DIGITAL CULTURE
ITMO UNIVERSITY

Classifiers: k-NN and Naive Bayes

High School of Digital Culture
ITMO University
dc@itmo.ru

Contents

1 k-NN Classifier

1.1 Brief Introduction

Hello everyone! In the previous module, we introduced you to a regression problem which is one of the two branches of supervised learning. As you know, regression aims to predict a numerical value (or, more precisely, a continuous variable) based on known input data. This module will cover several approaches to classification problems that come from another branch of supervised learning.

We have already noted that classification problems are a part of our everyday life. Just think about it. Children playing with a pyramid toy are stacking the rings on the cone in different orders by colors and size. Adults sort laundry to organize clothes by color or fabric. In a supermarket, you will find vegetables in the produce aisle, milk in the dairy aisle, and cleaning supplies will be somewhere away from food products. Applicants are classified when they apply for a loan (creditworthy or not) or a job (competent or not). Patients are classified into risk categories. The list goes on and on. These examples are just a tiny bit of what to classify in everyday life.

Anyway, classification problems prevail in business, and there's no doubt that they influence our lives. While checking email, one of the most annoying things is to delete an important email along with spam. On the other hand, the junk folder is never empty. How do emails get there? Why do some important emails go to spam? It all depends on the algorithms and settings of a so-called spam filter, which examines emails, detects unwanted emails, and stops the latter from getting into inboxes. It's a binary classification example. However, spam is usually harmless. But what about suspicious transactions?

For example, you lost a credit card but didn't notice it was missing. The person who found the card started to withdraw large amounts of money. We bet that within a minute or two you'd receive a call from your bank to confirm the transactions. Then, the bank would block the lost card. That's why clients are usually advised to notify banks of travel plans, big purchases, and things like that. Otherwise, activities may be considered suspicious, and the bank can block the card. It's such a shame for a traveler.

We've justified the solution to the classification problem, and now we can move on to one of the simplest and straightforward methods of metric classification, which is called the k-nearest neighbors method (or k-NN).

1.2 The Intuitive Conception of the k-NN Algorithm

Before we formally define an algorithm, let's talk about the idea and the essence of the method. First of all, we are going to describe the problem formally and see what we have and what we want.

In supervised learning, a training dataset is a standard one. Each object of the dataset has p predictors $X_1, X_2, ..., X_p$ and a response Y. In a classification problem, a set of responses Y consists of the so-called class labels (a known finite set of elements). For convenience, classes are usually numbered.

Remark 1.2.1 *When solving a binary classification problem, a set Y is often assumed to include two elements $+1$ and -1 (positive and negative classes), that is $Y = \{-1, 1\}$.*

When solving a multiclass classification problem, such as M-class classification (given $M > 2$), a set Y is usually assumed to be as $Y = \{1, 2, ..., M\}$, where each element of Y mutually unambiguously (bijectively) corresponds to a class of the domain.

Example 1.2.1 *For example, when solving a 4-class classification problem with possible class labels banana, apple, pear, orange, the mutually unambiguous correspondence can be as follows:*

$$1 \leftrightarrow \textit{banana},\ 2 \leftrightarrow \textit{apple},\ 3 \leftrightarrow \textit{pear},\ 4 \leftrightarrow \textit{orange}.$$

Numbers can differ. Sometimes they are even not needed. It depends on an algorithm.

Well, the input data is clear. Now let's discuss the logic behind the model that can make predictions based on the available training data. To develop an algorithm, let's look at Figure 1 first.

Assume that the considered objects are described by two numerical predictors X_1, X_2 and one of the two possible responses, for example, Os or Xs (binary classification). Since the training data are objects that belong to known classes, they will be denoted by Os or Xs for clarity (depending on the response).

Our aim is not to visualize training objects for a nice picture but to assign one of the two possible classes to a new test observation with two predictors. Look at the red object. What class to assign to it: Os or Xs? We assume that the new object is an X. Why do we think so? Well... We see that the new object is surrounded by Xs. Therefore, it's reasonable to assume that the new object is also an X. This thought reminds us of the saying: Show me who your friends are, and I'll tell you who you are. However, certainty is a rare thing.

Let's consider a less obvious variation (Fig. 2 a)). What class to assign to the red test object? Think of your choice and its justification. There is no correct answer to this question (and to the previous one). However, we can use some logic. As you probably guessed, the easiest and natural way is to find the **nearest** training object and check its class label. Based on what we see and the

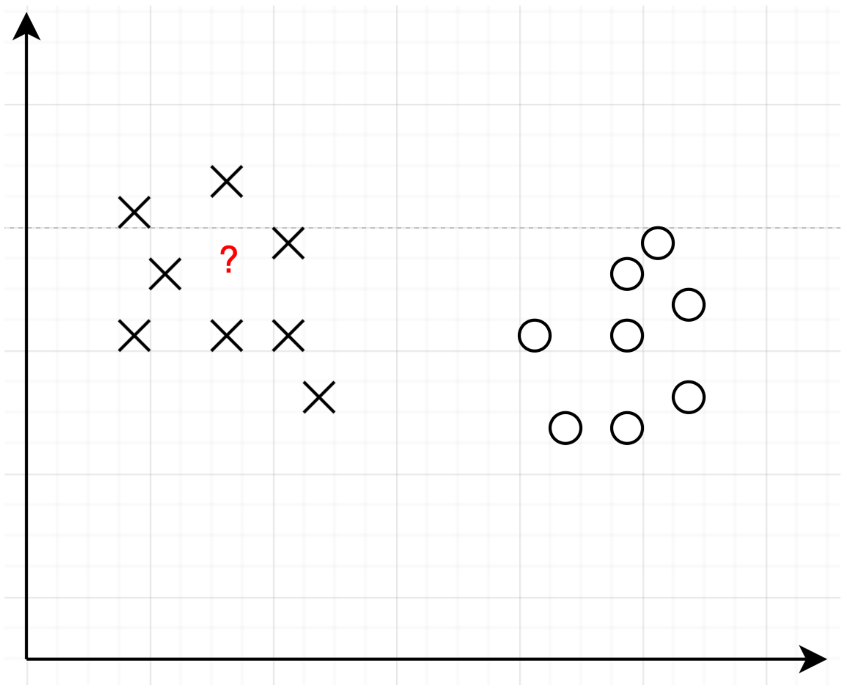

Figure 1: The example of the intuitive approach.

results obtained from the calculations, the nearest object belongs to Os, which means we can assign Os to the test observation. That's all? Or not?

A very reasonable question arises after a moment's thought. Why do we base our decision on a class label of one training object? What if it is an outlier? In such a case, we will be wrong. Besides, we cannot see the big picture because of this faulty logic. We keep looking at things nearby out of perspective. This approach doesn't always work, and we are not satisfied with that. So, what can we do? The solution is easy to find. We will make the algorithm see more training objects nearby. In our example, let's consider the response of three nearest neighbors instead of one. What happens in this case (Fig. 2 b))?

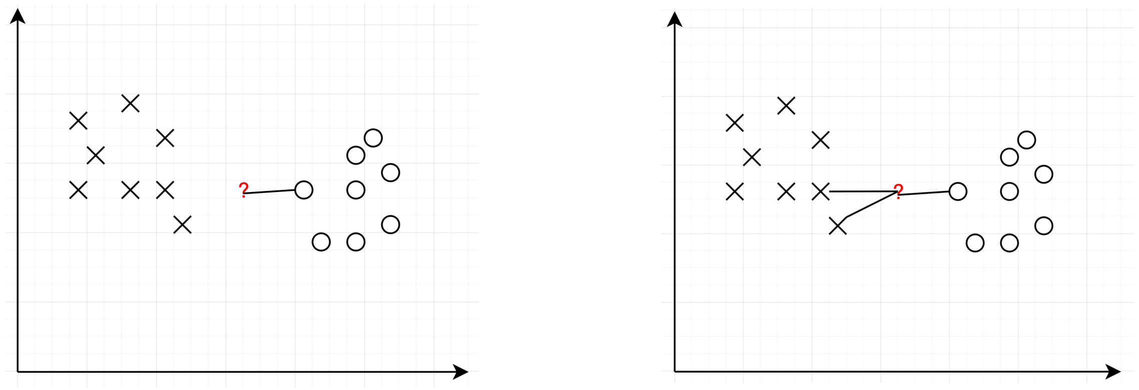

Figure 2: Choosing the number of neighbors.

This time, three nearest neighbors are two Xs and one O. What class to assign to the test observation? Perhaps, it's a majority decision. Therefore, an assigned class is Xs (based on the three nearest neighbors).

The k-nearest neighbors algorithm (or k-NN) works the same. Even this simple example shows that the right choice of k is one of the main tasks in the algorithm. This task will come up again and again, and later we will explain it in much more detail. However, for the moment, let's look at one more thing. It turns out that the choice of k is not the only problem.

In our example, we often used the word 'nearest', and our classification was based on the concept of distance. We also measured a straight-line distance. But is it always the case? What are other ways to measure distances? Why is that so important? Let's find out.

1.3 Metrics

When it comes to measuring the distance between two objects, the most obvious thing to do is to take out a ruler, measure the straight-line distance between the points (or the shortest route), and get the desired number. Meters or feet — it doesn't matter. However, this approach does not apply to every problem. Imagine a globe of the world and a penguin at the South Pole. The globe is a ball of, say, 3-meter radius. The penguin wants to get to the North Pole by the shortest route. What distance will the penguin cover? We cannot connect the two points and obtain 6 meters because the penguin cannot go through the globe. But what can we do then? Some of you might have guessed that the penguin should go half the length of the circumference of the central part of the ball. Since the central part is a circle of radius 3, then half the length of the circumference can be written as $3\pi \approx 9.4$ meters. And it is greater than 6.

There's another example. Imagine that you're visiting Saint Petersburg (in Russia) and plan your travel itinerary. The Winter Palace is just a stone's throw away from the Peter and Paul Fortress, but, unfortunately, there is no river crossing. So, you have to make a wide circle around it. You move across the Palace Bridge, go around the Old Stock Exchange, cross the Stock Exchange Bridge, and then follow the banks of the Neva river. Thus, the distance passed turned out to be greater than the straight-line distance.

What are we trying to say? The choice of measurement method is very important. It often crucial for solving a metric problem.

Definition 1.3.1 *Let X be a set. A metric (distance) defined on X is a function $d(x,y) : X \times X \to \mathbb{R}$ that, for any $x, y, z \in X$, satisfies three conditions:*

1. *It is non-negative, $d(x,y) \geq 0$, and*

$$d(x,y) = 0 \Leftrightarrow x = y.$$

2. *It is symmetrical:*
$$d(x, y) = d(y, x).$$
3. *The triangle inequality is satisfied:*
$$d(x, y) \leq d(x, z) + d(y, z).$$

Now let's talk about the statements in the definition. The first condition is logical. The distance must be non-negative. Moreover, the distance between the objects is zero if and only if one object is contained within another object. The second condition says that the distance from point x to y is the same as from y to x. The third condition is the triangle inequality, which states that any side of a triangle must not be greater than the other two sides added together if the vertices of the triangle are x, y, z.

Let's consider the metrics that we will use. Assume that we are in a space $\mathbb{R}^p$, $x, x' \in \mathbb{R}^p$ and

$$x = (x_1, x_2, ..., x_p), \quad x' = (x'_1, x'_2, ..., x'_p).$$

1. Let's begin with the generally known Euclidean metric (the Euclidean distance) which is a generalization of the Pythagorean theorem. According to the definition, the distance $d_E(x, x')$ between points x and x' is a square root of the sum of the squared differences between the corresponding coordinates:
$$d_E(x, x') = \sqrt{(x_1 - x'_1)^2 + (x_2 - x'_2)^2 + \ldots + \left(x_p - x'_p\right)^2},$$
$$d_E(x, x') = \sqrt{\sum_{i=1}^{p} (x_i - x'_i)^2}.$$
This distance is probably the most common one, which is a straight-line distance. It doesn't matter where we are: on a straight line (in $\mathbb{R}^1$), in a plane (in $\mathbb{R}^2$), or in a space (in $\mathbb{R}^3$).

2. The Euclidean distance is a special case of the Minkowski distance:
$$d_q(x, x') = \sqrt[q]{|x_1 - x'_1|^q + |x_2 - x'_2|^q + \ldots + \left|x_p - x'_p\right|^q}, \quad q \geq 1,$$
$$d_q(x, x') = \sqrt[q]{\sum_{i=1}^{p} |x_i - x'_i|^q}, \quad p \geq 1.$$
The Euclidean distance is derived from the Minkowski distance when $q = 2$.

3. When $q = 1$, we get what is called the Manhattan distance that was initially used to calculate city block distance in Manhattan.

$$d_1(x, x') = \sum_{i=1}^{p} |x_i - x'_i|.$$

This measure is also known as taxicab distance in recognition that many streets in a city are perpendicular or parallel to each other. In this case, a random but common-sense route connecting two points will be similar to the one we imagine (Fig. 3). The figure shows red, blue, and yellow reasonable routes connecting two points. As has been said, all these routes have the same distance according to the Manhattan distance. The green route is the shortest path, which is a straight line. However, this route doesn't exist.

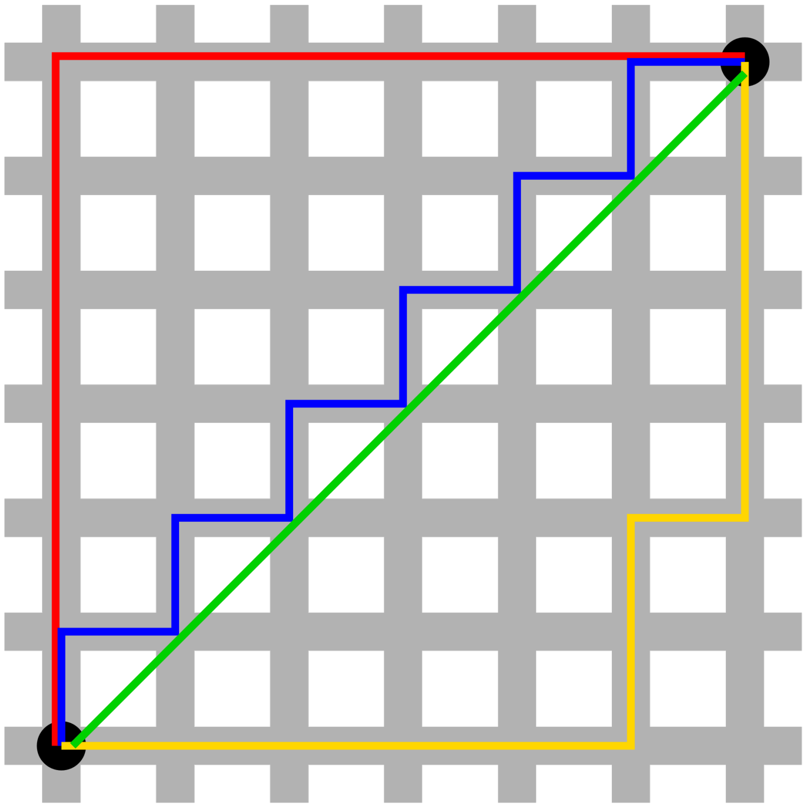

Figure 3: The Euclidean distance and Manhattan distance.

4. When q approaches $+\infty$, we get the Chebyshev distance:

$$d_\infty(x, x') = \max_{i \in \{1,\dots,p\}} |x_i - x'_i|.$$

The Chebyshev distance shows the maximum coordinate difference between the objects, and it is useful in the following case, for example. Suppose that you are measuring levels in the reservoirs $1, 2, ..., n$, and the normal water level of each reservoir equals $x'_1, x'_2, ..., x'_p$ respectively. The Chebyshev distance shows the maximum difference between water levels in the reservoirs and the norm.

We can prove that all the functions that we've discussed are metrics and that they satisfy the definition introduced earlier. Because we will use metrics to compare object proximity, it is important to understand their geometric interpretation. We will also find out which metric reflects your way of thinking :)

Assume there's a set of points in the plane. The points are less than 1 away from the origin $O(0, 0)$. How do you imagine it? Most likely, you are thinking of a unit circle with its center at the origin. If you sense there's a catch, you are right. Initially, we used the word distance, and the geometric pattern strongly depended on the chosen distance. To make it clear, let's translate the previously asked question into the language of mathematics: illustrate the set of points $x \in \mathbb{R}^2$ so that

$$d_q(O, x) \leq 1, \quad O(0, 0),$$

given different p.

In the case of the Euclidean distance and given $p = 2$, the inequality is written as follows:

$$d_2(O, x) = \sqrt{x_1^2 + x_2^2} \leq 1.$$

We square it to obtain an equivalent inequality:

$$x_1^2 + x_2^2 \leq 1,$$

which defines a unit circle with its center at the origin. Most people think in terms of the Euclidean metric.

In the case of the Manhattan distance and given $q = 1$, we obtain the inner part of the diamond (in fact, it is a square rotated through $\pi/4$):

$$|x_1| + |x_2| \leq 1.$$

In the case of the Chebyshev distance and given $q = +\infty$, we get the area inside the square with the side of two:

$$\max_{i \in \{1,2\}} |x_i| \leq 1 \Leftrightarrow |x_1| \leq 1, |x_2| \leq 1.$$

All sets are shown in Fig. 4.

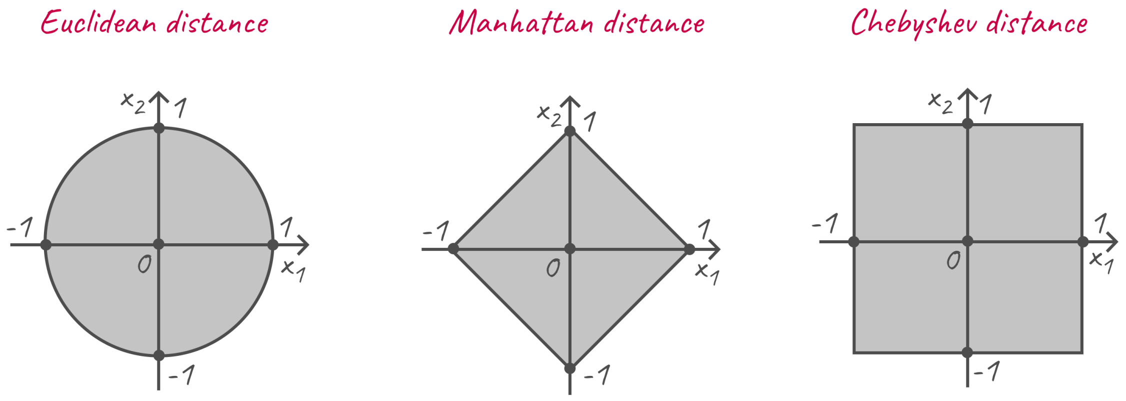

Figure 4: Distances from the left to the right: Euclidean, Manhattan, Chebyshev.

1.4 k-NN Classification

Now that you know the concept of a metric, we can move on to developing a classification algorithm based on k-NN. Assume that $X = (x_1, \dots, x_n)$ is a training dataset of size n,

$$x_i = (x_{i1}, x_{i2}, ..., x_{ip}), \quad i \in \{1, 2, ..., n\},$$

and each object x_i corresponds to the response $y_i \in Y$, and the metric d is defined on a p-dimensional set of objects. Thus, a classification algorithm is as follows:

1. For a new object z, calculate the distance $d(z, x_i)$ to each object x_i, $i \in \{1, 2, \dots, n\}$.

2. Arrange the elements of the training dataset in order of non-decreasing distances to z:

$$d(z, x_1^{(z)}) \leq d(z, x_2^{(z)}) \leq ... \leq d(z, x_n^{(z)}),$$

where $x_1^{(z)} = x_{t_1}$, and t_1 is a solution to the problem $\underset{i \in \{1,2,\dots,n\}}{\operatorname{Arg\,min}} d(z, x_i)$, $x_2^{(z)} = x_{t_2}$, and t_2 is a solution to the problem $\underset{i \in \{1,2,\dots,n\} \setminus \{t_1\}}{\operatorname{Arg\,min}} d(z, x_i)$, etc.

3. Renumber the responses according to the instructions in Step 2:

$$y_i^{(z)} = y_{t_i}, \quad i \in \{1, 2, ..., n\}.$$

4. Among k-nearest neighbors, find the most frequent class $y \in Y$ (choose any if there are more than two):

$$a\,(z, k) = \underset{y \in Y}{\operatorname{Arg\,max}} \sum_{i=1}^{k} \mathsf{I}\left(y_i^{(z)} = y\right),$$

where the function $\mathsf{I}(A)$ being the indicator of the event A is equal to one when the event A has occurred (true) and zero otherwise. In this case, the event indicator is one when the class label $y_i^{(z)}$ equals the class label y.

The first step is to calculate the distance from the test object to each element of a training dataset. The second step is to arrange these distances in non-descending order and renumber the elements of the training dataset (the first element is the closest element to the test object, the second is the second closest, etc.). The third step is to renumber the responses according to the renumbered objects, and the fourth step is to find a class (or classes) which objects are the most frequent among k nearest elements. The described algorithm replicates all the heuristic arguments made in the very beginning.

It's logical that the metric d chosen in the algorithm well reflects the similarity between objects. To put it differently, the larger the $d(x, x')$, the less similar the objects x and x' or vice versa. The described classification algorithm assumes that similar objects are more often found in the same class rather than different ones. Formally, we use the **compactness hypothesis**. It states that possible classes form compact subsets in the object space.

Definition 1.4.1 *Algorithms based on the analysis of the similarity between objects, which is found using the metric d, are often called metric algorithms.*

It follows from the definition that the described algorithm k-NN is a metric algorithm. Though some algorithms are also called metric algorithms when the similarity function d is not a metric according to the introduced definition (the triangle inequality is often not satisfied in such cases).

Note that the k-NN algorithm is also called **lazy** since training happens only at the time of prediction. In other words, training is keeping the training dataset.

Remark 1.4.1 *In practice, features may be expressed with different units of measurement, which can lead to misinterpretation of the real distance between the objects. A solution is data normalization. Recall that one of the simplest ways is to use relative values obtained by linear normalization:*

$$X_i' = \frac{X_i - X_{\min}}{X_{\max} - X_{\min}},$$

where $X_{\min}$ is the minimum value of the considered predictor, $X_{\max}$ is the maximum value, X_i is the normalizable value, and X_i' is the normalized value.

1.5 Classification Example

Let's consider an example of the algorithm application. The data is given in the table. Each object belongs to one of the three classes (fruit, vegetable, or

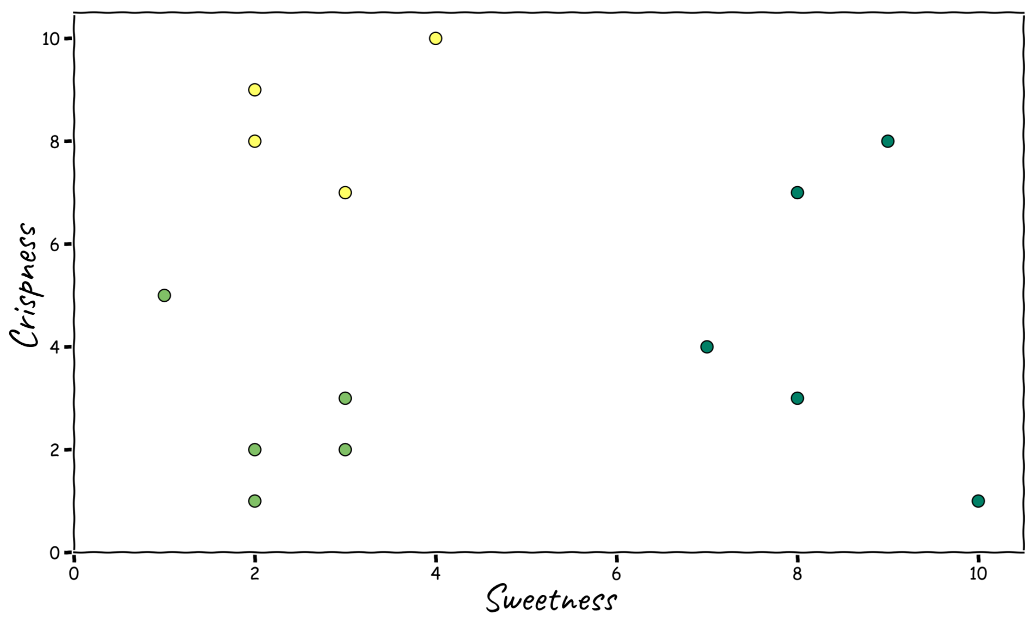

Figure 5: Visualization of the training data.

protein), and it is characterized by two attributes (predictors), which are sweetness and crispness.

Product	Sweetness	Crispness	Class
banana	10	1	fruit
orange	7	4	fruit
grapes	8	3	fruit
shrimp	2	2	protein
bacon	1	5	protein
nuts	3	3	protein
cheese	2	1	protein
fish	3	2	protein
cucumber	2	8	vegetable
apple	9	8	fruit
carrot	4	10	vegetable
celery	2	9	vegetable
iceberg lettuce	3	7	vegetable
pear	8	7	fruit

First of all, let's plot the data in the plane for clarity (Fig. 5). The sweetness values are plotted on the horizontal axis and crispness on the vertical. Thus, each vegetable, a piece of fruit, or a source of protein corresponds to a colored point in the plane: yellow points are vegetables, dark green points are fruit, light green points are proteins. The data is well separated into groups (Fig. 6).

Now suppose that we need to classify a new item based on the values of sweetness and crispness only. Assume that this new item is a bell pepper with the

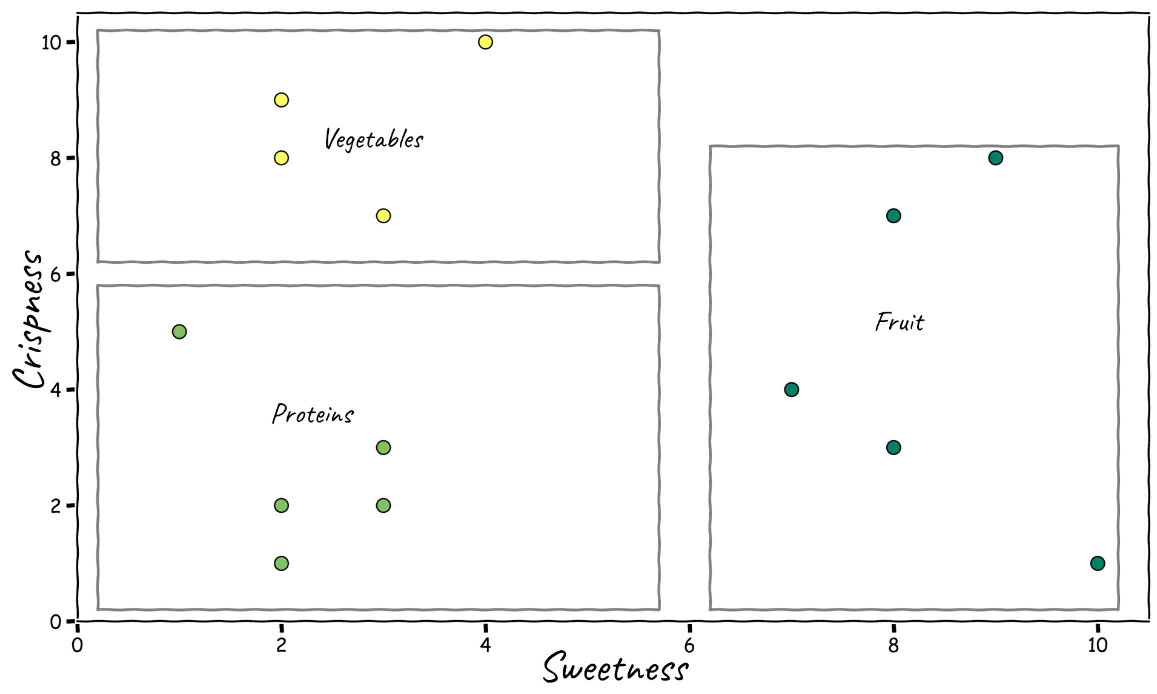

Figure 6: Grouping.

sweetness of 6 and crispness of 9. According to the described algorithm, we need to find the distance from the test object to all training objects using the metric d chosen earlier. First, let's choose the Euclidean metric d_2.

We assume that all variables in our example are normalized by a scale from 0 to 10. We find the distance between the carrot with attributes $(4, 10)$ and the bell pepper with attributes $(6, 9)$.

$$d_2(\text{carrot, bell pepper}) = \sqrt{(6-4)^2 + (9-10)^2} \approx 2.24.$$

We calculate the distances to other training objects in the same way. The results are shown in the table:

Product	Distance
banana	8.94
orange	5.10
grapes	6.32
shrimp	8.06
bacon	6.4
nuts	6.71
cheese	8.94
fish	7.62
cucumber	4.12
apple	3.16
carrot	2.24
celery	4
iceberg lettuce	3.61
pear	2.83

Let's arrange the data in the order of non-descending distances. Please see the table:

Product	Distance
carrot	2.24
pear	2.83
apple	3.16
iceberg lettuce	3.61
celery	4
cucumber	4.12
orange	5.10
grapes	6.32
bacon	6.4
nuts	6.71
fish	7.62
shrimp	8.06
banana	8.94
cheese	8.94

So, the carrot is the closest object to the test one, the second closest is the pear, apple, etc. Now let's choose the value of k. Let $k = 1$, thus, according to the algorithm, the test object is assigned the label of the class of the nearest training object. Since the closest object is the carrot that belongs to the class of vegetables, the bell pepper is also classified as a vegetable. Given $k = 3$, the apple, pear, and carrot are the closest items, and the bell pepper is assigned to the class of fruit because it is surrounded by fruits. Given $k = 4$, the lettuce becomes the nearest neighbor, which means the bell pepper can be assigned to fruit and vegetables. Given $k = 5, 6, 7$, the bell pepper is confidently classified as a vegetable.

Product	Class	Distance
carrot	vegetable	2.24
pear	fruit	2.83
apple	fruit	3.16
iceberg lettuce	vegetable	3.61
celery	vegetable	4
cucumber	vegetable	4.12
orange	fruit	5.10
grapes	fruit	6.32
bacon	protein	6.4
nuts	protein	6.71
fish	protein	7.62
shrimp	protein	8.06
banana	fruit	8.94
cheese	protein	8.94

Now let's see how the classification changes when we choose another metric. We are going to use the Manhattan distance d_1:

$$d_1(x, x') = \sum_{i=1}^{p} |x_i - x'_i| \, .$$

The task is to classify the bell pepper with the predictor values of sweetness equal to 6 and crispness equal to 9. We find the distance between objects carrot and bell pepper:

$$d_1(\text{carrot, bell pepper}) = |6 - 4| + |9 - 10| = 3.$$

The differences in distance calculations are shown in Fig. 7. So, since the Euclidean distance d_2 is the shortest path on a straight line (the hypotenuse), then the Manhattan distance d_1 is the sum of the lengths of the respective legs. We calculate the distance to the other points in the same way and immediately arrange the training objects in the order of non-descending distances to the test object. The results are given in the table:

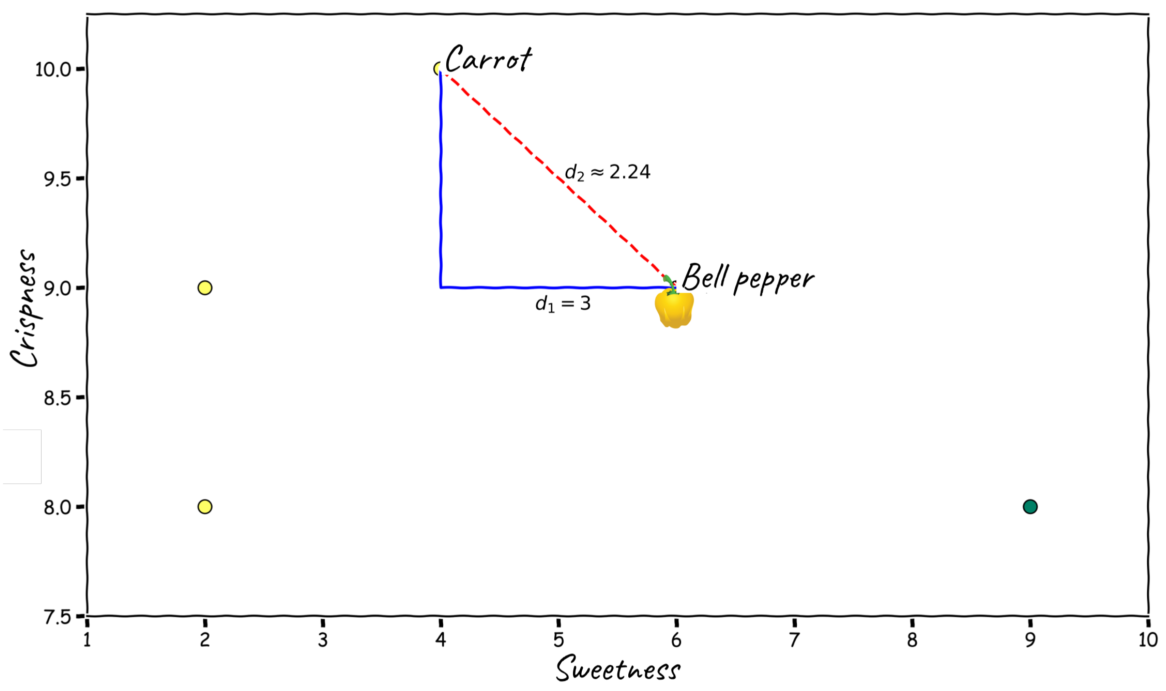

Figure 7: The differences between the distances.

Product	Class	Distance
carrot	vegetable	3
apple	fruit	4
celery	vegetable	4
pear	fruit	4
cucumber	vegetable	5
iceberg lettuce	vegetable	5
orange	fruit	6
grapes	fruit	8
bacon	protein	9
nuts	protein	9
fish	protein	10
shrimp	protein	11
banana	fruit	12
cheese	protein	12

Now we cannot confidently classify the bell pepper given $k = 2, 3, 4$, but we can assign it to the vegetables given $k = 1, 5, 6, 7$. Note that in the example, the Manhattan distance gives us integer values. Additionally, many distances are equal, and the order of equidistant objects that belong to different classes significantly affects the result.

1.6 Weighted k-NN

Well, we introduced you to the simplest form of the k-NN algorithm. As you know, when we were considering the last example, we encountered the problem. Given certain k values, a new test object was not classifiable because it could be assigned to multiple classes. The algorithm states that a new class can be any of the suitable ones, which sometimes seems irrational. Assume that you're solving a binary classification problem. The inability to classify a new object means that the algorithm responds something like: "I don't know! You need to decide on your own". The bad news is that such things happen, but the good news is that there are workarounds.

In the case of the binary classification, the easiest solution is to use odd numbers as k values. Because of this, it's not possible to divide the number of nearest neighbors by 2. Thus, no classes will have the same number of votes. Thus, the classification will be performed. What if there are more than two classes? This module will cover one approach to solving such a problem. This approach is called weighted k-NN. The idea is that each training object x_i is applied a weight ω_i that depends on a test object z:

$$\omega_i = \omega_i(x_i, z), \quad i \in \{1, 2, ..., n\}.$$

Consequently, the test observation is assigned to the class, which total weight is greater than that of other k nearest neighbors (or one of such classes if there are several of them).

We can formally describe the weighted k-NN algorithm as follows. Assume that $X = (x_1, \dots, x_n)$ is a training dataset of size n,

$$x_i = (x_{i1}, x_{i2}, ..., x_{ip}), \quad i \in \{1, 2, ..., n\},$$

and each object x_i corresponds to the response $y_i \in Y$, and the metric d is defined on a p-dimensional set of objects.

1. For a new object z, calculate the distances $d(z, x_i)$ to each object x_i and weights
$$\omega_i = \omega_i(x_i, z), \quad i \in \{1, 2, \dots, n\}$$
2. Arrange the elements of the training dataset in order of non-decreasing distances to z:
$$d(z, x_1^{(z)}) \leq d(z, x_2^{(z)}) \leq ... \leq d(z, x_n^{(z)}),$$
where $x_1^{(z)} = x_{t_1}$, and t_1 is a solution to the problem $\operatorname*{Arg\,min}_{i \in \{1,2,...,n\}} d(z, x_i)$, $x_2^{(z)} = x_{t_2}$, and t_2 is a solution to the problem $\operatorname*{Arg\,min}_{i \in \{1,2,...,n\} \setminus \{t_1\}} d(z, x_i)$, etc.

3. Renumber the responses and weights according to the instructions in Step 2:
$$y_i^{(z)} = y_{t_i}, \quad \omega_i^{(z)} = \omega_{t_i}, \quad i \in \{1, 2, ..., n\}.$$

4. Among k-nearest neighbors, find the class $y \in Y$ with the greatest weight (choose any if there are more than two):
$$a\left(z, k\right) = \operatorname*{Arg\,max}_{y \in Y} \sum_{i=1}^{k} \mathsf{I}\left(y_i^{(z)} = y\right) \omega_i^{(z)}.$$

One of the ways to assign weights is to consider the proximity of the object of interest to the other objects. The idea is simple. The less the distance from the training object to the test object, the more significant the training object for classification, or to be more precise, the more significant the vote of this training object or its class label. For example, we can take the value proportional to the inverse of the squared distance between the objects x_i and z as the weight w_i:
$$\omega_i = \omega_i(x_i, z) = \frac{1}{d^2(x_i, z)}.$$

Hence, the weights increase the significance of the nearest objects and decrease the significance of more distant objects. Later, we will describe a numerical example in detail but first, let's talk about the problem of choosing the value of k.

1.7 The Value of k in k-NN

To begin with, we will use concrete data to see how the change in k affects the k-NN classification in general. Look at Figure 8.

It's clear that when the values of k are small, our classifier is nearsighted. For example, when $k = 1$, it just 'memorizes' the training dataset which leads to uneven class boundaries and bubbles of curious shapes. It's also clear that when the values of k are small, the classifier is sensitive to outliers although it almost ideally processes the training data.

As k increases, the boundary becomes smoother and almost turns into a straight line. Classes are split more naturally, and more errors occur on the training data. The model is light and easy to interpret.

So, which k is better? It's a very complicated question. The most sensible approach is to estimate the error rate by splitting the initial dataset into test and training samples. Let's discuss it in detail.

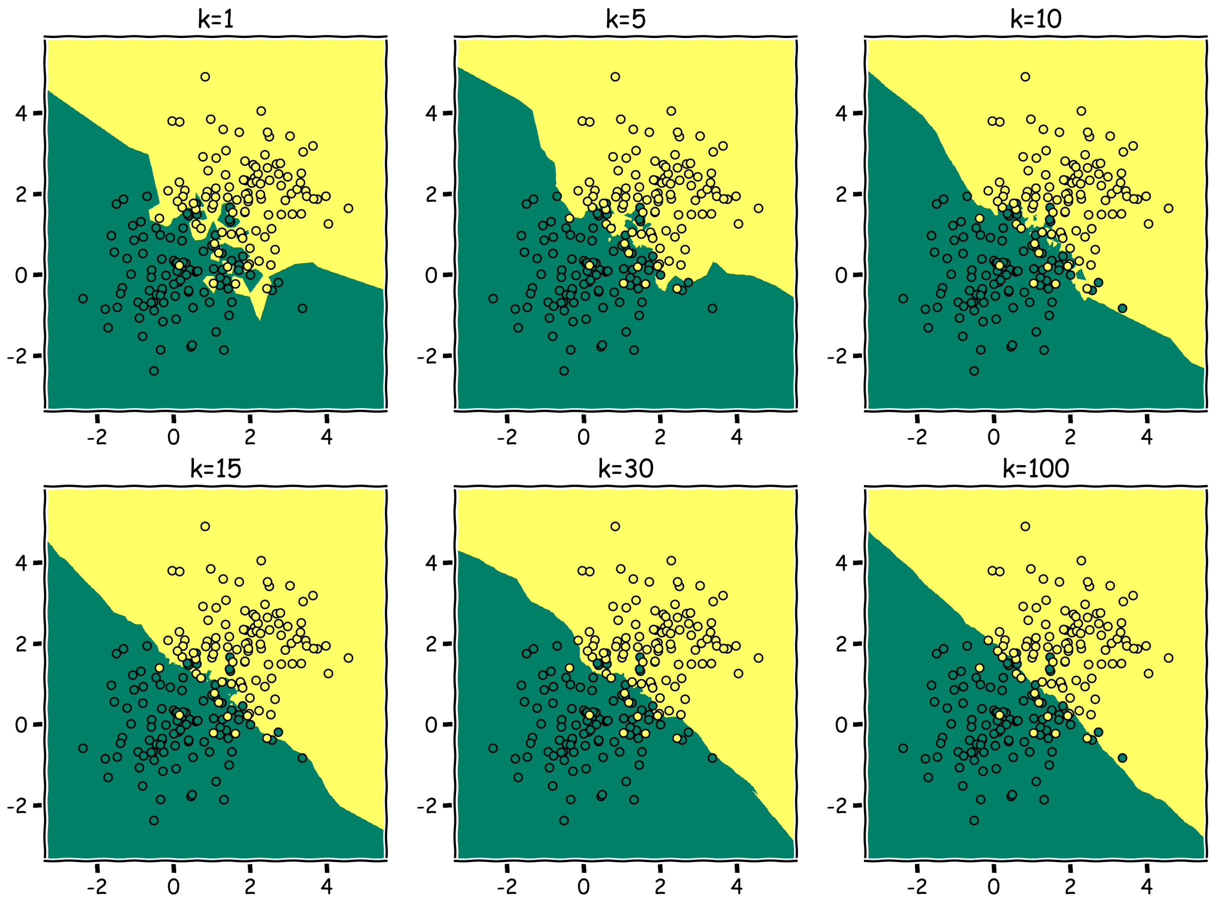

Figure 8: The impact of k value.

2 Algorithms, Algorithm Evaluation, and Dataset Splitting

So, the remained question is the choice of the parameter of the k-NN algorithm, that is, of the parameter k. The solution is not lying on the surface. To answer this question, we need to dig deeper and turn to the ideology. More precisely, we need to answer two questions first. What are we trying to achieve (generally) by solving a classification problem? What other problems arise?

We want to assign a class to a random object of the feature space (and it's better to be the right one) by choosing from the available set of classes. What other problems arise? They are quite clear. Our knowledge is limited because we have only a training dataset. We should take the experience gained on this finite dataset and transfer it to a larger data volume (probably infinite). So, our goal is to teach the algorithm to generalize the knowledge to new events that haven't happened before. When doing so using k-NN, for example, we need to consider one more thing. How to make sure that it works fine? What does this "fine" mean? To shed the light on these questions, let's begin with some basic definitions that we are going to use throughout our discussion.

2.1 Algorithms and Empirical Risk

We often talk about algorithms, their training, and assessment. But what does an algorithm mean? It turns out that everything is easy.

Definition 2.1.1 *Let X be a set of all possible objects and Y be a set of responses. The expression*

$$a : X \to Y$$

is called an algorithm.

So, an algorithm is a function that maps a random object to a response.

Remark 2.1.1 *Let's clarify the introduced notations. In the case of the simple linear regression model, we were considering a set X as a set of real numbers $\mathbb{R}$ or its continuous subset, and the algorithm was as follows:*

$$a(x) = \theta_0 + \theta_1 x.$$

In the considered example, the space X is a space

$$X = [0, 10] \times [0, 10],$$

since there are only two predictors (sweetness and crispness) that can take arbitrary values between 0 and 10 (inclusive), and Y is a three-element set:

$$Y = \{vegetable, fruit, protein\}.$$

k-NN and weighted k-NN described earlier are algorithms according to the introduced definition.

So, the next thing we need to do is to find out how to evaluate our algorithm.

Definition 2.1.2 *Let $a : X \to Y$ be an algorithm. The loss function $L(a, x)$ is an arbitrary non-negative function that characterizes the degree of error of an algorithm a on an object x.*

When there's no algorithm error on the object x, then $L(a, x) = 0$, and it's reasonable to introduce the next definition.

Definition 2.1.3 *Let $a : X \to Y$ be an algorithm, and $L(a, x)$ be a loss function. When $L(a, x) = 0$, the response $a(x)$ is called* ***correct****.*

To evaluate an algorithm, it's smart to consider its losses on multiple objects or, what's more common in statistics, to average these losses (in other words, to look at the average performance of the algorithm) instead of considering its losses on one object (because it can be an outlier). Here we come to the so-called empirical risk.

Definition 2.1.4 *Let $x_1, x_2, ..., x_n \in X$ be a dataset, $a(x) : X \to Y$ an algorithm, and $L(a, x)$ a loss function. A loss functional (empirical risk, average loss functional) is the following functional:*

$$Q(a, L, x_1, ..., x_n) = \frac{1}{n}\sum_{i=1}^{n} L(a, x_i).$$

Remark 2.1.2 *Note that the following function is often used as a loss function when solving a classification problem:*

$$L(a, x) = \mathsf{I}(a(x) \neq y(x)) =$$

$$= \begin{cases} 1 & \text{if the response } y(x) \text{ of the object } x \text{ doesn't match } a(x) \\ 0 & \text{otherwise} \end{cases},$$

where $y(x)$ is a known response of the object x. The idea behind this function is simple. The function outputs 1 when the response of the algorithm doesn't match the true response and 0 otherwise. In this case, the loss functional takes the following form:

$$Q(a, L, x_1, ..., x_n) = \frac{1}{n}\sum_{i=1}^{n} \mathsf{I}(a(x_i) \neq y(x_i))$$

and outputs nothing but the rate of incorrect responses of the classifier on a dataset $x_1, x_2, ..., x_n$. We intend to minimize the value of the functional. Remember that.

When solving the regression problem, we saw the following loss function:

$$L(a, x) = (a(x) - y(x))^2,$$

where $y(x)$ was a known response of the object x. The loss functional based on this loss function was as follows:

$$Q(a, L, x_1, ..., x_n) = \frac{1}{n}\sum_{i=1}^{n}(a(x_i) - y(x_i))^2.$$

As you remember, when solving the simple linear regression problem, the algorithm had the following form $a(x) = \theta_0 + \theta_1 x$. The loss functional in that case *took the familiar form:*

$$Q(a, L, x_1, ..., x_n) = \frac{1}{n}\sum_{i=1}^{n}(\theta_0 + \theta_1 x_i - y(x_i))^2,$$

and, to find the coefficients θ_0 and θ_1, we minimized it and solved the problem

$$\operatorname*{Arg\,min}_{\theta_0,\theta_1} \frac{1}{n}\sum_{i=1}^{n}(\theta_0 + \theta_1 x_i - y(x_i))^2 = \operatorname*{Arg\,min}_{\theta_0,\theta_1} \sum_{i=1}^{n}(\theta_0 + \theta_1 x_i - y(x_i))^2.$$

As you can see, it's the ordinary least squares method! Loss functions can take different forms but, in the meantime, we will not look into this.

Our observations and common sense show that an algorithm is better when the value of the corresponding loss functional is smaller. Hence, it makes sense to find such an algorithm or algorithms $a(x)$ that minimize the loss functional, in other words, to solve the following problem:

$$\operatorname*{Arg\,min}_{a\in\mathcal{A}} Q(a, L, x_1, ..., x_n).$$

In the described problem, $\mathcal{A}$ is a considered set of algorithms. In the case of the simple linear regression problem, the set of algorithms $\mathcal{A}$ was described as follows, for example:

$$\mathcal{A} = \{\theta_0 + \theta_1 x,\ \theta_0, \theta_1 \in \mathbb{R}\}.$$

When considering k-NN, we can consider $\mathcal{A}$ as the following set:

$$\mathcal{A} = \{\text{k-NN algorithms}, \quad k \in \mathbb{N}\}.$$

Remark 2.1.3 *The discussed examples of algorithm sets are nothing but a set of possible parameters (or hyperparameters) of some 'fixed' algorithms. In practice, we deal with more tricky sets.*

Algorithm learning is a process of choosing the best algorithm in the set $\mathcal{A}$.

Well, now we have a general formulation of the algorithm evaluation problem. At the same time, there remains the question of choosing (or constructing) a set $x_1, x_2, ..., x_n$ for evaluation. It's very important, and we shouldn't overlook it.

Remark 2.1.4 *Note that the raised question, as well as the question of choosing a set $\mathcal{A}$, is not a theoretical difficulty but rather one of the most important jumping-off points for developing a good model.*

To prove the point, let's consider a senseless example that shows how important it is to formulate a problem clearly and correctly in machine learning (and in any field). Assume that we want to evaluate an algorithm on the ***entire*** *training dataset $x_1, x_2, ..., x_n$ with responses $y_1, y_2, ..., y_n$ without limiting the set $\mathcal{A}$. What algorithm will better cope with the task? As you may guess, the following one:*

$$a(x_i) = y_i.$$

Those who listen carefully may note that the algorithm doesn't satisfy the very definition of it because the algorithm outputs a response only on the elements of the training dataset. No worries! Given $x \notin \{x_1, x_2, ..., x_n\}$, the value $a(x)$ can be any, and the following expression is always satisfied:

$$Q(a, L, x_1, ..., x_n) = 0,$$

and nothing can be better!

Well, we've developed the algorithm that is ideal according to the loss functional but useless in practice. The reason is not that we didn't limit the set $\mathcal{A}$. The problem, rather, is that the algorithm was **trained** on the sample $x_1, x_2, ..., x_n$ for which it had **seen** the correct answers. However, we would like to see how it performs on new **unseen** data. To evaluate it, we should know the correct answers. Let's see how it works in practice.

2.2 Training Dataset Splitting

As follows from the discussion, we are interested in finding such an algorithm $a(x)$ for which the loss functional value is small on a new unseen dataset, as long as the data has the same patterns as the training dataset. The last somehow odd phrase will be clear a bit later when we touch upon the probabilistic background to the classification problem. For now, we need to understand it in the sense of the **compactness hypothesis** we discussed earlier.

Remark 2.2.1 *Note that, in our opinion, this point is very important because it attempts to explain the ideology of training and testing machine learning algorithms. Even exceptional algorithms may not work well without proper training.*

Well, so let's return to the subject matter. Where do we get data that the algorithm hasn't seen if we have only the training dataset

$$D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}?$$

The first idea is to split the set into two parts D_{train} and D_{test} and train the model on the narrowed training dataset D_{train}, and then evaluate the results on the dataset D_{test}. Note that we artificially narrow down the initial training dataset

D to the dataset D_{train}. This way we can check how the trained algorithm $a(x)$ works on unseen data by calculating the loss functional

$$Q(a, L, D_{test})$$

on the test set D_{test}.

Remark 2.2.2 *The questions are what ratio works best and how the training dataset D should be split. A common ratio of $D_{train} : D_{test}$ is about $80 : 20$ but it's is very loosely. The next question is a delicate one. How to divide?*

Since there's no information about the time of collecting data from the set D, and the dataset D is representative, we can hardly find something better than random splitting.

If there's data about the time of collecting, it's better to set a certain timing for a split. For example, when the weekly data is collected (and we presume it is homogeneous), the data collected before Thursday (for example) should form a training dataset, and the data collected after Thursday should form a test dataset. This way we completely discard the information about the future and don't take it into account. That's exactly what we want the algorithm to do: learn today, work tomorrow. These are clearly empirical approaches to algorithm evaluation.

It turns out that professionals do it in a different way in practice. They split a dataset into three parts instead of two by chopping a so-called validation part D_{val} off D_{test}. It's a nontrivial and important point. Can you guess why they do it like that? Well, let's find out.

Imagine you have different models of several k-NN classifiers with different values of k, and each of these classifiers has been trained on the training dataset D_{train} and evaluated on the test dataset D_{test}. Which one do you choose? The one which loss functional outputs smaller value on the test dataset, right? If yes, there's a mistake (not ideological, but conceptual). The final decision is based on some information about the future, which is not good. We've been trying to avoid using unknown data for training because of negative consequences. Such reuse of the test dataset often leads to the so-called overfitting, and it is not doing us any good.

Here's an important observation. When the test dataset D_{test} is used too often, the algorithm begins to learn on the dataset D_{test} as well, which contradicts everything that has been said earlier.

It's time to practice. Since we have the dataset D_{val}, we need to choose the algorithm that makes fewer errors on this dataset, train the algorithm on the entire dataset D_{train} (including the cut-off dataset D_{val}), and evaluate the algorithm on D_{test}. It is reasonable to consider the obtained value of the loss functional on D_{test} the "average" error of the trained algorithm. Next, we are going to discuss the so-called k-fold cross-validation, which generalizes the idea described earlier.

2.3 k-Fold Cross-Validation

So, suppose we have a set of algorithms. How to choose the best one using k-fold cross-validation? The process of choosing is simple. The figure makes it clear (Fig. 9):

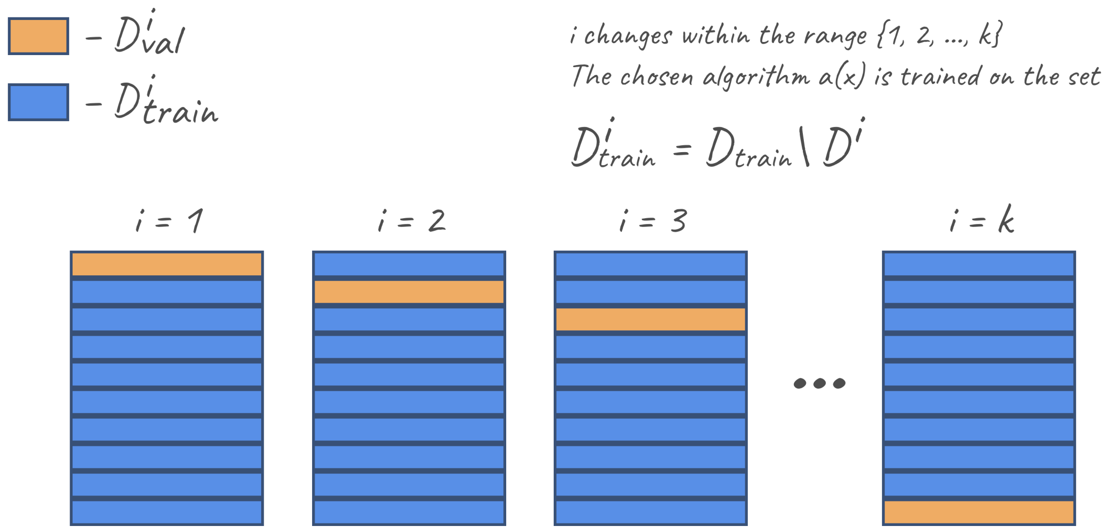
- D^i_{val}
- D^i_{train}
i changes within the range {1, 2, ..., k}
The chosen algorithm a(x) is trained on the set
$D^i_{train} = D_{train} \setminus D^i$
i = 1
i = 2
i = 3
i = k

Figure 9: k-fold cross-validation.

Let's formally describe the algorithm.

1. The initial dataset is divided into D_{train} and D_{test}.
2. The dataset D_{train} is split into k disjoint folds (sets) of almost the same size:
$$D_{train} = D_1 \cup D_2 \cup \ldots \cup D_k.$$
3. Let i change within the range $\{1, 2, ..., k\}$:
   (a) The chosen algorithm $a(x)$ is trained on the dataset:
   $$D^i_{train} = D_{train} \setminus D_i.$$
   (b) The chosen algorithm $a(x)$ is tested on the dataset $D^i_{test} = D_i$, and the following value is calculated:
   $$\mathsf{Loss}_i(a) = Q(a, L, D^i_{test}).$$
4. The final estimator of the algorithm $a(x)$ error as a result of k-fold cross-validation is as follows:
$$\mathsf{Loss}(a) = \frac{1}{k}\sum_{i=1}^{k} \mathsf{Loss}_i(a).$$

5. Steps $3 - 4$ are repeated for each evaluated (compared) algorithm.
6. The algorithm a with the smallest value of $\mathsf{Loss}(a)$ is considered the best.
7. The chosen algorithm is trained on the set D_{train} and tested on the set D_{test}. The error $Q(a, L, D_{test})$ is deemed to be an "average" error of the best algorithm.

The training dataset is split into several small pieces each of which is used as a test dataset in the respective iteration to evaluate the algorithm. The final evaluation of each algorithm consists of errors averaged in each iteration.

2.4 Example: Weighted k-NN and k-Fold Cross-Validation

Let's return to our example with products and use cross-validation given $k = 3$ to find out which algorithm is better (weighted k-NN given k = 3, 4, or ordinary k-NN given $k = 4$). Assume that the data has been split into D_{train} and D_{test}, and D_{train} is a familiar dataset:

Product	Sweetness	Crispness	Class
banana	10	1	fruit
orange	7	4	fruit
grapes	8	3	fruit
shrimp	2	2	protein
bacon	1	5	protein
nuts	3	3	protein
cheese	2	1	protein
fish	3	2	protein
cucumber	2	8	vegetable
apple	9	8	fruit
carrot	4	10	vegetable
celery	2	9	vegetable
iceberg lettuce	3	7	vegetable
pear	8	7	fruit

The size of the training dataset D_{train} is 14, and we can choose the folds of the following sizes: $5, 5, 4$.

At the first step, the first fold will be a test fold (Table 1), and the second and third folds will be training folds (Table 2). Let's plot the data from the training dataset (Fig. 10) in the plane and classify the products in the test dataset using weighted k-NN when the parameter k equals 4. We will use the Euclidean distance

Product	Sweetness	Crispness	Class
banana	10	1	fruit
orange	7	4	fruit
grapes	8	3	fruit
shrimp	2	2	protein
bacon	1	5	protein

Table 1: Iteration 1. Test data.

Product	Sweetness	Crispness	Class
nuts	3	3	protein
cheese	2	1	protein
fish	3	2	protein
cucumber	2	8	vegetable
apple	9	8	fruit
carrot	4	10	vegetable
celery	2	9	vegetable
iceberg lettuce	3	7	vegetable
pear	8	7	fruit

Table 2: Iteration 1. Training data.

as a metric, and ω_i as a weight. The latter is a value proportional to the inverse of the squared distance between the objects x_i and the classified object z:

$$\omega_i = \frac{1}{d^2(x_i, z)}.$$

Let's find the distance from the banana to the other elements of the training dataset as well as their weights. For example, the distance from the banana to the cheese equals:

$$d_2\,(\text{banana, cheese}) = \sqrt{(10-2)^2 + (1-1)^2} = 8.$$

In this case, the weight of the training object cheese equals:

$$w_{\text{banana, cheese}} = \frac{1}{d_2^2\,(\text{banana, cheese})} = \frac{1}{64} \approx 0.016.$$

Similarly, we calculate distances to other objects and their weights. The data is shown in Table 3. We assign the class according to four nearest neighbors, which are pear, apple, fish, and nuts. In the case of the ordinary k-NN, we would have encountered a problem of two competing classes (fruit and protein). Weighted

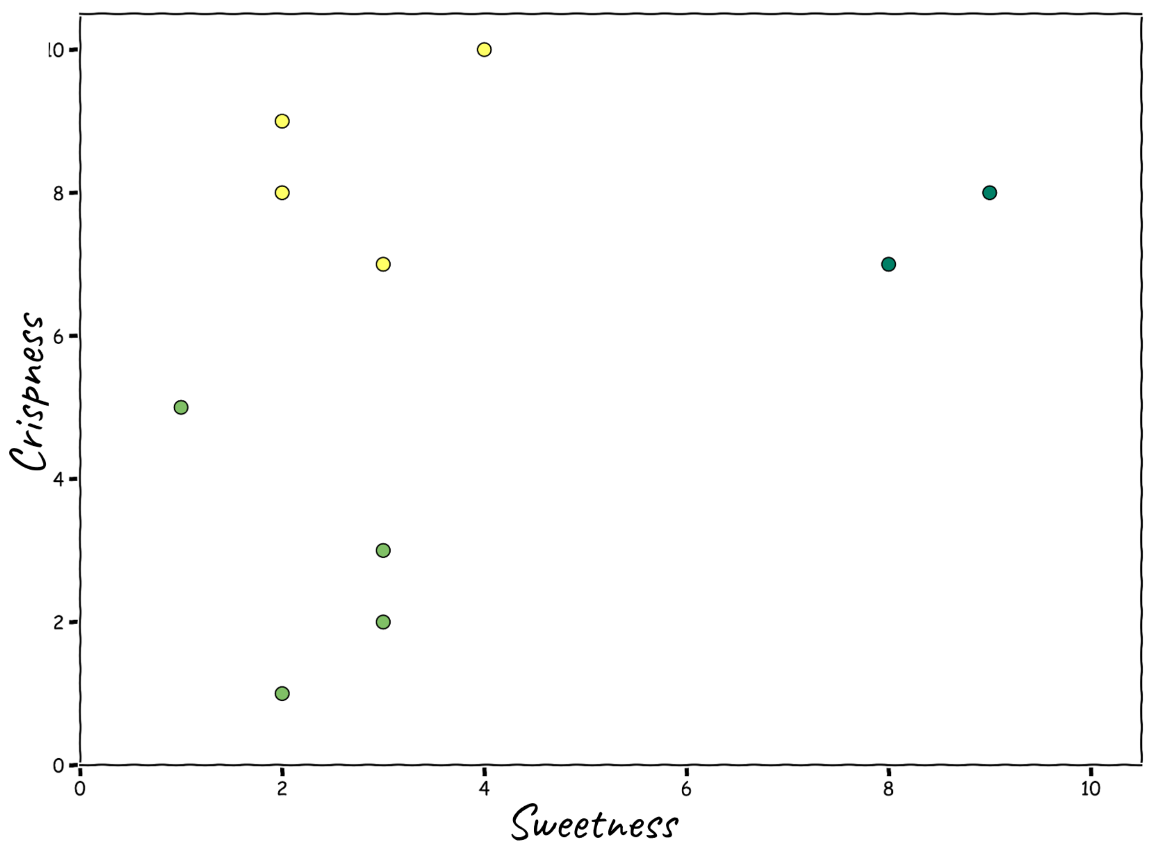

Figure 10: Iteration 1. Training data.

k-NN avoids this problem. The sum of the fruit weights is 0.045, and that of protein is 0.039. Thus, the banana will be classified as a fruit.

Similar calculations are performed for each product in the test dataset (Table 4). You can perform calculations later to test yourself, but now let's jump right into the classification results.

The table shows that there's 1 error (caused by bacon). Hence, when using the loss function that is standard for classification problems:

$$L(a, x) = \mathsf{I}(a(x) \neq y(x)),$$

the value of the loss functional will be equal to 0.2:

$$\mathsf{Loss}_1(\text{W. 4-NN}) = Q(a, L, D^1_{train}) = \frac{1}{5} = 0.2.$$

Later, you can review the results of iterations 2 and 3. The steps are similar to the ones described earlier (Tables 5, 6). The classes assigned in these iterations perfectly match the initial classes. Hence, the error rate is zero, and the mean value of the errors in three iterations is about 0.07:

$$\mathsf{Loss}(\text{W. 4-NN}) = \frac{1}{3}\sum_{i=1}^{3} \mathsf{Loss}_i = \frac{1}{15} \approx 0.07.$$

Product	Class	Distance	Weight
nuts	protein	7.28	0.019
cheese	protein	8	0.016
fish	protein	7.071	0.020
cucumber	vegetable	10.63	0.009
apple	fruit	7.071	0.020
carrot	vegetable	10.817	0.009
celery	vegetable	11.314	0.008
iceberg lettuce	vegetable	9.22	0.012
pear	fruit	6.325	0.025

Table 3: Banana distances and weights

Product	Sweetness	Crispness	Initial class	Assigned class
banana	10	1	fruit	fruit
orange	7	4	fruit	fruit
grapes	8	3	fruit	fruit
shrimp	2	2	protein	protein
bacon	1	5	protein	vegetable

Table 4: Iteration 1. Test data.

When k in the weighted k-NN equals 3, then, for a new algorithm,

$$\mathsf{Loss}(\text{W. 3-NN}) = \frac{1}{3}\sum_{i=1}^{3}\mathsf{Loss}_i = \frac{1}{5} = 0.2.$$

When $k = 4$, and neighbors are not weighted, for such an algorithm,

$$\mathsf{Loss}(\text{4-NN}) = \frac{1}{3}\sum_{i=1}^{3}\mathsf{Loss}_i = \frac{1}{4} = 0.25.$$

According to the described algorithm, we need to choose the classifier with the smallest evaluated average error Loss from the set. Among the considered algorithms, the smallest error is the error Loss(W. 4-NN). Thus, among the evaluated algorithms, we choose the weighted 4-NN and use the entire set D_{train} as a training dataset for predictions (training). To estimate the true average error of the chosen algorithm, we can test it on the prepared dataset D_{test}.

2.5 Curse of Dimensionality

Having discussed k-NN, let's turn to another metrical machine learning problem called a curse of dimensionality.

Product	Sweetness	Crispness	Initial class	Assigned class
nuts	3	3	protein	protein
cheese	2	1	protein	protein
fish	3	2	protein	protein
cucumber	2	8	vegetable	vegetable
apple	9	8	fruit	fruit

Table 5: Iteration 2. Test data.

Product	Sweetness	Crispness	Initial class	Assigned class
carrot	4	10	vegetable	vegetable
celery	2	9	vegetable	vegetable
iceberg lettuce	3	7	vegetable	vegetable
pear	8	7	fruit	fruit

Table 6: Iteration 3. Test data.

Let's think about the problem. Assume that our training data (n items) is uniformly distributed within a unit hypercube in a space $\mathbb{R}^d$ given quite large d. A quite large value of d means that each object has a relatively large number of features (d items).

Let's place a new test object in the hypercube and answer the question: What length l should have the side of the hypercube surrounding the test object so that this hypercube contains k nearest neighbors (Fig. 11), that is, k points of the training dataset?

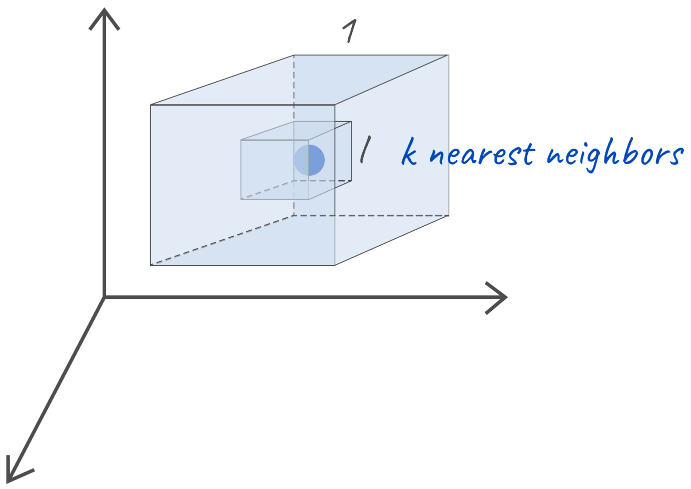

Figure 11: Multiboxing.

In this case, the math is quite simple. Since the volume of the hypercube equals l^d and the training data is uniformly distributed in the unit hypercube, it should contain k nearest neighbors,

$$l^d \approx \frac{k}{n} \Rightarrow l \approx \left(\frac{k}{n}\right)^{1/d}.$$

At first glance, it is not surprising, but let's do the math. First of all, we need some reasonable parameters. Let $k = 10$, $n = 1000$, for example. Hence, let's examine the edge length given different d, considering that

$$l \approx \left(\frac{1}{100}\right)^{1/d}.$$

When $d = 2$, we get 0.1. Not bad. When $d = 10$, we obtain about 0.631. Given $d = 100$, we get about 0.955, and given $d = 1000$ about 0.995. Figure 12 shows an attempt to illustrate the relationships.

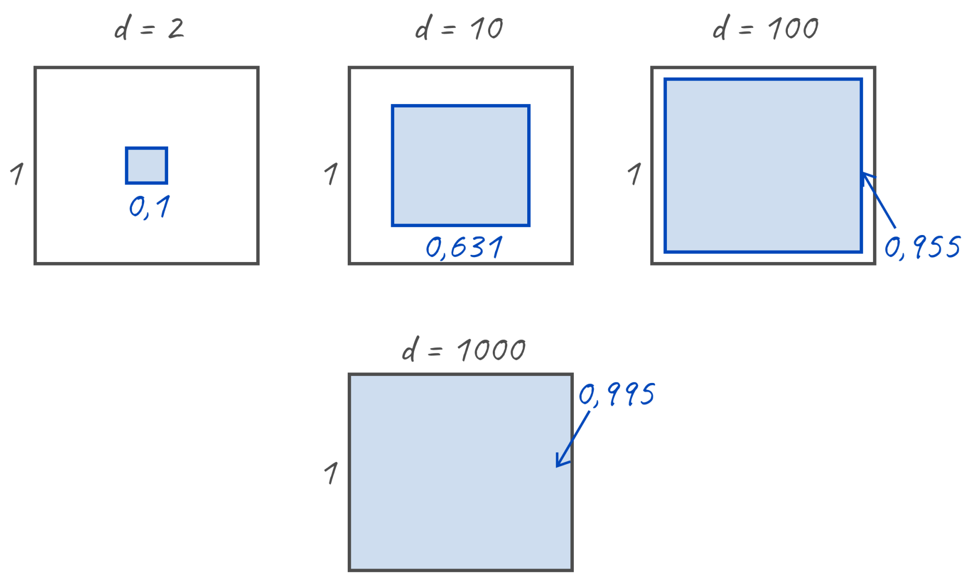

Figure 12: Filling the space.

What do we see? Well, the greater the value of d, the more distanced the objects are from each other and, roughly speaking, the closer they are to the edges of the hypercube. Note that they are not inside the hypercube. Otherwise, we could have taken a smaller edge of the hypercube containing k neighbors and the space would not have been filled like that. This observation poses a problem because it reveals that k-NN is unsound. When d increases, the nearest neighbors are on the edges of the hypercube along with the rest of the training data. When we select a random neighbor, we find many training data objects nearby that were not included in the nearest neighbors. What kind of compactness is it?

All these arguments are natural in math but not for our understanding of reality. And that's okay because we are used to three dimensions of space. So, now we will explain what's going on but in the language of probability. Let's find the probability that a random point in the hypercube is inside it but not on the edge. For that, we need each one-dimensional coordinate of the point to lie on a line segment, for example, $[0, 1]$, but not on its border. We move $\varepsilon > 0$ away from border points, and the probability of lying on a line segment equals $(1 - 2\varepsilon)$ (Fig. 13). For d-dimensional space (due to independence of features), the probability

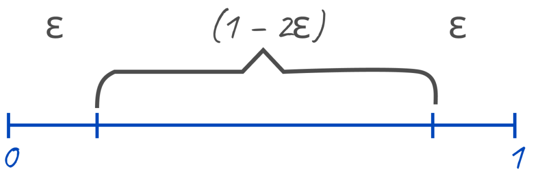

Figure 13: The probabilistic explanation.

of being inside the hypercube equals

$$(1 - 2\varepsilon)^d,$$

and the last expression approaches zero very fast as d increases. So, when values of d are high, the probability of being inside the hypercube is almost zero.

After this discussion, it may seem that metric approaches are completely unsound. Of course, it's not true. We discussed cases when data is uniformly distributed, which is rare. It's more common when data fills a subspace of the space $\mathbb{R}^d$ (a hyperplane, a more complicated variety), and k-NN works well. Our discussion indicates the importance of data preprocessing, dimensionality reduction, and other extravagances because, in practice, it's rare when all predictors of the input dataset are important.

Well, we discussed metric classifiers, and it's time to move on to an alternative — probabilistic classifiers.

3 Naive Bayes Classifier

3.1 Introductory Notes

Earlier we discussed one of the ways to solve a classification problem, which was a metric classifier k-NN. To classify a new object, we used only the available training data with no (even presumed) probabilistic relationship between predictors and responses, although we mentioned it in the Introduction. We also used the compactness hypothesis. Now, we are going to discuss a classification approach of a probabilistic nature. Our main tool will be the well-known Bayes theorem.

The need in probabilistic models (and the their adequacy) is supported by the following observations. In supervised learning problems, the elements of a training dataset are not real objects, but the available data about real objects. This observation is fundamental. For example, data may be **inaccurate**. Measurements of feature values (attributes) for training objects and measurements of responses are often inaccurate (at least because of rounding). Just think about the regression model assumptions discussed earlier. Moreover, data may be **incomplete** because we don't usually measure all possible features but focus on those that are available. You know that sometimes it's not clear what features are inherent in an object and what features are not. Consequently, one description of the object x may correspond to different objects or different responses in reality. That's why the assumption about the explicit relation that outputs the only correct response seems naive. Probability models can, in a sense, eliminate such errors. Moreover, explicit relations are a special case of probabilistic ones, but we will not focus on this matter.

3.2 Probabilistic Background

Unlike in the case of k-NN, we will not begin with considering the illustrative geometrical considerations, but we will rather look at the probabilistic background and the algorithm, and then consider its application in practice.

To begin with, we will review some probability theory facts. So, let's start with the definition of conditional probability.

Definition 3.2.1 *Let the event B be so that $\mathsf{P}(B) > 0$. Then, the conditional probability of the event A given that the event B has occurred is the value*

$$\mathsf{P}(A|B) = \frac{\mathsf{P}(A \cap B)}{\mathsf{P}(B)}.$$

Thus, conditional probability is the ratio of the probability of $A \cap B$ (that both A and B have occurred) to the probability of B. In other words, we find the ratio of the probability of the rate of event A given B to the probability of event B, which is renormalization or simple changing (narrowing) of the probability space.

This definition can be rewritten as:

$$\mathsf{P}(A \cap B) = \mathsf{P}(A|B)\mathsf{P}(B).$$

Moreover, if $\mathsf{P}(A) > 0$, we can also consider the following conditional probability:

$$\mathsf{P}(B|A) = \frac{\mathsf{P}(B \cap A)}{\mathsf{P}(A)} = \frac{\mathsf{P}(A \cap B)}{\mathsf{P}(A)},$$

hence,

$$\mathsf{P}(A \cap B) = \mathsf{P}(B|A)\mathsf{P}(A).$$

After combining the written expressions, we get the so-called **multiplication rules**:

$$\mathsf{P}\left(A \cap B\right) = \mathsf{P}(A|B)\mathsf{P}(B) = \mathsf{P}(B|A)\mathsf{P}(A).$$

We replace the numerator in the definition of the conditional probability with the second ratio to obtain the **Bayes' formula** that is widely used in probability theory, as well as in machine learning:

$$\mathsf{P}(A|B) = \frac{\mathsf{P}(B|A)\mathsf{P}(A)}{\mathsf{P}(B)}.$$

Note that the conditional probabilities displaced each other in the formula. Let's name the components of the formula.

Definition 3.2.2 *The probability* $\mathsf{P}(B|A)$ *is known as a likelihood. The probabilities* $\mathsf{P}(A)$ *and* $\mathsf{P}(B)$ *are prior probabilities. The probability* $\mathsf{P}(A|B)$ *is a posterior probability.*

However, the question remains, how the introduced concepts can help us, and what they have to do with machine learning? To figure this out, we are going to review the problem of classifying spam and non-spam emails and see what we can get. An example of training data is in the table. To simplify the task, we assume that there's one predictor that takes two values only, 0 and 1, and the value that is equal to 1 means that the email contains the word prize. The spam column is a response that takes only two values (0 and 1), and the value that is equal to 1 means that the email is classified as spam.

Email	**prize**	**spam**
You've won the prize!	1	1
Make the first month's payment and receive a prize.	1	1
Recommend your bank and receive a prize.	1	0
You've inherited a house.	0	1
......	...	...

Assume that 80 emails were analyzed. 5 spam emails and 3 ham emails contained the word prize. The remaining 72 emails didn't contain the word prize, and 27 of them were marked as spam and the remaining 45 were marked as ham. We fill in the table with the data:

	spam	**ham**
emails in total	32	48
there is 'prize'	5	3
there is no 'prize'	27	45

.

So, there are 32 emails marked as spam, and 5 of them contain the word prize, and 48 emails marked as ham, and 3 of them contain the word prize. The question is how to estimate the probability that email belongs to the spam class if it contains the word prize.

Let's recollect how to calculate event probabilities based on the data. In our case, true probabilities are unknown, but we can estimate them using frequency. For example, the probability that a random email classified as spam contains the word prize is estimated according to the frequency of 5/32 because, based on the training data, only 32 emails are classified as spam, and exactly 5 of them contain the word prize. According to the data, it's reasonable to assume that

$$\mathsf{P}(\text{prize} \mid \text{spam}) = \frac{5}{32}.$$

In turn, the probability that the received email is spam is estimated by 32/80, and the probability that the message contains the word prize is estimated by 8/80:

$$\mathsf{P}(\text{spam}) = \frac{32}{80},\ \mathsf{P}(\text{prize}) = \frac{8}{80}.$$

We can use the Bayes' formula to find the probability that the email is spam if it contains the word prize:

$$\mathsf{P}(\text{spam} \mid \text{prize}) = \frac{\mathsf{P}(\text{prize} \mid \text{spam}) \cdot \mathsf{P}(\text{spam})}{\mathsf{P}(\text{prize})} = \frac{\frac{5}{32} \cdot \frac{32}{80}}{\frac{8}{80}} = \frac{5}{8} = 0.625.$$

As you can see, the email containing the word prize is more likely classified as spam rather than ham. We can find the probabilities of other events in the same way. So, you've dusted off your knowledge of how to find probabilities based on frequencies. Well, let's move on to the classifier construction.

3.3 Model Development and Formula Derivation

First, we need to turn to the probabilistic formulation of the problem. Let X be a set of objects, and each object is described by p features that are random variables $X_1, X_2, ..., X_p$ and a response Y (note that the response takes a finite set of values). We will think of $X \times Y$ as of a probability space with some joint probability distribution. Moreover, for ease of understanding, we will assume that all random variables $X_1, X_2, ..., X_p$ have a **discrete distribution with a finite set of values**. Our task is to assign a new test object with predictors $X_1, X_2, ..., X_p$ one class label in the set Y of possible classes. How do we do it? It's probably clear that the first step is to calculate the probabilities of assigning a new object to each of the available classes, that is a set of probabilities:

$$\mathsf{P}(\text{Class} = y | X_1, X_2, ..., X_p), \quad y \in Y.$$

Remark 3.3.1 *It's important to understand what this (non-formal) expression means given fixed $y \in Y$. It shows the probability of assigning the object of a previously unknown class to the class y considering that the object has attributes $X_1, X_2, ..., X_p$ (formally, this expression defines the corresponding probability distribution).*

So, we have a set of probabilities. How to understand which class $y^* \in Y$ to assign to the test object? Of course, the most probable class (or one of the most probable classes if there are many of them), which means to choose such $y^* \in Y$ so that

$$y^* = \underset{y \in Y}{\operatorname{Arg\,max}} \; \mathsf{P}(\text{Class} = y | X_1, X_2, ..., X_p)$$

According to the Bayes' formula, the last expression can be rewritten as follows:

$$y^* = \underset{y \in Y}{\operatorname{Arg\,max}} \; \mathsf{P}(\text{Class} = y | X_1, X_2, ..., X_p) =$$

$$= \underset{y \in Y}{\operatorname{Arg\,max}} \frac{\mathsf{P}(X_1, X_2, \ldots, X_p | \text{Class} = y)\mathsf{P}(\text{Class} = y)}{\mathsf{P}(X_1, X_2, \ldots, X_p)}.$$

Since the probability in the denominator doesn't depend on y, it's equal for all y and doesn't affect maximization. Hence,

$$y^* = \underset{y \in Y}{\operatorname{Arg\,max}} \left(\mathsf{P}(X_1, X_2, \ldots, X_p | \text{Class} = y)\mathsf{P}(\text{Class} = y)\right).$$

It's all right in theory, but what about practice? The problem is that we don't know the joint probability distribution in practice. It's easy to estimate the probabilities $\mathsf{P}(\text{Class} = y)$. To do so, we can estimate the frequency of classes in the training dataset. However, it's not so easy to estimate the conditional probabilities $\mathsf{P}(X_1, X_2, \ldots, X_p | \text{Class} = y)$ because there are too many of them. To estimate them properly, we need to observe each case several times, but it's impossible. That's why the classifier is naive.

The naive Bayes classifier assumes that **predictors are conditionally independent given the class label**. In math, it means the following:

$$\mathsf{P}(X_1, X_2, \ldots, X_p | \text{Class} = y) = \mathsf{P}\left(X_1 | \text{Class} = y\right) \cdot \mathsf{P}\left(X_2 | \text{Class} = y\right) \cdot \ldots \cdot$$

$$\mathsf{P}\left(X_p | \text{Class} = y\right) = \prod_{i=1}^{p} \mathsf{P}\left(X_i | \text{Class} = y\right).$$

Despite the significant limitation, the Bayes classifier is coping well with the task. Moreover, until recently, many web services used naive Bayes classifiers for spam recognition.

Remark 3.3.2 *Note that, in the classification of emails or texts, for example, a naive limitation means that different words occur independently from one another in a text on one topic. Just think how far it is from reality.*

How does the independence assumption help us? Of course, it helps us to estimate conditional probabilities $\mathsf{P}(X_i|\text{Class} = y)$. These probabilities are estimated by frequency as usual. We will discuss it in detail a little bit later. Given the naive assumption, the classifier takes the following form:

$$y^* = \operatorname*{Arg\,max}_{y\in Y} \left(\mathsf{P}(\text{Class} = y) \prod_{i=1}^{p} \mathsf{P}\left(X_i|\text{Class} = y\right) \right).$$

However, that's not all. When the size of the training dataset is large and probabilities (or their estimates) are low, the result of multiplication on a computer with limited precision will be something like zero. To solve this problem, we need natural logarithms. Since a natural logarithm is an increasing function, it makes maxima transition to maxima, and the classifier takes the following form:

$$y^* = \operatorname*{Arg\,max}_{y\in Y} \left(\ln \left(\mathsf{P}(\text{Class} = y) \prod_{i=1}^{p} \mathsf{P}\left(X_i|\text{Class} = y\right) \right) \right),$$

or, according to the properties of logarithms, as follows:

$$y^* = \operatorname*{Arg\,max}_{y\in Y} \left(\ln \mathsf{P}(\text{Class} = y) + \sum_{i=1}^{p} \ln \mathsf{P}\left(X_i|\text{Class} = y\right) \right).$$

We are almost done. Here's the last question. How to return to the probabilities of assigning to classes? It's quite easy. We need to get rid of the logarithms and normalize the results. For each class $y \in Y$, let's calculate the value

$$F(y) = \ln \mathsf{P}(\text{Class} = y) + \sum_{i=1}^{p} \ln \mathsf{P}\left(X_i|\text{Class} = y\right).$$

Then, to find the probability of assigning an object X to a certain class $y^* \in Y$, we need the following ratio:

$$\mathsf{P}(\text{Class} = y^*|X_1, X_2, ..., X_p) = \frac{e^{F(y^*)}}{\sum\limits_{y\in Y} e^{F(y)}}.$$

Remark 3.3.3 *To lower the number of exponential operations, we can make the below transformation:*

$$\mathsf{P}(Class = y^*|X_1, X_2, ..., X_p) = \frac{1}{e^{-F(y^*)} \sum\limits_{y \in Y} e^{F(y)}} = \frac{1}{1 + \sum\limits_{y \in Y,\ y \neq y^*} e^{F(y) - F(y^*)}},$$

where the index of the last sum includes all class labels, except y^*.

So, we are ready to formulate an algorithm of classifier construction on a certain sample.

3.3.1 An Algorithm of Classifier Construction on a Certain Sample

Let $X = (x_1, x_2, ..., x_n)$ be a training dataset of size n,

$$x_i = (x_{i1}, x_{i2}, ..., x_{ip}), \quad i \in \{1, 2, ..., n\},$$

and each object x_i corresponds to the response $y_i \in Y$. Assume that the task is to classify a new object z with predictor values $(z_1, z_2, ..., z_p)$. The object z is assigned any class in the set

$$\operatorname*{Arg\,max}_{y \in Y} \left(\ln \mathsf{P}(\text{Class} = y) + \sum_{i=1}^{p} \ln \mathsf{P}\left(X_i | \text{Class} = y\right) \right),$$

where the probability estimator $\mathsf{P}(\text{Class} = y)$ is the frequency of the class in the training dataset, that is

$$\mathsf{P}(\text{Class} = y) = \frac{\sum\limits_{i=1}^{n} \mathsf{I}(y_i = y)}{n} =$$

$$= \frac{\text{the number of training elements with the class label } y}{\text{the total number of training elements}},$$

and the probability estimator $\mathsf{P}\left(X_i | \text{Class} = y\right)$ is the frequency of the predictor value z_i in the training dataset of the class y, that is

$$\mathsf{P}(X_i | \text{Class} = y) = \frac{\sum\limits_{k=1}^{n} \mathsf{I}(x_{ki} = z_i, y_k = y)}{\sum\limits_{k=1}^{n} \mathsf{I}(y_k = y)}$$

$$= \frac{\text{the number of training elements of the class } y \text{ for which } X_i = z_i}{\text{the number of training elements of the class } y}.$$

Remark 3.3.4 *Note that the introduced probability estimators are maximum likelihood estimators. The proof is quite technical, so you can find it in the additional materials.*

3.4 An Example of Classifier Construction

To examine the described algorithm in greater detail, let's consider the example of Naive Bayes classifier construction.

Let's assume that you collected football statistics about the games on the playground near your house. On the day of the game, you logged weather conditions (Rainy, Cloudy, or Sunny), average temperature (Hot, Cold, Breezy), humidity (Wet or Dry), and wind (Yes or No). The results are given in the table:

Weather	**Average temperature**	**Humidity**	**Wind**	**Game**
Sunny	Hot	Wet	No	No
Sunny	Hot	Wet	Yes	No
Cloudy	Hot	Wet	No	Yes
Rainy	Breezy	Dry	No	Yes
Rainy	Cold	Dry	No	Yes
Rainy	Cold	Dry	Yes	No
Cloudy	Breezy	Wet	Yes	Yes
Cloudy	Breezy	Wet	No	No
Cloudy	Breezy	Wet	Yes	Yes

.

The task of the naive Bayes classifier is to decide which of the two classes a new observation z with the predictors $Cloudy, Cold, Wet, Yes$ belongs.

So, we have four predictors. X_1 is Weather that takes 3 different values, X_2 is Average temperature that also takes three different values, X_3 is Humidity that takes 2 different values, and X_4 is Wind that also takes 2 different values. There's also a binary response Y. It is simply whether a game was played or not. According to the algorithm, a new object z is assigned any class in the set

$$\operatorname*{Arg\,max}_{y \in Y} \left(\ln \mathsf{P}(\text{Class} = y) + \sum_{i=1}^{4} \ln \mathsf{P}\left(X_i | \text{Class} = y\right) \right).$$

Let's look closely at the calculations in the case when the game was played, that is, for the class Yes. First, we calculate

$$\mathsf{P}(\text{Class} = \text{Yes}) = \frac{\text{the number of training elements with the class label Yes}}{\text{the total number of training elements}}.$$

It is easy to understand that we have 9 training items and only 5 of them have the response Yes. Therefore,

$$\mathsf{P}(\text{Class} = \text{Yes}) = \frac{5}{9}.$$

Next, we calculate

$$\mathsf{P}(X_1|\text{Class} = \text{Yes}) = \mathsf{P}(\text{Weather}|\text{Class} = \text{Yes}).$$

For the test item, the value of the Weather predictor X_1 is Cloudy. According to the algorithm, we need to divide the set of training objects with the response Yes and the Cloudy value of the Weather predictor by the total number of training objects of the class Yes. Hence,

$$\mathsf{P}(X_1|\text{Class} = \text{Yes}) = \mathsf{P}(\text{Weather} = \text{Cloudy}|\text{Class} = \text{Yes}) = \frac{3}{5}.$$

The next case is similar. For the test item, the value of the Average temperature predictor X_2 is Cold. According to the algorithm, we need to divide the number of training objects with the response Yes and the Cold value of the Average temperature predictor by the total number of training objects of the class Yes. Hence,

$$\mathsf{P}(X_2|\text{Class} = \text{Yes}) = \mathsf{P}(\text{Average temperature} = \text{Cold}|\text{Class} = \text{Yes}) = \frac{1}{5}.$$

Consequently, we get

$$\mathsf{P}(X_3|\text{Class} = \text{Yes}) = \mathsf{P}(\text{Humidity} = \text{Wet}|\text{Class} = \text{Yes}) = \frac{3}{5}$$

and

$$\mathsf{P}(X_4|\text{Class} = \text{Yes}) = \mathsf{P}(\text{Wind} = \text{Yes}|\text{Class} = \text{Yes}) = \frac{2}{5}.$$

Thus, for the class Yes, the value of the expression equals

$$\ln\frac{5}{9} + \ln\frac{3}{5} + \ln\frac{1}{5} + \ln\frac{3}{5} + \ln\frac{2}{5} \approx -4.135.$$

Similar calculations for the class No lead us to the following result:

$$\ln\frac{4}{9} + \ln\frac{1}{4} + \ln\frac{1}{4} + \ln\frac{3}{4} + \ln\frac{2}{4} \approx -4.564.$$

Since the first of two expressions is larger, the class Yes should be assigned to the test observation. Thus, there likely will be a game when it's cloudy, wet, windy, and cold. So, the classification problem is solved. However, how sure is the classifier in its decision? Let's see. In the introduced notations,

$$F(\text{Yes}) \approx -4.135, \quad F(\text{No}) \approx -4.564,$$

then,

$$\mathsf{P}(\text{Class} = \text{Yes}|X_1, X_2, X_3, X_4) \approx \frac{1}{1 + e^{-4.564+4.135}} \approx 0.606,$$

therefore,

$$\mathsf{P}(\text{Class} = \text{No}|X_1, X_2, X_3, X_4) = 1 - \mathsf{P}(\text{Class} = \text{Yes}|X_1, X_2, X_3, X_4) \approx 0.394.$$

The classifier is not confident about assigning the observation to the class Yes.

3.5 Email Classification. Laplace Smoothing

We have assumed so far that the number of predictors for any object is known in advance. But it's not always the case. Let's go back to the email classification problem and consider it in detail. What are predictors in an email? Words of course. Can we be sure about the number of words an email contains? No, we cannot.

Perhaps, it's reasonable to assume that the number of predictors in each email is equal to the number of the words it contains. The predictors take values in the dataset V, which is a dictionary of words contained in the training data. Well, the dictionary is a large table of three columns. The first column contains a word, the second column shows the number of instances of this word in emails classified as spam, the third column contains the number of instances of this word in emails classified as ham.

Let's jump right into the example. Our training data will be three emails with the following content:

- 'Win a million rubles' is spam.
- 'Ruble drops again' is ham.
- 'A million ways to get rich' is spam.

Let's create the dictionary V. In practice, natural language processing is first applied to preprocess raw data, and we will consider the words 'ruble' and 'rubles' as the same word and disregard articles and particles. We get the following table.

Word (X)	**spam**	**ham**
win	1	0
million	2	0
ruble	1	1
again	0	1
drops	0	1
ways	1	0
get	1	0
rich	1	0

It's easy to estimate the probabilities of encountering a word based on where it is (in spam or not). It's done the same as before:

$$\mathsf{P}(X = \text{win}|\text{Class} = \text{spam}) = \frac{1}{7},$$

since we encountered the word win in spam emails once, and spam emails contained 7 words in total (including repetitions). Unfortunately, this estimate may produce unexpected results. For example,

$$\mathsf{P}(X = \text{win}|\text{Class} = \text{ham}) = 0,$$

it means that an email that contains the word 'win' will never be classified as ham despite that the rest of the words ('again' and 'drops') in the email indicate that it's not spam. A solution is to use Laplace smoothing. We assume that we have encountered each word once more than we did. Then,

$$\mathsf{P}(X|\text{Class} = y) = \frac{1 + \text{the number of words } X \text{ in the class } y}{|V| + \text{the number of words in the class } y}, \quad y \in \{\text{spam}, \text{ham}\}$$

where $|V|$ is the number of words in the dictionary. In such a case,

$$\mathsf{P}(X = \text{win}|\text{Class} = \text{spam}) = \frac{1+1}{8+7} = \frac{2}{15},$$

$$\mathsf{P}(X = \text{win}|\text{Class} = \text{ham}) = \frac{0+1}{8+3} = \frac{1}{11}.$$

How to classify an email 'Win a ruble to get rich'? In the same way as earlier. First, we calculate $F(\text{spam})$:

$$F(\text{spam}) = \ln \mathsf{P}(\text{spam}) + \ln \mathsf{P}(X = \text{win}|\text{Class} = \text{spam}) +$$

$$+ \ln \mathsf{P}(X = \text{ruble}|\text{Class} = \text{spam}) + \ln \mathsf{P}(X = \text{get}|\text{Class} = \text{spam}) +$$

$$+ \ln \mathsf{P}(X = \text{rich}|\text{Class} = \text{spam}) =$$

$$= \ln\frac{2}{3} + \ln\frac{2}{15} + \ln\frac{2}{15} + \ln\frac{2}{15} + \ln\frac{2}{15} \approx -8.465.$$

Next, we calculate $F(\text{ham})$:

$$F(\text{ham}) = \ln \mathsf{P}(\text{ham}) + \ln \mathsf{P}(X = \text{win}|\text{Class} = \text{ham}) +$$

$$+ \ln \mathsf{P}(X = \text{ruble}|\text{Class} = \text{ham}) + \ln \mathsf{P}(X = \text{get}|\text{Class} = \text{ham}) +$$

$$+ \ln \mathsf{P}(X = \text{rich}|\text{Class} = \text{ham}) =$$

$$= \ln\frac{1}{3} + \ln\frac{1}{11} + \ln\frac{2}{11} + \ln\frac{1}{11} + \ln\frac{1}{11} \approx -9.997.$$

Thus, the email should be classified as spam.

We are almost done. What if an email contains the word that is not in the dictionary? There are two options. We either ignore this word at all and omit it or use Laplace smoothing to retrain the classifier while it is learning:

$$\mathsf{P}(X|\text{Class} = y) = \frac{1 + \text{the number of words } X \text{ in the class } y}{|V| + r + \text{the number of words in the class } y}, \quad y \in \{\text{spam}, \text{ham}\},$$

where r is the number of words that are not in the dictionary.

For example, let's classify an email with the text 'Get rich with ruble'. We see that the word 'with' is not in our dictionary. We use Laplace smoothing given $r = 1$. Thus,

$$F(\text{spam}) = \ln \mathsf{P}(\text{spam}) + \ln \mathsf{P}(X = \text{get}|\text{Class} = \text{spam}) +$$
$$+ \ln \mathsf{P}(X = \text{rich}|\text{Class} = \text{spam}) + \ln \mathsf{P}(X = \text{with}|\text{Class} = \text{spam}) +$$
$$+ \ln \mathsf{P}(X = \text{ruble}|\text{Class} = \text{spam}) =$$
$$= \ln \frac{2}{3} + \ln \frac{2}{16} + \ln \frac{2}{16} + \ln \frac{1}{16} + \ln \frac{2}{16} \approx -9.416.$$

Next, we calculate $F(\text{ham})$:

$$F(\text{ham}) = \ln \mathsf{P}(\text{ham}) + \ln \mathsf{P}(X = \text{get}|\text{Class} = \text{ham}) +$$
$$+ \ln \mathsf{P}(X = \text{rich}|\text{Class} = \text{ham}) + \ln \mathsf{P}(X = \text{with}|\text{Class} = \text{ham}) +$$
$$+ \ln \mathsf{P}(X = \text{ruble}|\text{Class} = \text{ham}) =$$
$$= \ln \frac{1}{3} + \ln \frac{1}{12} + \ln \frac{1}{12} + \ln \frac{1}{12} + \ln \frac{2}{12} \approx -10.345.$$

Thus, the email should be classified as spam.

4 Conclusion

In this module, we began to study approaches to classification problems and discussed such metric classifiers as k-NN and weighted k-NN as well as the probabilistic Naive Bayes Classifier. You've learned how important it is to split a dataset into test and training datasets in the proper way and how to evaluate the developed model, and you also can explain why known distances are only half the work. Further, we will explore other methods and approaches to classification. Well, that's all for now. See you soon!