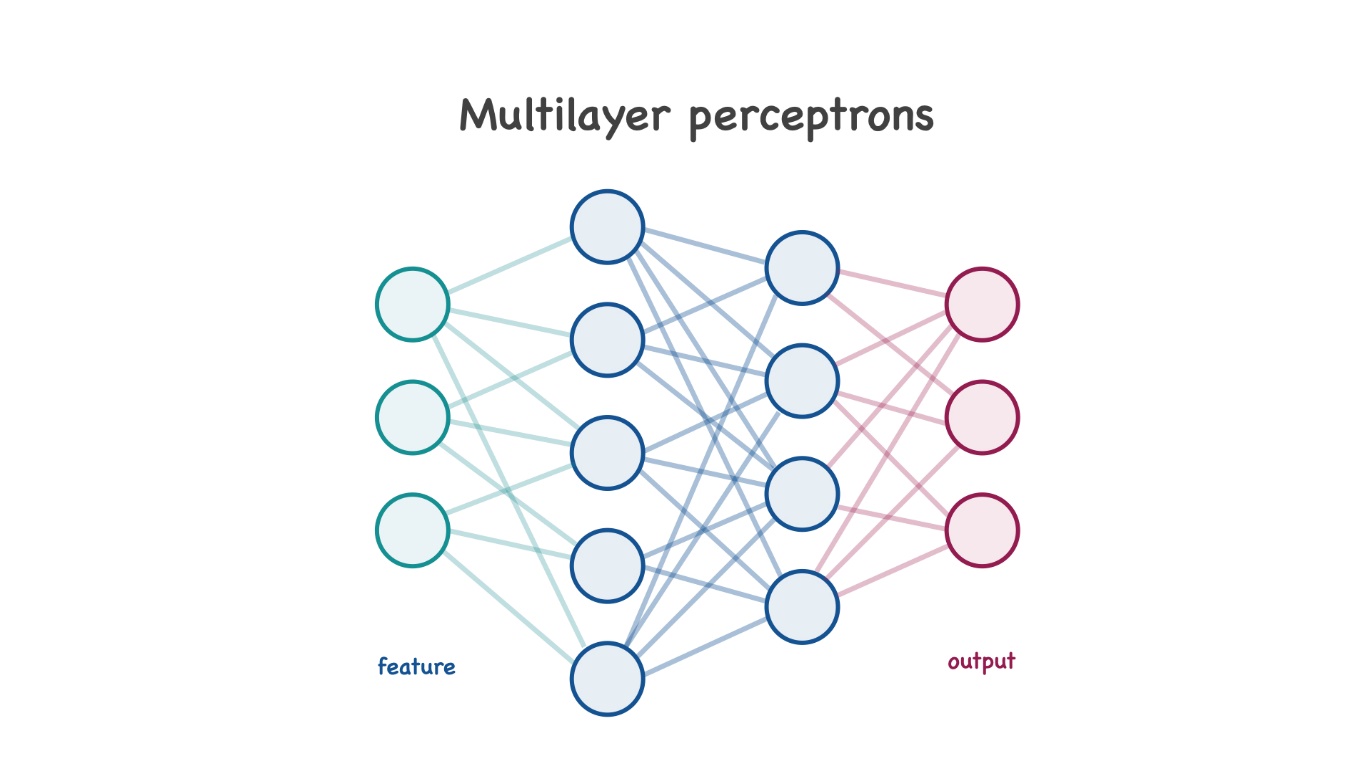
The advantages of linear models are that they are relatively fast and can be easily interpreted. But they do not work with the missing data and cannot restore nonlinear dependencies between features, which are often the case.

## Non-linear algorithms

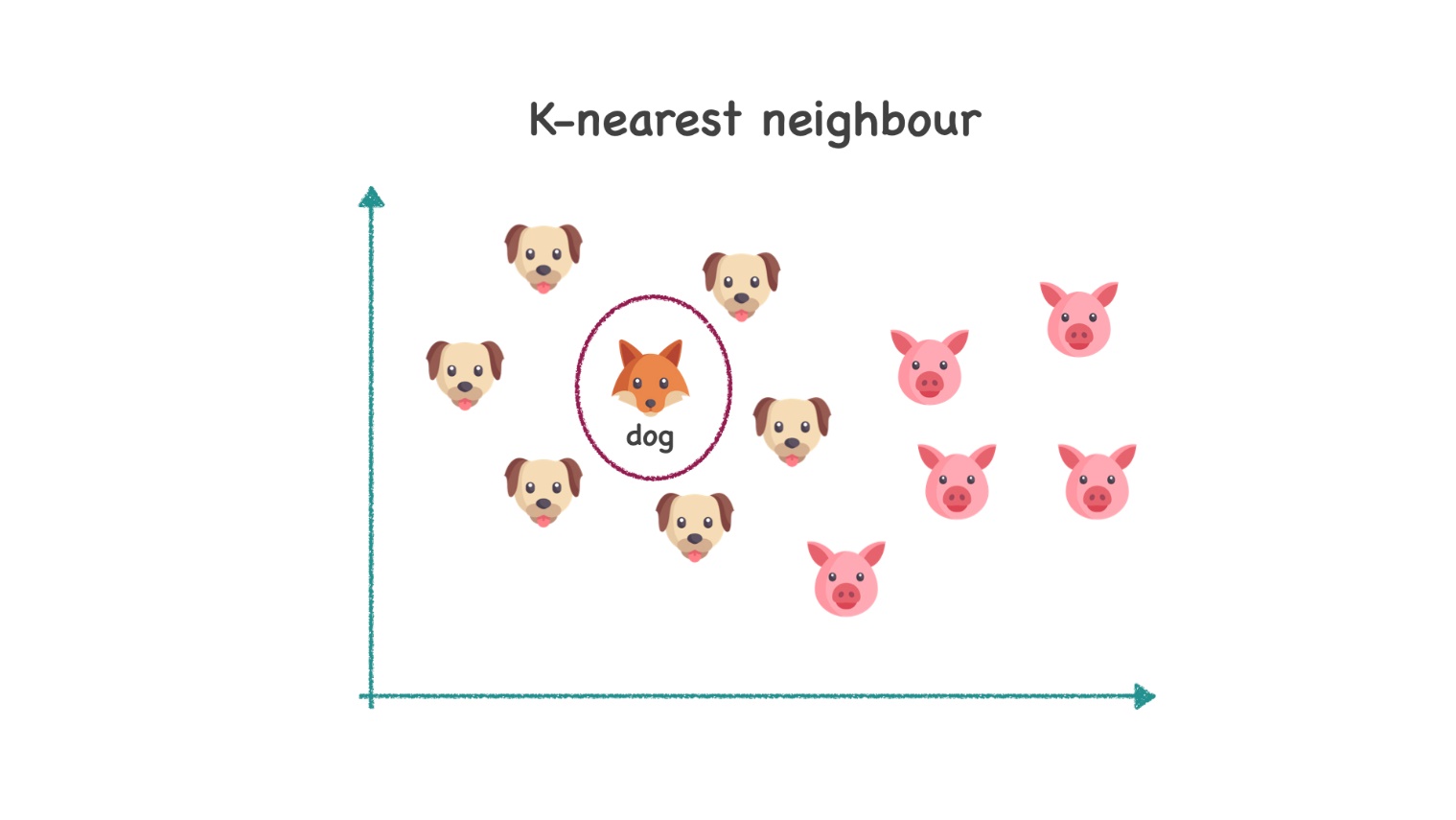
Nowadays, artificial neural networks (multilayer perceptrons) are one of the most popular tools for solving machine learning tasks. So, it is no surprise that they can be used for classification.

In the first approximation of a single neuron layer, this algorithm is closely related to logistic regression. The activation function adds nonlinearity to this system. This approach allows us to recover complex functions and find deeply hidden structures within the data. This is why they're so popular.

As for the disadvantages, it is worth noting that neural networks take a very long time to train in comparison with other models and require a larger amount of data.

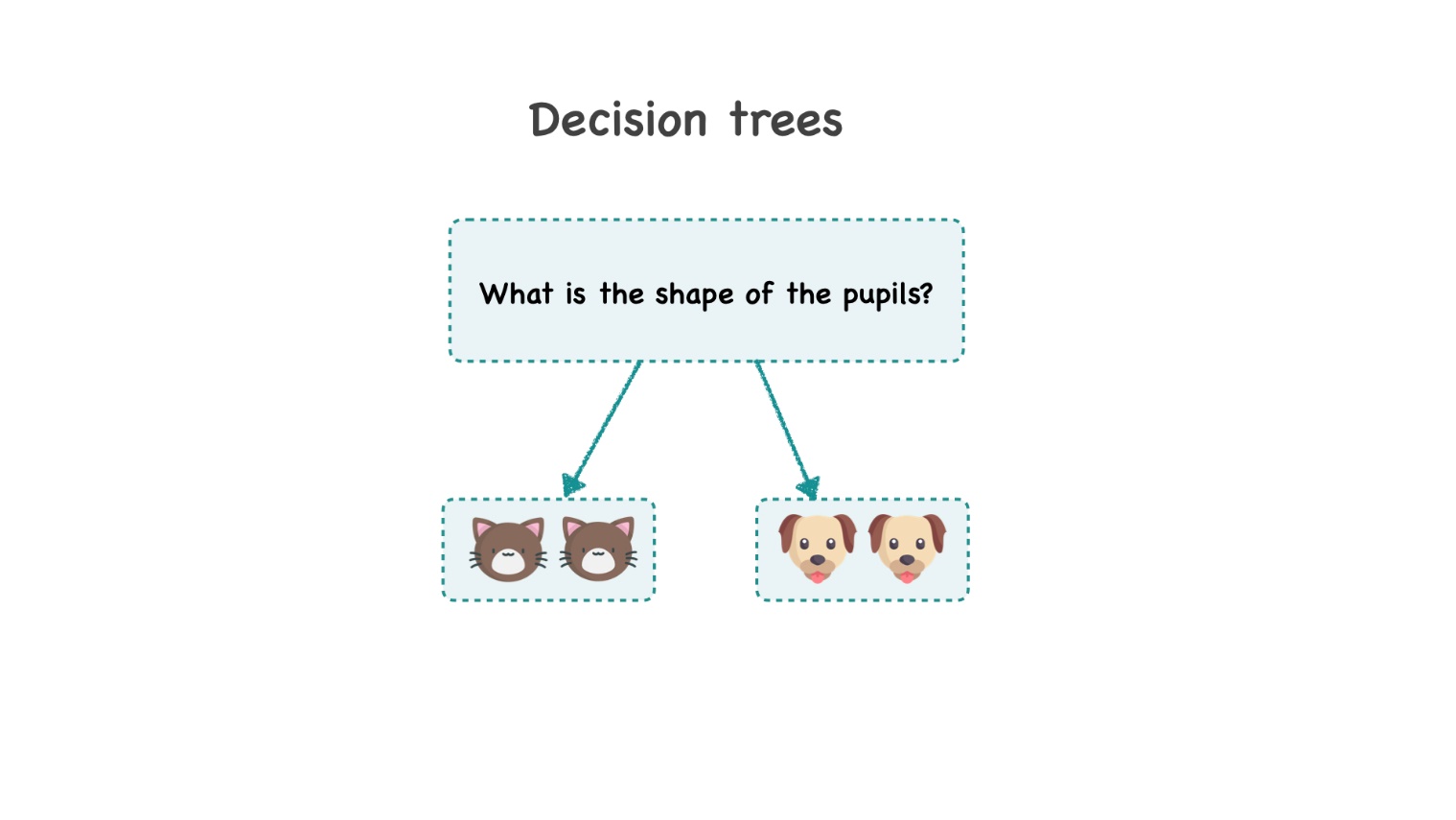


Another popular algorithm is the K-nearest neighbors (k-NN). This algorithm's idea is simple, and as the saying goes — tell me who your friend is (or rather your neighbor), and I will tell you who you are. In other words, we look at the nearest neighbors of the sample we want to predict and say that it belongs to the same class.



The main disadvantage of this model is that it rarely considers the entire training set.

Another versatile algorithm is a decision tree. It divides the data into subsets according to features until an exact solution is given in each branch.



## Summary

In this topic, we've discussed what a classification problem is and how it differs from a clustering problem. Classification is the task of determining the label for a sample, where labels come from a finite number of options. The classification task can be binary, multiclass, or multilabel. The basic algorithms can be divided into two types: linear and non-linear, which respectively create planes or complex surfaces.

If you feel a little overwhelmed with this amount of information, do not worry. In the following topics, we will discuss each of these methods separately.

Report a typo

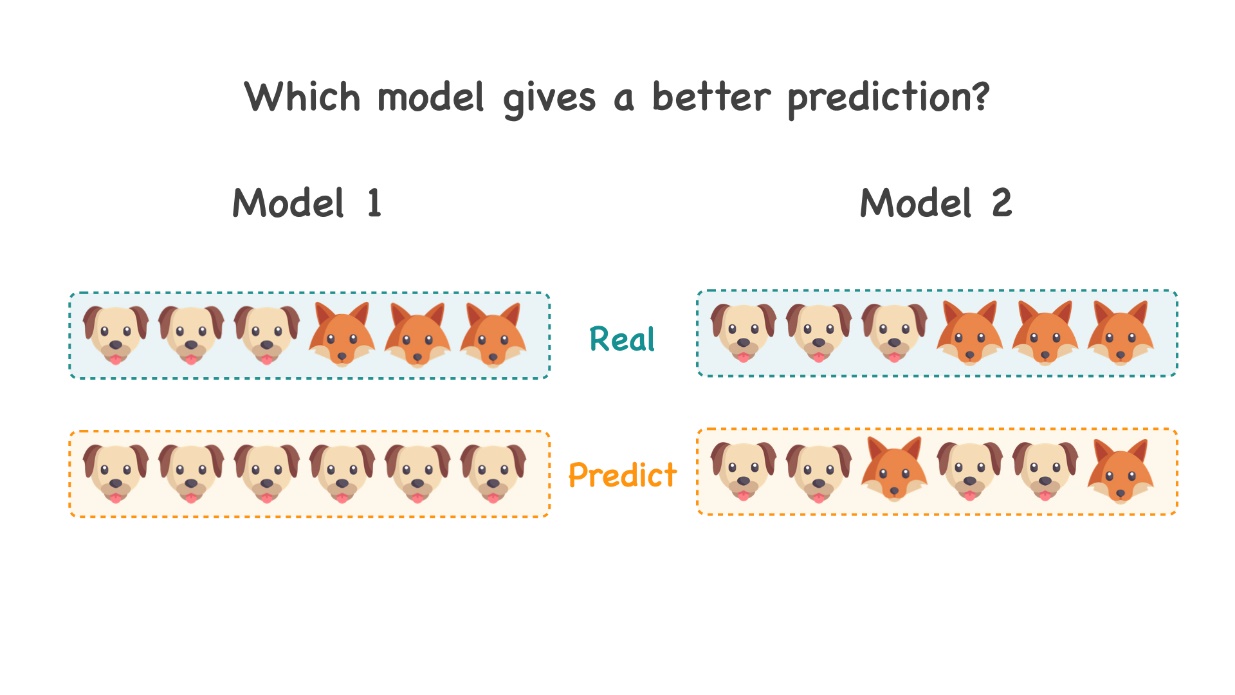
10 users liked this theory. 0 didn't like it. **What about you?**

**Theory: Classification performance metrics**

41 minutes 0 / 4 problems solved

**51** users solved this topic. Latest completion was **2 days ago**.

You have already done a lot of work at this stage — you prepared the data, you figured out the libraries, you built a couple of models. But how do you determine which model is better? Compare the following two model outputs before answering this question: "Which model gives a better prediction?"



Any ideas? As you can see, the question of the model's quality is not as trivial as it might seem at first glance. In this topic, we are going to analyze various widespread metrics used in binary classification problems and look at cases where each metric performs best.

**The basics**

A **metric** reflects the relationship between two lists — the real data and the prediction. It is important to note that none of the metrics is related to the model. A metric is introduced to our research after fitting, so it is a useful tool for performance evaluation.

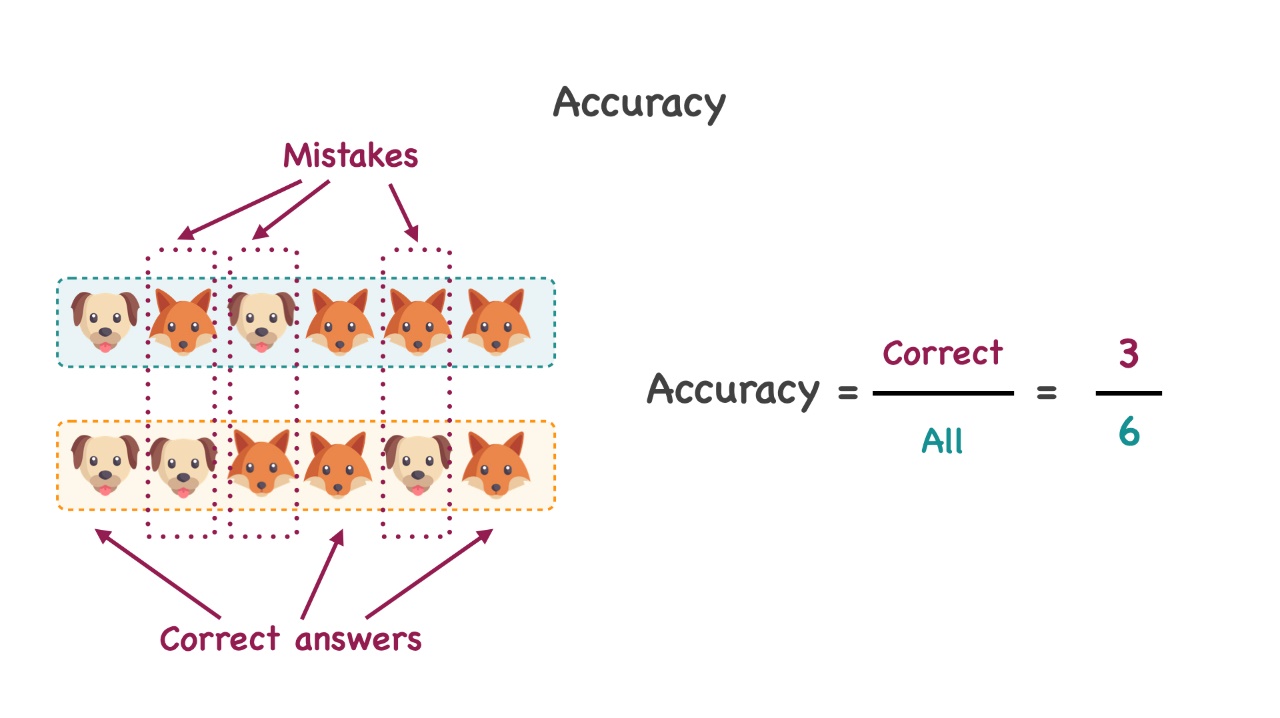
Let's think about errors and their profoundness. Some classification problems infer two classes that are equal in importance and quantity. For example, what is in the picture, a fox, or a dog? But most often, in real life, we meet with complicated problems, where errors can be multiple in their nature. For example, finding a fraud banking transaction. It's okay if a normal transaction was predicted as a fraud, you can call the bank user and make sure everything is fine. However, if the model predicted a fraud transaction as a sound one — you and your client could lose a large amount of money.

So, it is essential to select the correct model estimator based on the data and the context of the problem.

**Accuracy**

The easiest way to determine a model performance is to count the number of correct answers out of a total number. In other words, we can check the **accuracy**. As we've mentioned above, this works well with a balanced dataset, where errors are more or less equally distributed per class.

Accuracy=Correct answersAll answersAccuracy = \frac{\text{Correct answers}}{\text{All answers}}Accuracy=All answersCorrect answers​



Remember, algorithms are lazy! Suppose you have 100 emails. 98 of them are legit, so there are just two spam emails. If your model predicted that, then the model's accuracy is 98%. Is that a good model though?

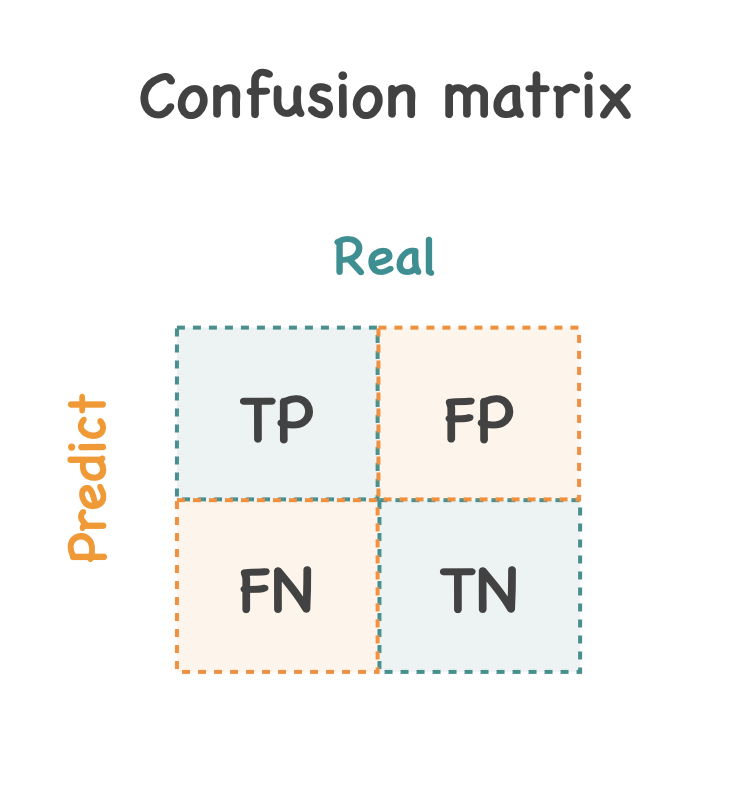
This example shows that accuracy is not a very relevant metric with unbalanced problems or when each class has a different error weight. But if your problem has a balanced sample, then it is obvious that your goal is to achieve accuracy equal to 1. And don't forget about overfitting.

**Confusion matrix**

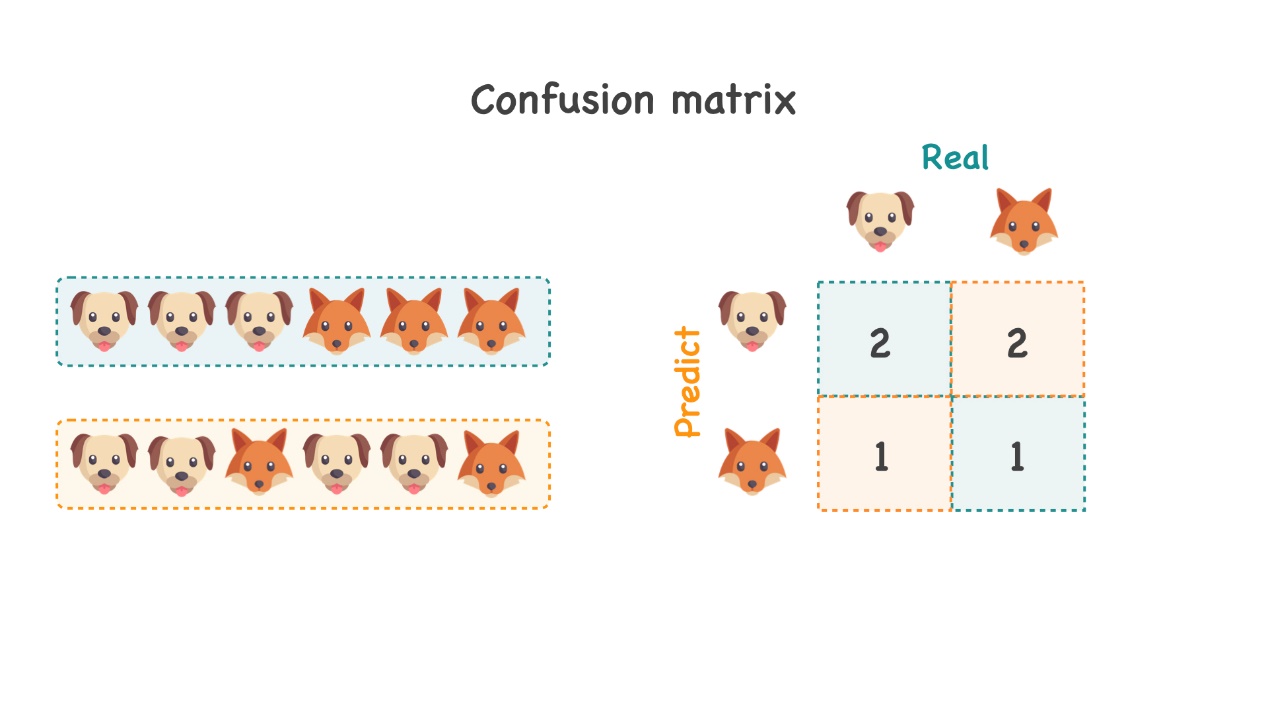
We suggest the following notation to define more relevant metrics. For binary classification problems, with classes marked as 0 and 1, we can define the following notions:

* True Positive (TP) — the real "ones" that were correctly predicted as "ones"
* True Negative (TN) — the real "zeros" that were correctly predicted as "zeros"
* False Positive (FP) — the real "zeros" that were predicted as "ones"
* False Negative (FN) — the real "ones" that were predicted as "zeros"

This can be represented as a **confusion matrix**:



As an example, consider the confusion matrix based on the example above:



Accuracy can now be calculated as follows:

Accuracy=TP+TNTP+TN+FP+FNAccuracy = \frac{TP+TN}{TP+TN+FP+FN}Accuracy=TP+TN+FP+FNTP+TN​

Obviously, for multi-classification, a confusion matrix is generalized to any number of classes. It is important to note here that when you get a confusion matrix, your goal is to get large values on the main diagonal, and small values for all other elements.

**Precision and recall**

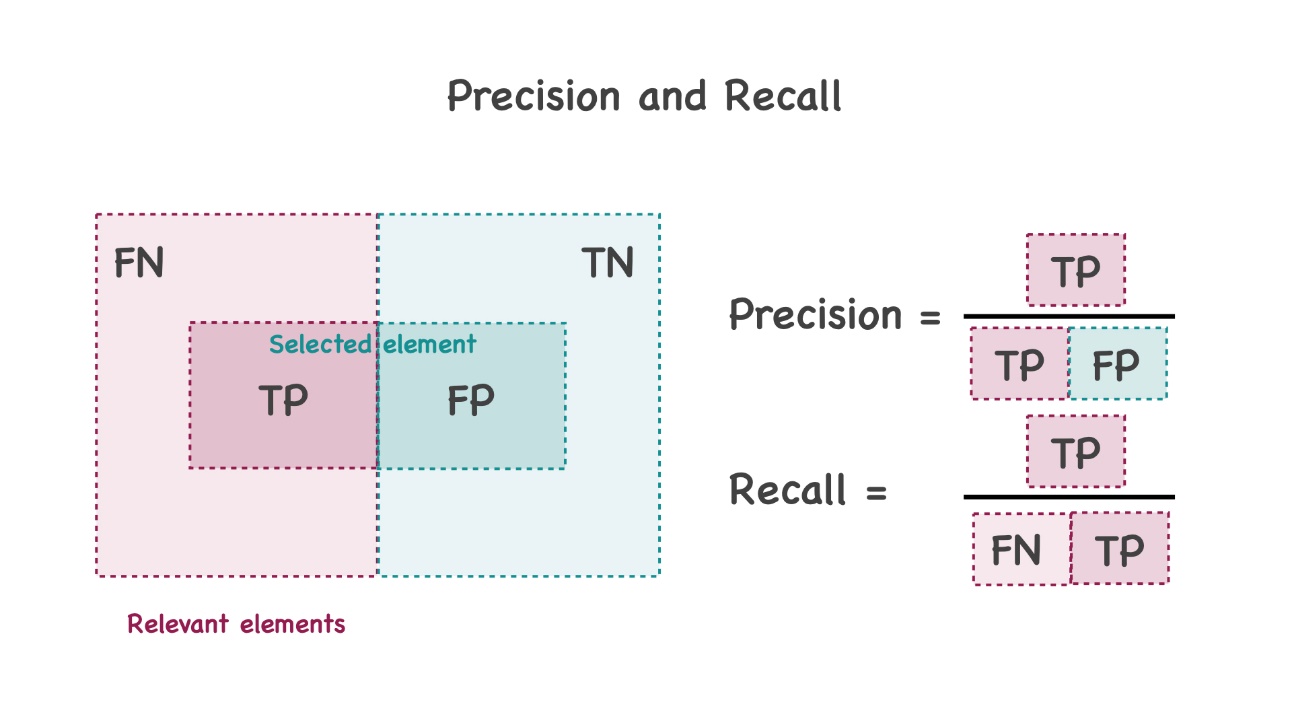
These metrics are usually considered together since they are two sides of the same thing. **Precision** is the proportion of the predicted "ones" that were the real "ones". **Recall** is the fraction of all "ones" which were detected by a model.

Precision=TPTP+FPPrecision=\frac{TP}{TP+FP}Precision=TP+FPTP​

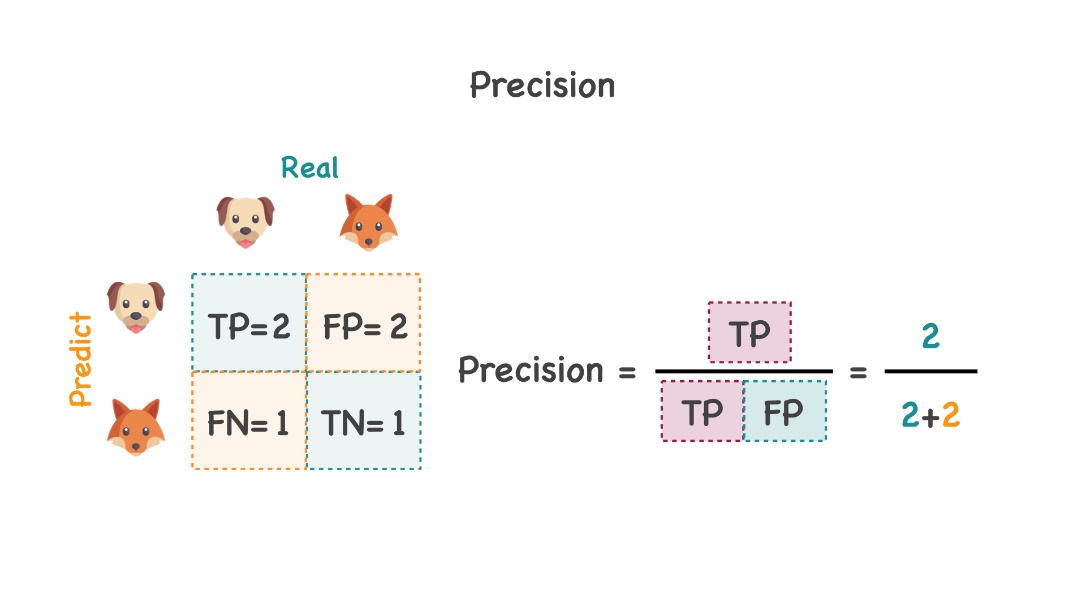
Recall=TPTP+FNRecall=\frac{TP}{TP+FN}Recall=TP+FNTP​

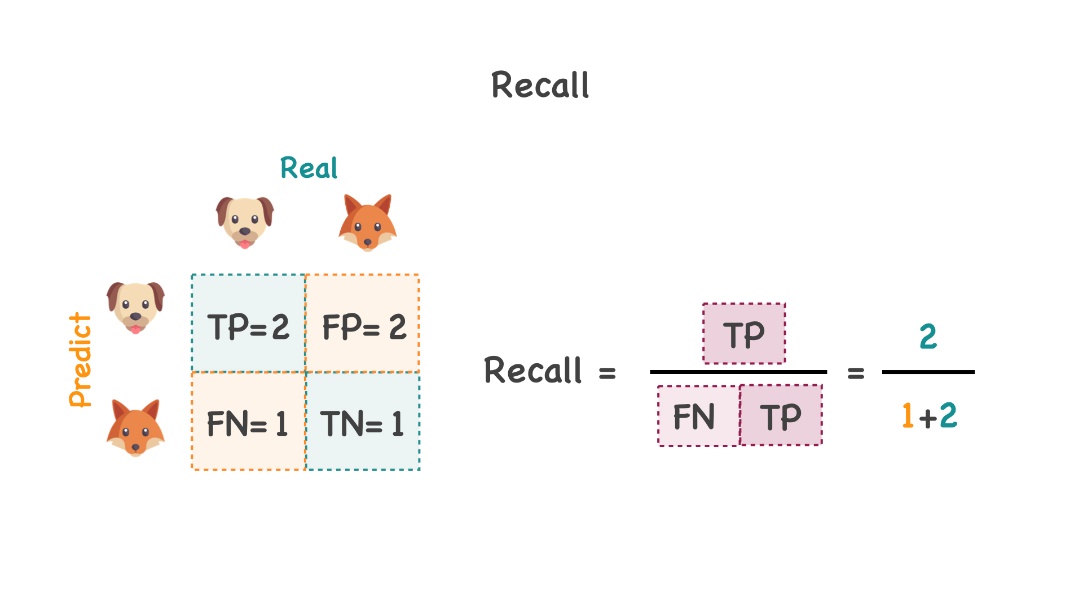
In other words, recall demonstrates the algorithm's ability to detect some class in general, and precision displays the ability to distinguish this class from other classes.

As an appropriate illustration of the above, take a look at the following picture:



Finally, fox-dog wise:



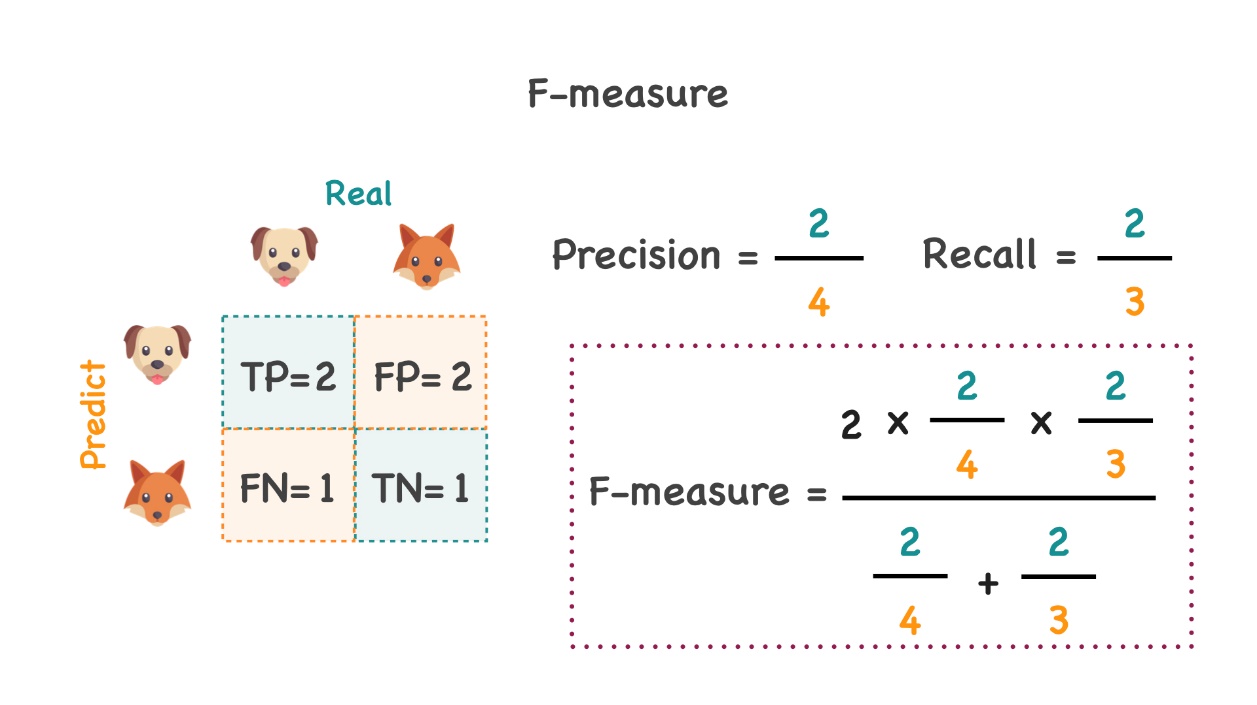


In contrast to accuracy, it is essential to note that precision and recall do not depend on the ratio of classes, so they apply to unbalanced samples. In a similar manner to accuracy, your goal is to get as close to 1 as possible.

**F-score**

Precision and recall are two convenient metrics, and there is a way to optimize them. To do this, we can combine them into one, and the result will be the so-called **F-score**or**F-measure**.

F=2 Precision × Recall  Precision + Recall F=\frac{2 \text { Precision } \times \text { Recall }}{\text { Precision }+\text { Recall }}F= Precision + Recall 2 Precision × Recall ​



The F-score reaches its maximum when the precision and recall results are equal to one. It is close to zero if at least one of the two is close to zero. In our example, the F-score has a value of 0.57, which is somewhere in between (and not very good).

If in your problem, one metric (either precision or recall) is more important than another, we can further complicate the formula. To establish the importance of a particular metric, we can introduce an additional beta parameter and recast the formula in the following form:

Fβ=(1+β2)Precision × Recallβ2Precision + Recall{F}\_{\beta}=\frac{\left(1+\beta^{2}\right) \text {Precision } \times \text { Recall}}{\beta^{2} \text {Precision } + \text { Recall}}Fβ​=β2Precision + Recall(1+β2)Precision × Recall​

Regardless of the selected beta value, a good model should calculate a result close to 1.

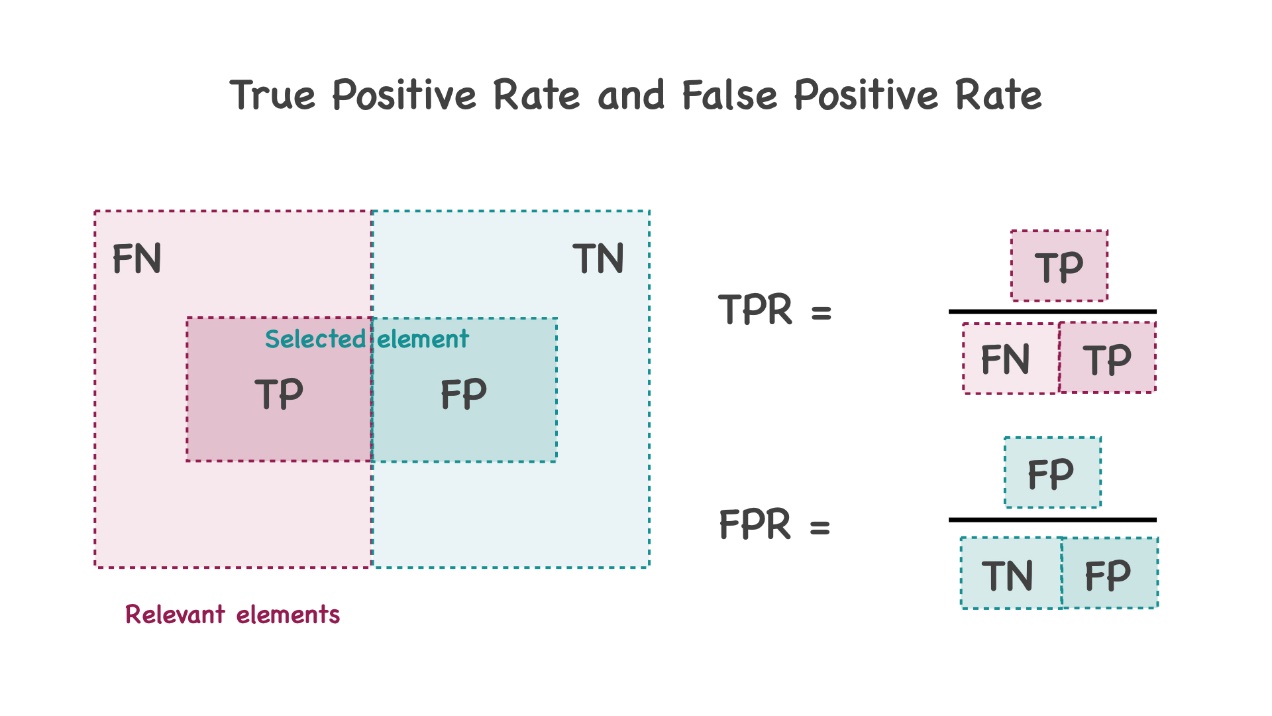
**AUC-ROC**

The previous metrics, rigorously speaking, referred to a situation where the model predicts a result in zeros or ones (integers, if there are multiple classes). But many models do not output the class itself, but the probabilities of classes — real numbers from zero to one. In this case, the data scientist sets a threshold up to which all results are zeroed, and after which they become ones. However, in real practice, the threshold is often not so obvious. There is a way to evaluate the model without determining a specific threshold — to draw the **Receiver Operating Characteristic Curve** (ROC curve) and estimate the **Area Under Curve** (AUC).

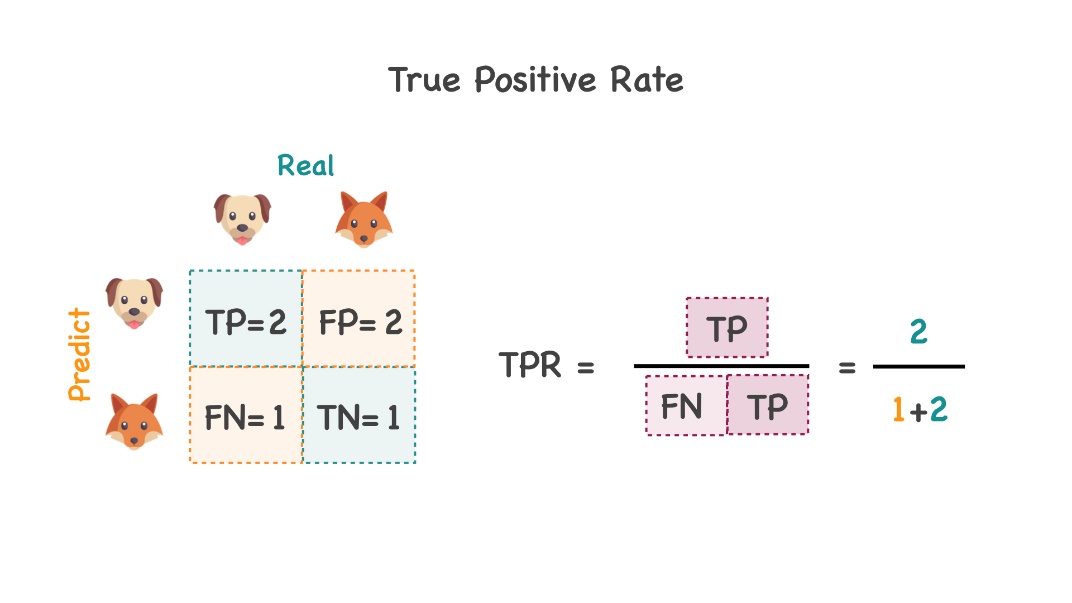
This curve is plotted in two coordinates — **True Positive Rate** (TPR) and **False Positive Rate** (FPR), which are defined in the above terminology as:

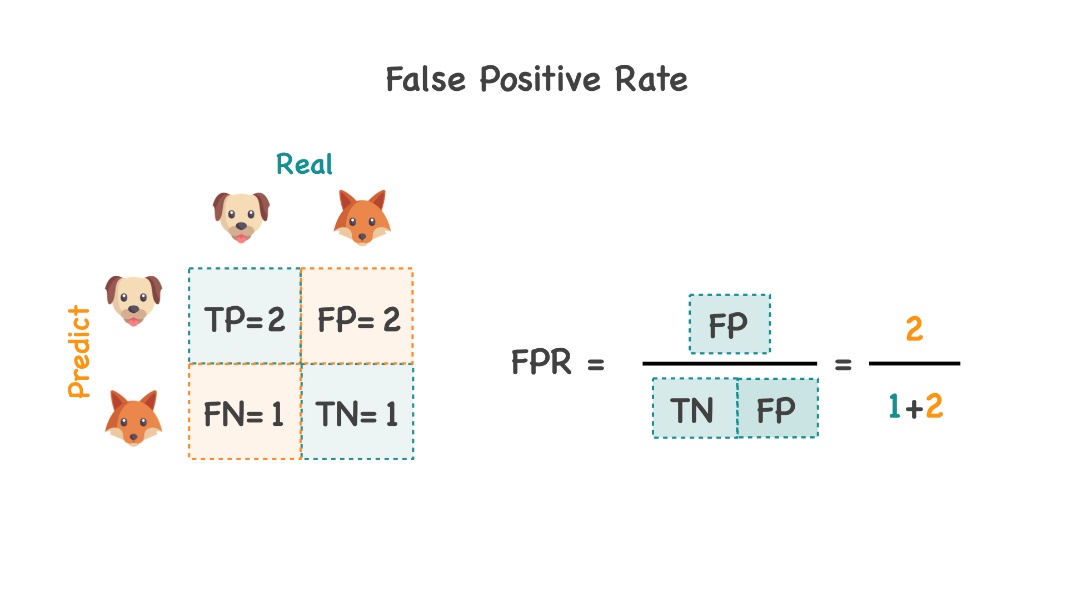
TPR=TPTP+FNT P R=\frac{T P}{T P+F N}TPR=TP+FNTP​

FPR=FPFP+TNF P R=\frac{F P}{F P+T N}FPR=FP+TNFP​

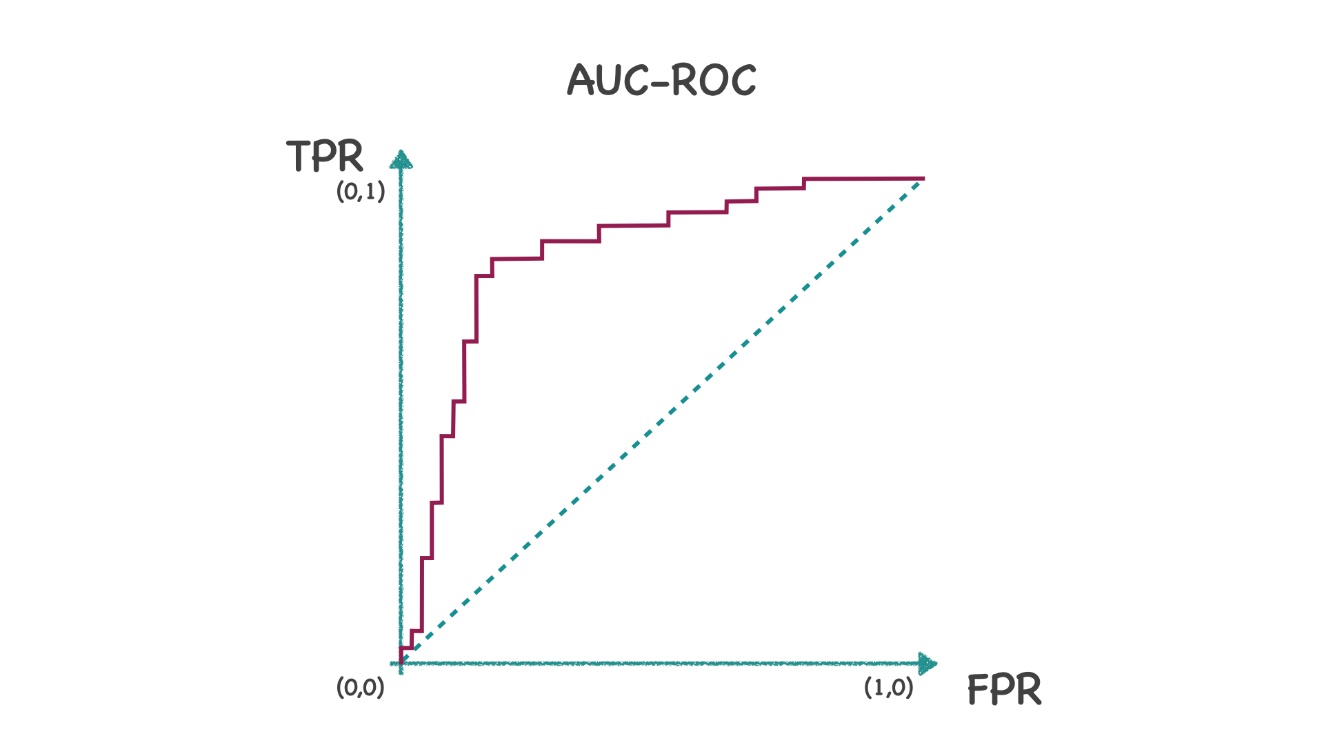


In our fox-dog terms we would have the following:



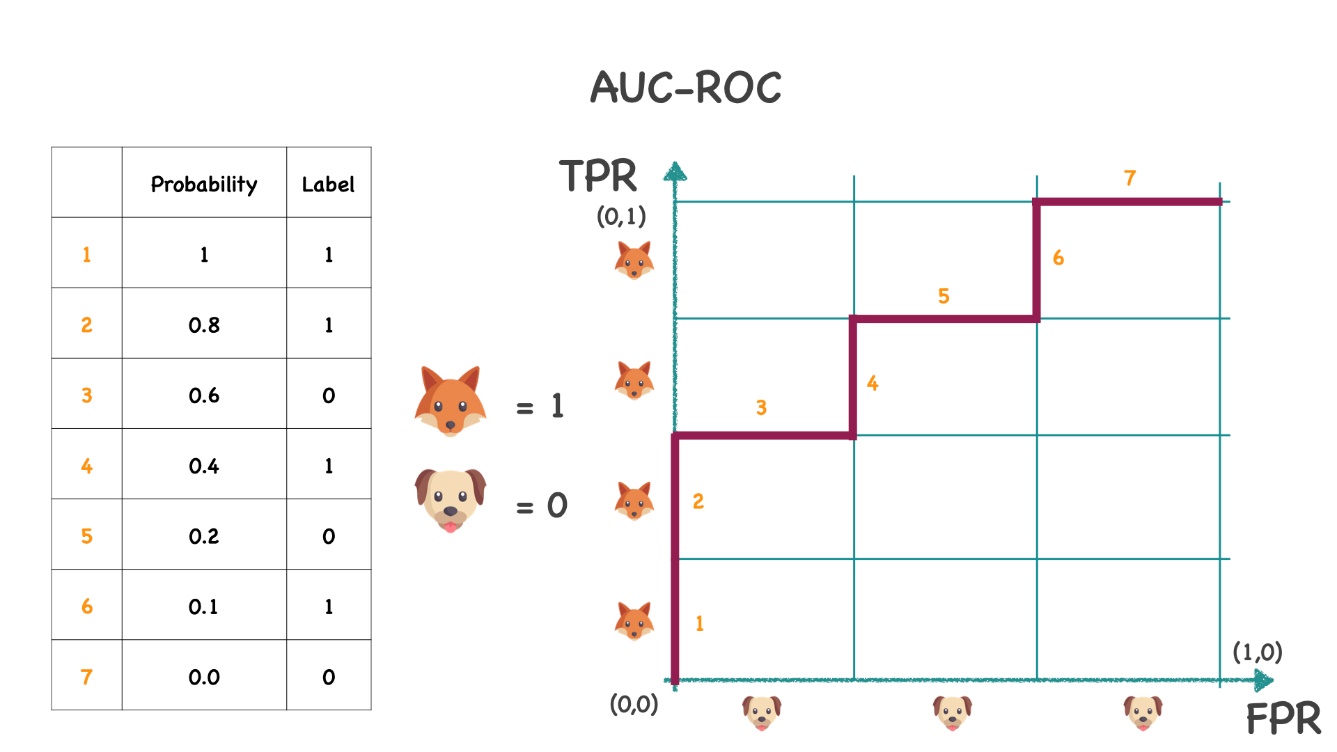


The True Positive Rate, also known as the **sensitivity**, shows which percentage of class "1" objects we classified correctly. The False Positive Rate, also known as the **specificity**, shows the proportion of class "0" that were mistakenly classified as "1". If you look closely, you will notice that Sensitivity (TPR) and recall are the same, but they are usually called different terms depending on the task.



Our ideal curve describes the case when the TPR is maximum, and the FPR is minimum, which means that the curve should tend to the point (0,1). Moreover, each point on the graph corresponds to the choice of a certain threshold.

Let's try to build a ROC-Curve for a simple model. Let's say we have some model that predicts not the labels, but the probability that the selected object is a fox. Then let's order the objects in descending order of this probability and give the objects numbers, as shown in the table:

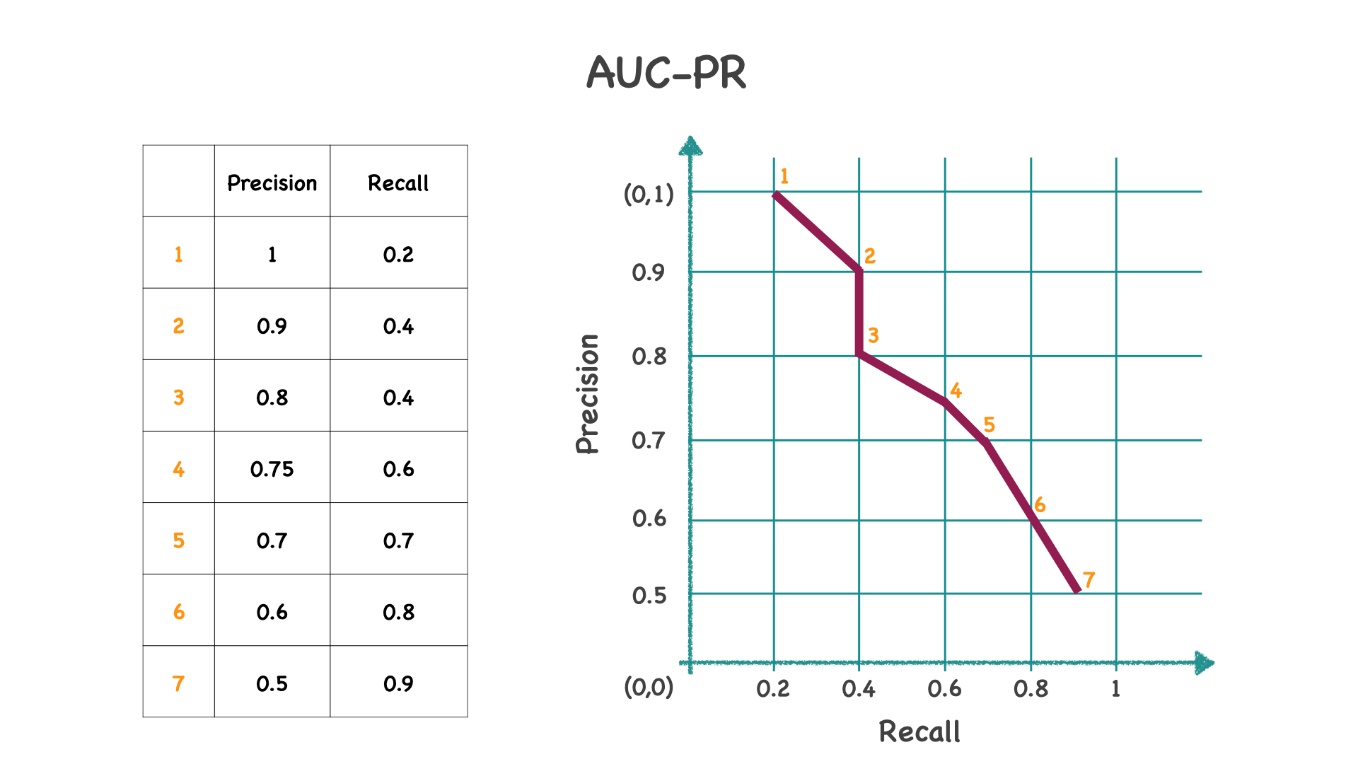


Divide the TPR axis by the number of ones (4 foxes), and the FPR axis by the number of zeros (3 dogs). Now we need to build a broken line from point (0, 0) to (1, 1). We would take a step up if the real label is "fox" (1) and step to the right if the real label is "dog" (0). This is how we get a ROC curve.

This metric is entirely different from the previous ones by the numbers you would expect from a good model. If the model predicts good results, then the ROC-AUC will be about 1 (a square with sides of 1), but a bad model will give results close to 0.5, which is equivalent to a random prediction.

**AUC-PR**

The ROC-AUC metric also has a significant drawback in its performance on unbalanced data. It can give an inadequate evaluation of the algorithm's performance in unbalanced sets, as is the case with accuracy. For these reasons, similarly to the logic above, we can come back to the precision and recall terms and build a curve with these coordinates.



Similar to the example above, we have a set of precision and recall metrics for each sample. We sort it by precision. Next, we start our curve from the upper left corner and set the Precision and Recall points according to the table. As you can see from the reasoning, this approach does not involve each class's quantity, so this approach is more relevant to unbalanced data. Likewise, try to avoid an area equal to 0.5.

**Logistic Loss**

This feature is also called **cross-entropy**. This is a prevalent and essential metric derived from the maximum likelihood method, which we will talk about in another topic.

logloss=−1l⋅∑i=1l(yi⋅log⁡(y^i)+(1−yi)⋅log⁡(1−y^i))logloss=-\frac{1}{l} \cdot \sum\_{i=1}^{l}\left(y\_{i} \cdot \log \left(\hat{y}\_{i}\right)+\left(1-y\_{i}\right) \cdot \log \left(1-\hat{y}\_{i}\right)\right)logloss=−l1​⋅∑i=1l​(yi​⋅log(y^​i​)+(1−yi​)⋅log(1−y^​i​))

In the formula above, we use the notation: yiy\_iyi​ — the real label, y^i\hat{y}\_iy^​i​ — the predicted label.

Let's look at the formula more closely. We can understand that we are looking for maximum accuracy, taking into account the penalty for incorrect predictions. This metric is interesting in the way that we do not want to have high results. A good result will be close to 0.

**Summary**

In this topic, we have covered several vital metrics for binary classification. Note that many of these can be generalized to multi-classification cases.

Let's once again recall all the main points, pros, and cons of each of the metrics:

* Accuracy is a simple metric, easy to interpret, but unfortunately does not work well with unbalanced samples.
* The confusion matrix is a convenient presentation of the results in TP, FP, TN, and FN terms, which allows us to look more broadly at the predictions, also see the mistakes for each class. Cons — it is not easy to compare two matrices to measure the quality of the models.
* Precision is a measure that shows how many of all predicted samples of a certain class were predicted correctly. It can be used to show how the algorithm works on a specific class but it is not exhaustive.
* Recall shows how many of the real values of a specific class the model could predict. It shows well the quality of the model with small classes, but it is not exhaustive.
* F-measure is a combination of Precision and Recall metrics since tasks often need to be optimized concurrently. Not very intuitive to interpret, difficult to set an additional β−\beta-β−parameter.
* ROC-AUCis the area under the ROC curve. The most popular metric for analyzing model performance as it predicts probabilities rather than classes. It helps to determine the relevant *threshold* for binary classification tasks but doesn't work well with unbalanced samples.
* AUC-PR is the area under the Precision-Recall curve. It is a quality metric that is not related to the number of classes and therefore works well on unbalanced classes.
* LogLoss is a quality metric, which is based on the addition of a penalty for the model's confidence if a prediction is wrong. Easy to optimize and positively to reveal.

Further, in practice, you will have the opportunity to solve problems and remember all the information of this topic.

Report a typo

4 users liked this theory. 4 didn't like it. **What about you?**