Slide 1

Good day and welcome to the second lecture of the machine learning course! My name is Alexei Platonov and I am glad to present you the material of today's lesson. The topic of today's lesson is the metric-based classification and regression solving problem. Today we will talk about the simplest approach to machine learning, which is based on the idea that objects having the same class or similar values ​​will be described by close vectors in terms of a feature description. We will start with a simple idea for a classifier and figure out how to improve it.

Slide 2

So, here is the plan for today's lecture. First of all, we will consider the basic idea that underlies all metric methods of classification and regression, namely the hypothesis of compactness and continuity. Having examined these ideas, we will briefly talk about how to measure the distance between objects described by feature vectors. Next, we will move on to developing our own classifier based on the compactness hypothesis. We will develop a basic idea and get a classification algorithm. But it will have a number of shortcomings and we will try to eliminate them. Along the way, we will improve our algorithm to such an extent that it will have its own parameters, methods for their selection, and many modifications that will make the algorithm almost no worse than the other classical classification algorithms that we will explore in this course. Finally, we learn how to go from regression to the idea of ​​classification by looking at the Nadaraya-Watson method for regression.

Slide 3

As I said, several simple ideas are at the heart of metric classification and regression methods. But first, let's recall the basic formulation of the machine learning problem. Let me remind you that in the supervised machine learning, we have many objects that are described by feature vectors. Each object is accompanied by a response label. If this is a classification problem, then the number of answers is fixed. If we solve the regression problem, then the number of answers is infinite, because the answer can take any number from the set of numeric values. So, how do metric algorithms solve this problem? The regression is based on the continuity hypothesis; according to it, objects that have close feature vectors have the same or close values ​​of the function being restored. Let's look at an example.

Slide 4

Here we see a simple sine function with the addition of random noise. Despite the noise, we still see that the function is continuous - it does not have sharp jumps or breaks, it does not tend to infinity. With a smooth change in the value of X, the function changes its value quite smoothly. Please note that in this case the feature vector consists of only one value, and the answer is a rational number. Thus, we simply restore the function of one variable. The continuity hypothesis tells us that the function does not change dramatically, we can try to estimate its value based on the values ​​at the nearest points.

Slide 5

But here we see a function that does not at all correspond to the continuity hypothesis. It sharply changes its value when X changes. Frankly speaking, this is just random noise, which does not depend on the parameter X. Thus, we can make the assumption that if the function somehow depends on the feature vector, then most likely it can be applied to it continuity hypothesis. At a minimum, this assumption can be used because in the real world more often you can find continuous processes than some others.

Slide 6

Now consider the compactness hypothesis. This hypothesis suggests that if we solve the classification problem, then objects of the same class having similar feature vectors will correspond to one class. In the figure on the left you can see an example of a problem that corresponds to the compactness hypothesis. Here, the blue dots correspond to class 1, and the orange dots correspond to class -1. In this problem, two sets of points of different colors are clearly visible, each of which can be grouped into one class. In the figure on the right you can see an example of a problem that does not correspond to the compactness hypothesis. In this problem, the points of different classes are strongly mixed and we cannot clearly group them. Perhaps this is also due to the fact that the feature vector is weakly dependent on the class label. When we see a situation of violation of the compactness hypothesis or the continuity hypothesis, then most likely we are faced with a situation where the feature vectors poorly describe our problem. In such a situation, it is worth trying to come up with new fetures to solve the problem. In the future, we will talk about how features are developed to solve the problem of machine learning, but we need to understand that in most cases it is impossible to come up with a universal algorithm for generating features for a task. Coming up with new features is a kind of engineering art.

Slide 7

So let's look at the classification problem. How can we use the compactness hypothesis for this task? We argue that we can use the training set of examples as a set of prototypes for classification. Imagine that we want to classify some object X. Let's look at the training set all objects closest to it. To determine the class label for a new object, we can see which labels have the nearest objects and select the answer by voting. Object X will receive a label, which is most often found in nearby objects. That's exactly what doctors do. The doctor’s training consists in collecting a large number of examples of the syndromes of some diseases. In the process, the doctor determines the patient’s syndrome, recalls other patients with the same or similar syndromes, and prescribes the treatment that was prescribed to patients in the past. We can translate this task into machine learning terms. Syndrome is a vector of symptoms, and treatment go disease is a class label. At the previous lecture, we already cited the problem of medical diagnostics as an example. In this sense, solving this problem using the compact hypothesis allows one to obtain an interpretable algorithm. This is what doctors need.

Slide 8

On this slide you can see another problem on which the compact hypothesis is clearly visible. This is the task of determining the kind of iris flower. This is a classic problem that appeared in botany long before machine learning, but even then botanists came up with a solution similar to the compactness hypothesis. In this problem, there are many features - the width and length of the sepal and the width and length of the flower petal. It turns out that if you look at a couple of features the width of the petal and the length of the sepals, then all three classes of the flower are very easily separated. Each class has its own dot color on the graph. You can easily make sure that this task is well suited to the compactness hypothesis if you carefully look at this graph.

Slide 9

Thus, we come to a method for solving the classification problem using the idea of ​​the compactness hypothesis. Let me remind you that for such a solution we must find the nearest objects in the training set. But what is near objects? We must introduce a function called a metric. This function allows you to measure the distance between the vectors of objects. It is for this reason that the group of algorithms that we are considering is called metric. Let me remind you that each object in machine learning is described by a feature vector - it is a vector of numbers. In mathematics, there are several ways to measure the distance between objects. The simplest and most known to us is the Euclidean distance. You can see the formula for the multidimensional case on the slide. Let me remind you that according to this formula, the distance between the vectors is the root of the sum of the squares of the differences in the values ​​of the coordinates of the vectors. In general, distance is a function that takes a pair of vectors and returns a number from 0 to infinity. Zero corresponds to completely identical vectors. If we have the larger value of the distance function, then we will have the farther vectors from each other in the vector space.

Slide 10

But here is the interesting thing. We may not explicitly use feature vectors. For some tasks, there are metrics that allow you to estimate distance without using feature vectors. So for example, for strings there is a Hamming distance or the Levenshtein distance. Both metrics are used to evaluate how much the characters of one line differ from the characters of another line. This is the so-called editing distance. For images, you can use SSIM algorithms or the distance between image histograms. You can easily find descriptions of the algorithms for these metrics on the Internet. The main idea that I wanted to express in this slide is that humanity has been working for a very long time on the invention of metrics to assess the similarity of objects and, perhaps, when solving your problem, you can find a ready-made metric.

So, as soon as the metric is selected, we can proceed to the solution of the classification or regression problem using the compactness or continuity hypothesis.

Slide 11

So, let's formally define a generalized metric classifier. What it is? Here is a direct implementation of the idea of ​​a metric classifier. So, at starting pont, we will choose some object “u” - this is a vector that corresponds to the object of the problem being solved. For example, it can be a vector of numbers corresponding to the physical condition of the patient - temperature, blood pressure, general well-being, and so on. The training set, the large “X”, is a set of pairs of vectors in the same format with a class label that describes the patient's disease. Let's sort the whole set of objects of the training set by the distance from the object “u” to the object of the training set. Then we get a sorted set of neighbors of the object “u”. The lower index of the object in such list corresponds to the closer object of the training set to the vector “u”.

Now apply the following classification logic. Let's look at each object of the training set in order of distance to the vector “u” and iterate over all possible classes from our problem. If the object of the training set corresponds to some class, then he votes for it with some weight, which corresponds to the index of this object. For each class, we gain the sum of the weights. The answer in such algorithm will be the class that gained the most weight. This is exactly what is written in the formula on the slide.

Slide 12

Let's look at the pseudocode that implements this idea. Please note that such an algorithm accepts as input parameters an unknown vector “U”, a training set and a weight function that participates in the vote. At the line 2, the map of responses is initialized. The key to this map is the class label, and the value is the sum of the weights for this class. In line 3, we create a list of objects in the training set, which is sorted by the distance to the vector “u”. The loop on lines 4-6 iterates over all the objects in the list and performs the vote. In line 5, we retrieve the class label for the i-th object of the list, and in line 6 this object votes with some weight for this label.

Finally, line 7 defines the answer. The class label is selected based on the highest value of the label weight. Of course, this code is not the most optimal, but it fulfills its task - to present in the form of code a solution to the classification problem using the compactness hypothesis.

But what is a weight function?

Slide 13

A weight function is a function that, depending on the distance to the classification object, returns a numerical value. The higher numerical value of weight corresponds to the more confidently class label. The simplest example is the weight function on a slide. Let me remind you that a certain condition is written in square brackets, and if it is satisfied, the function returns one, if not, then it will return a zero value. According to this function, a weight equal to one will be set only for an object with index equals to one. All other objects will receive a weight equal to zero. What does it mean? Let me remind you that we are working with a list that is ordered by distance to the classified object. An object with a serial number equal to one is simply the closest object from the training set.

Thus, we developed the simplest classification algorithm! This algorithm is called the nearest neighbor method. The advantage of this algorithm is easy in implementation and easy in interpretation. Remember the medical diagnostic example? With this decision, the doctor will receive the closest example of a patient with such diagnosis. But this algorithm is very unstable. If there is a random outlier in the training set, then we may incorrectly classify the object. The algorithm also has a low classification quality. Let's improve it.

Slide 14

If one object may not be enough for us to be sure of the classification, let's look at the k-nearest objects. The weight function on the slide creates exactly the algorithm of k-nearest neighbors. Such a weight function will return number one only for k nearest objects. All others will receive weights equal to zero. Such algorithm will be much more stable and we will get the parameter k, which will allow us to configure the algorithm during training. Usually more options are better. Since a larger number of parameters allows you to create more flexible classification or regression algorithms. But there is a problem. What if several labels give the same weight? We must somehow choose the answer. How to choose the parameter k?

Slide 15

On this slide you can see how a change in the parameter k affects the behavior of the algorithm. The green zone corresponds to class one, and the white zone corresponds to class two. The greener the area on the graph, the more the algorithm is confident in the answer. But it is not always good when the algorithm is very confident in its answer. You may notice that with k equal to one, adjacent objects of different classes can be separated by a sharp line, which means a sharp change in class. For large values ​​of k, a smoother boundary of the change of class domains is obtained - this allows a more flexible solution to the classification problem.

Slide 16

Let's move on to methods for improving the k-nearest neighbor algorithm. First, we need to somehow learn how to choose the value of the parameter k. If for some reason we don’t know in advance what value we need, we can use the Leave-One-Out method. This method also solves the optimization problem. It is needed when the algorithm has an integer hyper-parameter that affects the quality of the solution to the problem. The main idea of ​​the method is as follows. We sequentially iterate over the value of the target parameter and sequentially iterate over the objects of the training set. We seem to temporarily forget each object of the training set, and by changing the target parameter in a certain range, we estimate the number of classification errors on the forgotten object. That is, we seem to consider the forgotten object to be a new unknown object, and using the remaining training sample, we try to determine the class label for this object. Let's look at the pseudocode of this method.

Slide 17

We solve the problem of finding the value of the parameter k such that, according to the Leave-One-Out method, the error will be minimal. Thus, to search for the minimum value, the error on the training sample is initialized with the value “infinity”. The loop in line 3 iterates over the value of the parameter k from the minimum equal to one to the maximum value that the user sets. The counter in line 4 is the number of errors in the training set when using the current value of parameter k. The for loop on line 5 iterates over all the objects in the training set. So, in line 6 we remember the object of the training sample and its class, and in line 7 we create a new selection from which we delete the current object. In line 8, we apply the k-nearest-neighbor algorithm and verify the equality of response classes. If the algorithm for the current object x with index L returns the wrong answer, then the error counter in line 9 is incremented. If, after enumerating all the objects in the training set, the error counter takes a value less than min\_error, then we remember this value and consider the current k to be the optimal value of the algorithm parameter. After we sort through all possible k, we exit the algorithm by returning the optimal value of k.

Slide 18

So, we learned how to choose the optimal value of the parameter k. But is everything successful in our algorithm? Why should we equally consider objects at different distances? In the algorithms we examined, all weights, regardless of distance, were equal to one This is fundamentally wrong. Look at the picture on the right. If we choose k equal to five, then the result of the classification will be a red cross, not a green circle. And this despite the fact that the black dot, which represents the object for classification, is closer to the green circles! Let's try to improve the algorithm, taking into account the distances to the objects of the training set. Now each i-th object acquires its own weight, which depends on the serial number of the object in the sorted array. How to find that weight?

Slide 19

You can, for example, assign weight using the heuristics shown on the slide. In any case, the larger the index number, the lower the weight value. In the first case, the weight decreases linearly from the value of the serial number. In the second case, an exponential function is used. The parameter q is less than one, and raising it to the power of i, we will get smaller and smaller values.

But both heuristics do not take into account the distance to objects. We only indirectly count it through sorting the training set. How to come up with a weight function that depends on distance?

Slide 20

Let's try to explicitly consider the distance to the object. A function that considers a weight value from zero to one depending on distance is called the kernel. On the slide you can see two examples of kernels - a linear kernel and a quadratic one. Both functions take a nonzero value in the range from zero to one. In all other cases, the functions are equal to zero. Such functions are very convenient when using metrics that count distances between vectors. Such metrics never take negative values. Please note that kernels only work in the range from zero to one. We must normalize the distance between the vectors so that it is not more than one. If K-large is a kernel function, then we can transfer the distance between vectors into it divided by the maximum allowable distance. If the distance between objects exceeds the maximum, then the kernel takes the value zero.

Slide 21

If you use a constant as a function divider on the previous slide, you can get a very interesting effect. The fact is that if the distance between objects is too large and max\_distance is too small, zones of unknown classes may appear. In the picture on the left you can see many zones of purple color. These are areas in which the classifier simply cannot assign any value to the vector during classification. If we begin to increase the max\_distance parameter, the zones of confident classification will begin to increase. In the right figure, we can see how the zones of the two classes begin to mix, and the purple zones are getting smaller.

Slide 22

We can continue to increase the max\_distance parameter. We see how the classifier responds more confidently with class labels.

Slide 23

We can continue to increase max\_distance until there are no purple zones at all. With a max\_distance value of six, there is no violet color on the chart at all. But how to choose the max\_distance value in the real task? The problem is that we cannot give an answer to this question with accuracy. It depends a lot on the task itself. Moreover, if the max\_distance value is incorrectly selected, it may turn out that the classifier will not be able to respond with the class label at all. What we need to do?

Slide 24

We cannot use the Leave-One-Out algorithm to select the max\_distance value because this parameter is not integer. But we can use a different approach to normalize the distance between objects. The max\_distance parameter is somewhat similar to the k parameter in the original algorithm. Ultimately, the more max\_distance, the more neighbors the algorithm considers. Let's consider the parameter k explicitly. As a divisor in the kernel, we choose the distance to k plus the first neighbor. This approach corresponds to the Parzen-window method for the k-nearest neighbors method. Thus, we solved two problems at once - we returned the parameter k, which can be optimized using the Leave-One-Out method, and we got rid of the problem of uncertainty in the classification. Now, even if the new vector for classification is very far from the objects of the training sample, we will still assign it some kind of label, albeit with less confidence.

Slide 25

There remains the last parameter that affects the behavior of the algorithm - this is the kernel used to calculate the weight of the vector from the training set. How to choose a kernel? The slide shows the kernels we saw before. Which one is better?

Slide 26

Let's look at how the classifier works with different kernels. The image on the left uses a linear core. In the figure to the right, a quadratic kernel is used. Good vision is required to see the difference between the pictures. It turns out that the choice of the kernel practically does not affect the solution of the problem! Thus, when solving the problem using the Parzen window method, any kernel can be selected and this will not greatly affect the quality of the algorithm.

Slide 27

Now let's move on to another improvement of the original algorithm. The fact is that the original algorithm uses the entire sample for training in classification. Recall the algorithm. We need to sort all the objects in the training set, then iterate over them in sorting order and weigh them to select the class label. And so we must do for each new vector. This is a very expensive operation and the less we need to sort the objects, the better. Thus, we need to come up with a method for selecting a subset of the objects of the training sample so as not to lose much as a classification. We will begin to develop the algorithm with the concept of margin. The margin value for the vector x is a value that shows how strongly this vector is typical of its class. The margin is the difference between the weight of the original class label and the label, which has the next weight in order, excluding the original label.

Slide 28

If you select a test set that does not intersect with the training set and count the margin for each object of the test set, then you can construct graphs of margins. This slide shows the margin plot. Note that if the margin for the vector is negative, then this means that the algorithm was wrong and assigned the wrong class label. The lower the margin value, the stronger the classifier is mistaken. And vice versa. The greater the value of the margin, the more confident the classifier is in the answer.

The margins plot can be divided into several zones. The red zone on the left, which corresponds to very large negative margin values, corresponds to the so-called outliers in the sample. Such objects are most likely errors in the test sample and they are assigned the wrong label in the sample itself. Orange and yellow zones are areas in which the classifier is uncertain. Such objects are called boundary objects. Finally, the green zone is the objects in which the classifier gives confident and correct answers. Such objects are called typical objects of their class.

Slide 29

The margins plot allows us to select useful objects that need to be stored in the classifier. Such a graph leads to the idea of ​​the STOLP algorithm. This algorithm is used to select prototypes in the k-nearest neighbor algorithm. At the very beginning in the algorithm, you remove all outliers from the training set that correspond to the red zone of the margins graph. Then we select typical objects from the green area of ​​the margins chart. Now, in the loop, we must read the number of classification errors and add new objects until this number is greater than used in the algorithm parameter. To do this, we build a new classifier based on the selected objects, count the number of negative margins and select the so-called boundary selection objects. These are the objects that were with the yellow and orange zones. The STOLP algorithm terminates when the number of error classifications becomes less than the specified value

Slide 30

The STOLP algorithm allows you to select prototypes for the k-nearest neighbors algorithm. It significantly reduces the size of the stored data and speeds up the running time of the algorithm. But it has a number of disadvantages. In particular, we must somehow choose the delta parameter. The algorithm also has great computational complexity - it requires a very large number of calculations to select prototypes in the classification algorithm. Nevertheless, this simple heuristic, in combination with other methods of improving the algorithm, allows you to build a classifier with good quality and interpretability.

Slide 31

We spent a lot of time solving the classification problem, but the methods we developed are also suitable for solving the regression problem. How? Recall the continuity hypothesis. Close objects have similar values without sharp gaps. Let's try to use the idea of ​​k-nearest neighbors for regression. The difference will only consist in the fact that we will predict some continuous values ​​from the set of rational numbers. The main idea is to somehow use the information about the nearest elements in predicting the answer. How? Let's get a look.

Slide 32

When solving the regression problem, the main function of the error is the average sum of squared errors. You can see the formula for calculating this characteristic on the slide. Our task is to select an algorithm a (x) in order to minimize the value of the error function.

Slide 33

To simplify the problem, let's imagine that the algorithm returns some constant value in the interval on a subset of all kinds of vectors. At the same time, we will not forget about the weight function, which depends on the distance to the nearest objects. This weight function can be exactly the same as when solving the classification problem! Thus, we significantly simplified the task. We want to build a piecewise continuous function that weighs the objects surrounding a certain point by the distance to it and returns a constant value.

Slide 34

The error function is continuous and differentiable. Recall from the course of higher mathematics that to obtain the minimum of such a function, we need to equate the derivative to zero. In our case, we consider the derivative with respect to alpha. In the section one, this expression is demonstrated. Note that we can now easily open the brackets and determine the alpha value. You can see the sequential calculations on the slide. In the end, we get an expression that for an arbitrary vector x returns some alpha value, which will be the answer to the task. Let's try to figure out the resulting expression. In the numerator, we see that the answers on the training set are weighted by the function w. This may be one of the weighting functions that we examined earlier for classification problem. The denominator is simply the sum of the weights. It turns out that we want to calculate the answer based on the known y and normalize these labels with weights w i-ths. Please note that the weight function is considered on the entire training set. Actually, this does not mean that alpha will only be a constant. If we use a restricted kernel, as in the examples above, then alpha will change depending on the selected point in the set X.

Thus, it is very simple to convert the classification algorithm based on k-nearest neighbors into an algorithm for solving the regression problem.

Slide 35

So, let’s conclude today’s class. We studied continuity and compactness hypotheses. We studied how to implement a classier using the compactness hypothesis. We considered methods of improvements of the nearest neighbor approach. We developed the model of regression using the nearest neighbor approach.

In the next lecture we will meet with the rule-based approach to classification and regression tasks solving.