

MINISTRY OF SCIENCE AND HIGHER EDUCATION OF THE RUSSIAN FEDERATION
FEDERAL STATE AUTONOMOUS EDUCATIONAL
INSTITUTION OF HIGHER EDUCATION
NORTH CAUCASIAN FEDERAL UNIVERSITY

Methodological guidelines
for the completion of practical work
in the discipline
" Machine Learning Theory / Теория машинного обучения"
for students majoring in
02.04.01 Mathematics and Computer Science
direction (profile)
"Training Program for Electronic Information"

Introduction

The purpose of studying the discipline " Theory of Machine Learning " is to develop general professional competence (OPK-1) and professional competence (PK-5) in future bachelor's students in the field of training 02.03.01 Mathematics and Computer Science

Objectives of the discipline :

- developing students' theoretical knowledge and practical skills in the fundamentals of machine learning;
- students' mastery of machine learning tools, models, and methods;
- acquisition of skills as a data scientist and developer of mathematical models, methods and algorithms for data analysis.

A list of planned learning outcomes for the discipline, correlated with the planned learning outcomes for the educational program

Code, wording competencies	Code, indicator formulation	Planned learning outcomes for the discipline, characterizing the stages of formation of competencies, indicators
OPK-1. Able to find, formulate and solve current and significant problems in applied and computer mathematics	ID-1 _{OPK-1} Finds and formulates significant problems of applied and computer mathematics. ID-2 _{OPK-1} Knows and uses methods for solving significant problems of applied and computer mathematics.	Will be able to identify and formulate significant problems in applied and computer mathematics Able to use methods to solve significant problems in applied and computer mathematics
PC-5. Able to present and adapt mathematical knowledge, programming methods and information technology in various ways, taking into account the level of the audience	ID-1 _{PC-5} Assesses the level of preparation of listeners during their public speeches ID-2 _{PC-5} Adapts the presentation of mathematical knowledge and programming methods to the level of the students	Able to apply machine learning tools, models, and methods to solve professional problems

These guidelines for laboratory work have been developed in accordance with the work program for the discipline "Machine Learning Theory / Теория машинного обучения" for students majoring in 02.04.01 "Mathematics and Computer Science".

Practical work 1

Topic: "Introduction to Machine Learning".

The purpose of the work

Test your knowledge of the basic tools used to handle data in machine learning projects - numpy and pandas libraries, matplotlib and seaborn visualization tools .

Methodological guidelines

Successful machine learning tasks and projects require constant manipulation of data in specific formats. Specialized libraries greatly facilitate this task. In the Python ecosystem, the numpy and pandas modules are almost always used for data analysis and transformation. Therefore, mastery of these core libraries is essential for effective work with machine learning models.

Among the basic capabilities of the Python programming language , the following are most often used when working with data analysis and machine learning:

- array indexing and slicing
- generator expressions with conditions and loops
- anonymous functions

Among the tools in the numpy library , the following are the most commonly used:

- array generation functions arange , linspace , logspace
- mgrid mesh generation function
- matrix creation functions diag , zeroes , ones
- Generating random values - the random package
- array shape - shape
- reshaping an array
- ellipsis operator - ...
- complex indexing
- index masks
- element-wise operations

pandas library is particularly useful for manipulating two-dimensional data arrays. Its most commonly used capabilities in machine learning include the following:

- saving and reading files in csv , xls , xlsx formats
- analytical functions info , describe , head
- iterating over rows and columns, accessing rows and columns by numbers and indices/names
- table joins and joins
- aggregate functions and grouping
- replacement of individual values in a table, including by condition

Furthermore, we often need to visualize data using various graphs and histograms. Visualization tools are also very important and useful. An analyst must be proficient in various graph types and select the most appropriate one for a specific task. Remember that visualization is needed to reveal hidden relationships in the data. This means that visualization should be simple, clear, and understandable, with clear objectives and conclusions.

This and all subsequent work should be completed in a Python notebook editor that allows for cell-by-cell execution. This format is more suitable for prototyping and building a data analysis and machine learning project. Jupyter Notebook or Google Sheets are recommended. Colab .

Assignments for independent completion

1. Create a multiplication table using numpy arrays.
2. Create a function that takes as arguments an integer N and the first element (a real number) and the difference (a real number) and creates a numpy matrix with the diagonal containing the first N terms of the arithmetic progression.
3. Generate a 5 by 5 matrix A containing consecutive numbers from 1 to 25 using NumPy. Use slices to extract all odd elements of this matrix into a flat array.
4. Create a two-dimensional array containing ones on the border and zeros inside.
5. Create two matrices of size (5,5). One matrix contains 5s in a checkerboard pattern as in the homework problem, the other is triangular with 5s on the main diagonal and in positions above it, and all 0s below. Calculate their determinants and find their inverses.
6. Using pandas, download the apartment price prediction dataset included with this paper.
7. Display the first few lines of the file and the last few lines of the file.
8. Use pandas methods to output the main quantitative parameters of the dataset : the number of rows and columns, the data type of each field, the number of values in each column, and the measurement scale of each numerical field.
9. Remove columns containing identifiers from the table and rename all remaining column names in Russian.
10. Print the price column separately, by number and name. Print the first, tenth, and second-to-last rows of the table by number and index.
11. Select the last ten rows into a separate table. Remove the price column. Merge it with the original table using append . Fill in the missing price values with the average for the table.
12. Select the last five columns into a separate table. Delete any rows with a price below average. Join this table to the original one (choose the most appropriate join type).
13. Derive a table containing the average price and number of apartments on each floor from the original data set.
14. csv and xlsx files . Read them and ensure the data is displayed correctly.
15. Create a table containing several numeric and text fields in Excel or another spreadsheet program. Read it into the program using pandas .
16. Complete further tasks using the original version of the dataset . Graph titles, axis names, and axis values must be indicated. The assessment will be based on the number of different attributes used in plotting the graphs and their visual appeal.
17. Create a pie chart for the Rooms feature , illustrating the number of apartments as a percentage of the number of rooms. Make the sector with the largest number of apartments prominent.

18. Plot a histogram for the target variable Price . Visually estimate the price at which the largest number of apartments are sold.
19. Construct scatterplots for the features Rooms , Square , HouseFloor , and HouseYear as functions of the target variable Price in a single region (figure) . Visually assess whether any of these scatterplots have a distribution close to a linear function.
20. Construct a kernel density estimate for the target variable Price . Visually assess whether the resulting plot resembles a normal distribution. Construct a bivariate kernel density estimate for the target variable Price and the HouseFloor feature , then visually assess on which floors and at what price the majority of apartments are sold.
21. Construct a box plot of the Square feature . Visually assess whether there are outliers, and if so, at what area size can the feature value be considered an outlier.
22. Using the PairGrid graph grid, visualize the pairwise relationships between the features Rooms , Square , HouseFloor , HouseYear , and Price as follows: histograms on the diagonal, kernel density estimates below the diagonal, and scatterplots above the diagonal. Draw conclusions based on the visualization results.
23. Create a heatmap of the correlation matrix (`df.corr()`) for the features Rooms , Square , HouseFloor , HouseYear , and Price . Use it to determine which features are dependent (for such features, the correlation coefficient is close to one).

Security questions

1. What data structures are used in Numpy ? How are they different from Python lists ?
2. Numpy use to generate arrays ?
3. What methods does Numpy offer for extracting data from arrays?
4. What is code vectorization and why does it speed up programs?
5. What types of matrix operations are implemented in Numpy ?
6. What functions are used to transform the shape, size, and join arrays?
7. What are the two main data structures used in pandas ? How do they differ?
8. How to join two tables in pandas ?
9. pandas and how do they work ?
10. What are the main types of graphs used in data analysis in machine learning?
11. What's the difference between matplotlib and seaborn ? What are the advantages of each?
12. How to set plot size in matplotlib ?
13. How to install styles in seaborn ?
14. subgraphs used for? subplots ?
15. What are the main plot types supported by matplotlib ? What is depicted in a box plot?
16. How do I change the color palette for a heat map?
17. Additional tasks
18. Study the documentation of the numpy , pandas , and matplotlib modules on the topics mentioned in the guidelines.
19. Learn how to work with dates and times: the strptime module .
20. Try downloading both the local file containing the data and the file hosted on a hosting service (such as GitHub) via URL.
21. Demonstrate different types of table joins in pandas .
22. Write SQL equivalents of all the operations we performed in the lab using pandas .
23. On the laptop

24. Create a jointplot (a hybrid of scatterplot and histogram) from the seaborn library . Create a violinplot (a hybrid of boxplot and kernel density estimation) from the seaborn library .
25. Demonstrate how to work with the FacetGrid subheading grid .
26. Demonstrate how to work with 3D graphs. Create an interactive graph.

Practical work 2

Topic: "Metric Classifiers"

The purpose of the work

Get acquainted with the basic techniques for working with classification models in scikit-learn .

Tasks to complete

1. Download your diabetes diagnostic [data](#) .
2. Build a classification model to predict the presence of a disease.
3. Evaluate the quality of the built model using the classification report and classification matrix.
4. Build an alternative polynomial model and compare it with the previous one.

Methodological guidelines

diabetes dataset . This is a well-known dataset that contains information on the medical parameters of over 700 patients examined for diabetes. We'll use it to practice building classification models.

First, let's download the source dataset. This can be done either by copying the CSV file to a local folder or via a public URL:

data =

```
pd.read_csv("https://raw.githubusercontent.com/koroteevmv/ML_course/2023/ML2.2%20real%20classification/data/diabetes.csv")
```

Note that the *sklearn library* has a very similar dataset built in. `pima-indian-diabetes` . Please note that this paper uses a slightly different dataset .

As before, it's a good idea to visually familiarize yourself with the dataset before beginning your analysis. Let's display the dataset :

```
data.head()
```

Get to know the basic structure of the dataset :

index	Pregnancies	Glucose	Blood Pressure	Skin-Thickness	Insulin	BMI	DiabetesPedigree-Function	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

When conducting a serious analysis, before building a machine learning model, it's necessary to thoroughly process and clean the dataset, including removing missing values, analyzing scales, normalizing, and removing outliers and anomalies. For educational purposes, we'll limit ourselves to mandatory checks for critical data errors.

First, let's check the data for missing values:

```
data.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Pregnancies                          768 non-null    int64
1   Glucose                              768 non-null    int64
2   BloodPressure                        768 non-null    int64
3   SkinThickness                       768 non-null    int64
4   Insulin                              768 non-null    int64
5   BMI                                  768 non-null    float64
6   DiabetesPedigreeFunction             768 non-null    float64
7   Age                                  768 non-null    int64
8   Outcome                              768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB

```

We see that there are no gaps in the data. Furthermore, it's clear that all data is expressed on numerical scales. This means that this dataset doesn't require any special processing; it's already quite clean. Now we can display the basic statistics for the dataset :

data.describe ()

index	Pregnancies	Glucose	Blood Pressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.0	768.0	768.0	768.0	768.0	768.0	768.0	768.0	768.0
mean	3.8450 520833 333335	120.89 45312 5	69.105 46875	20.536 458333 333332	79.799 479166 66667	31.992 578124 999998	0.4718 763020 833332 5	33.240 885416 666664	0.3489 583333 333333
std	3.3695 780626 988694	31.972 61819 51362 2	19.355 807170 644777	15.952 217567 727637	115.24 400235 133817	7.8841 603203 75446	0.3313 285950 127749	11.760 231540 678685	0.4769 513772 427989 6
min	0.0	0.0	0.0	0.0	0.0	0.0	0.078	21.0	0.0
25%	1.0	99.0	62.0	0.0	0.0	27.3	0.2437 5	24.0	0.0
50%	3.0	117.0	72.0	23.0	30.5	32.0	0.3725	29.0	0.0

index	Pregnancies	Glucose	Blood Pressure	Skin Thickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
75%	6.0	140.25	80.0	32.0	127.25	36.6	0.62625	41.0	1.0
max	17.0	199.0	122.0	99.0	846.0	67.1	2.42	81.0	1.0

Here we see the measurement scale for each characteristic. It can be concluded that there are no obvious anomalies in the data. Let's pay special attention to the " Outcome " column—it contains the target variable. In this case, it is also expressed as a number (0—healthy, 1—sick), but this is not always the case.

Now let's identify the target variable and factors:

```
y = data.Outcome
```

```
X = data.drop ( ["Outcome"], axis=1)
```

Let's derive the form of the resulting arrays:

```
y.shape , X.shape
```

```
((442,), (442, 10))
```

The data looks completely ready for machine learning. First, let's import the required class and create an instance:

```
from sklearn.linear_model import LogisticRegression
```

```
logistic = LogisticRegression ( )
```

Let's train the created model using the data we have:

```
logistic.fit ( X, y)
```

After following this instruction, we may see a special warning:

```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
```

```
https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:
```

```
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
```

```
n_iter_i = _check_optimize_result(
LogisticRegression())
```

The meaning of this message is that the training process terminated due to the condition of reaching the maximum number of iterations, rather than the condition of stabilizing the error function. This means that the model is learning slowly and with difficulty. This may indicate that the results may not be very satisfactory.

But let's take a look at the model we obtained after this training. First, let's derive the model coefficients:

```
print ( " Coefficients : \n ", logistic.coef_ [0] )
```

```
Coefficients:
```

```
[ 1.17252338e-01 3.35998450e-02 -1.40873757e-02 -1.27047756e-03
-1.24032162e-03 7.72023360e-02 1.41904174e+00 1.00353608e-02]
```

In linear models, coefficients have a physical meaning—they indicate the significance of the corresponding features. Therefore, it is particularly interesting to examine the coefficients along with the names of the features.

To do this, we'll combine the array of column names from the dataset and the array of coefficients. You can use, for example, a generator expression to iterate through the resulting array. The `"_ = [ ... ]"` construct is only needed in the notebook to suppress automatic expression inference:

```
_ = [ print( k, v) for k, v in zip( X.columns , logistic.coef_[0])]
```

Pregnancies 0.11725233809260834

Glucose 0.03359984495281752

BloodPressure -0.014087375706652597

SkinThickness -0.0012704775628293837

Insulin -0.001240321623795555

BMI 0.07720233600112382

DiabetesPedigreeFunction 1.4190417369640222

Age 0.010035360773350831

Analyze this data yourself and draw a conclusion about which attributes have the greatest influence on the value of the target variable.

As in the linear regression model, this vector does not include a free coefficient. It is stored in a separate class field:

```
print ( " Intercept :\n ", logistic . intercept_ )
```

Intercept :

```
[-7.70291388]
```

Now we can build a forecast using the resulting model. To do this, we'll pass our feature matrix to the corresponding method:

```
y_pred = logistic.predict (X)
```

Having formed a vector of predicted values of the target variable, we can compare it with the actual values:

```
_ = [ print( a, b) for a, b in list(zip(y, y_pred ))[:10]]
```

You can see that most of the values match, but there are also errors—differences in values.

However, manually comparing all the values is very inconvenient. Therefore, it's better to use special functions called metrics. The simplest of these calculates the number of correctly and incorrectly recognized objects and presents the result as a classification matrix:

```
from sklearn import metrics
```

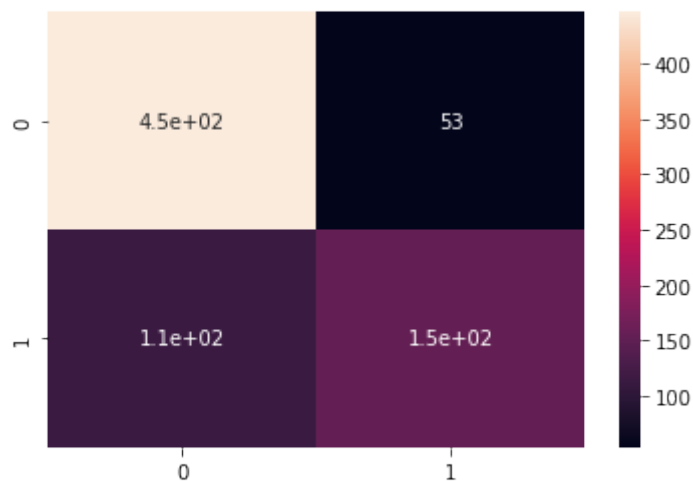
```
metrics.confusion_matrix (y, y_pred )
```

The classification matrix provides us with very useful information: the joint distribution of the number of objects in the predicted and actual classes. By examining this matrix, we can obtain important information: how many objects we classified correctly, how many incorrectly, which classes our model gravitates toward, which classes are recognized well, which poorly.

It's much more convenient to analyze the same information in graphical form. For this, we'll use a specialized library, *seaborn* , which makes it very easy to create visualizations useful for machine learning:

```
import seaborn as sns
```

```
sns.heatmap ( metrics.confusion_matrix (y, y_pred ), annot =True)
```



In addition to the classification matrix, it's very useful to use numerical metrics of classification performance. The simplest and most common of these, the prediction accuracy metric, measures the proportion of correctly recognized objects. The calculation of this metric is built into the model object itself and is accessible using a special method:

```
logistic.score(X, y)
0.7825520833333334
```

This same metric can be calculated differently, using a separate function from the *metrics package*. Note the different method signature:

```
metrics.accuracy_score(y_test, y_pred)
```

The metric value (0.78) indicates that the model makes errors on average 22% of the time. This is the primary indicator of model quality. In future work, we will demonstrate how to measure it more accurately.

If this level of performance isn't satisfactory, we can try using other classes of classification models and select the best one. For example, we could try building a polynomial model. The *sklearn library* doesn't provide a separate polynomial model class. One can create one using a special *PolynomialFeature* object, which adds polynomial features to the data. To use it, first import it:

```
from sklearn.preprocessing import PolynomialFeatures
```

Now we can create a transformation object (just like we created a model object):

```
poly = PolynomialFeatures(2)
```

Here we specify that we will be creating second-order polynomial features. Now we can use this object to create the features themselves:

```
poly = poly.fit_transform(X)
```

```
poly
```

This code prints the following data to the screen:

```
array([[1.00000e+00, 6.00000e+00, 1.48000e+02, 7.20000e+01, 3.50000e+01,
0.00000e+00, 3.36000e+01, 6.27000e-01, 5.00000e+01, 3.60000e+01,
8.88000e+02, 4.32000e+02, 2.10000e+02, 0.00000e+00, 2.01600e+02,
3.76200e+00, 3.00000e+02, 2.19040e+04, 1.06560e+04, 5.18000e+03,
0.00000e+00, 4.97280e+03, 9.27960e+01, 7.40000e+03, 5.18400e+03,
2.52000e+03, 0.00000e+00, 2.41920e+03, 4.51440e+01, 3.60000e+03,
1.22500e+03, 0.00000e+00, 1.17600e+03, 2.19450e+01, 1.75000e+03,
0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 1.12896e+03,
2.10672e+01, 1.68000e+03, 3.93129e-01, 3.13500e+01, 2.50000e+03],
[1.00000e+00, 1.00000e+00, 8.50000e+01, 6.60000e+01, 2.90000e+01,
0.00000e+00, 2.66000e+01, 3.51000e-01, 3.10000e+01, 1.00000e+00,
```

```
8.50000e+01, 6.60000e+01, 2.90000e+01, 0.00000e+00, 2.66000e+01,
3.51000e-01, 3.10000e+01, 7.22500e+03, 5.61000e+03, 2.46500e+03,
0.00000e+00, 2.26100e+03, 2.98350e+01, 2.63500e+03, 4.35600e+03,
1.91400e+03, 0.00000e+00, 1.75560e+03, 2.31660e+01, 2.04600e+03,
8.41000e+02, 0.00000e+00, 7.71400e+02, 1.01790e+01, 8.99000e+02,
0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 7.07560e+02,
9.33660e+00, 8.24600e+02, 1.23201e-01, 1.08810e+01, 9.61000e+02]]])
```

Now this data can be used as input for modeling. We'll construct a standard logistic regression:

```
polynomial = LogisticRegression ()
```

```
polynomial . fit ( poly , y )
```

```
y_pred_poly = polynomial.predict ( poly)
```

Explore the accuracy of this model and compare it to the linear one yourself.

Assignments for independent completion

1. Explore the *sklearn documentation* for the [LogisticRegression class](#). What other information can be output for a trained model? Try changing the arguments when creating the model and see how this affects the prediction quality.
2. Try applying other classification models to the same problem. For each, output the classification matrix and accuracy score. We recommend exploring the following models:
 - i. Support vector machine
 - a. Without core
 - b. With a linear kernel
 - c. With a Gaussian kernel
 - d. With a polynomial kernel
 - ii. Nearest neighbor method
 - iii. Multilayer perceptron
 - iv. Decision tree
 - v. Naive Bayes classifier
 - vi. (*) Other methods:
 - a. Passive-aggressive classifier
 - b. Comb classifier
 - c. Random Forest
 - d. Begging
 - e. Other models on request
3. Write a function that automatically trains all of the listed models and produces an accuracy score for each.
4. Repeat the entire analysis for another task - recognizing the iris species based on plant parameters (you can use the `sklearn.datasets.load_iris ()` method).

Security questions

1. What is the difference between using different classification models in the *sklearn library* ?
2. What does the regression accuracy metric show?
3. What is the significance of logistic regression coefficients?
4. What does the classification matrix show?
5. What parameters does the logistic regression object constructor have?
6. What attributes does a logistic regression object have?
7. *sklearn* library machine learning model objects have ?

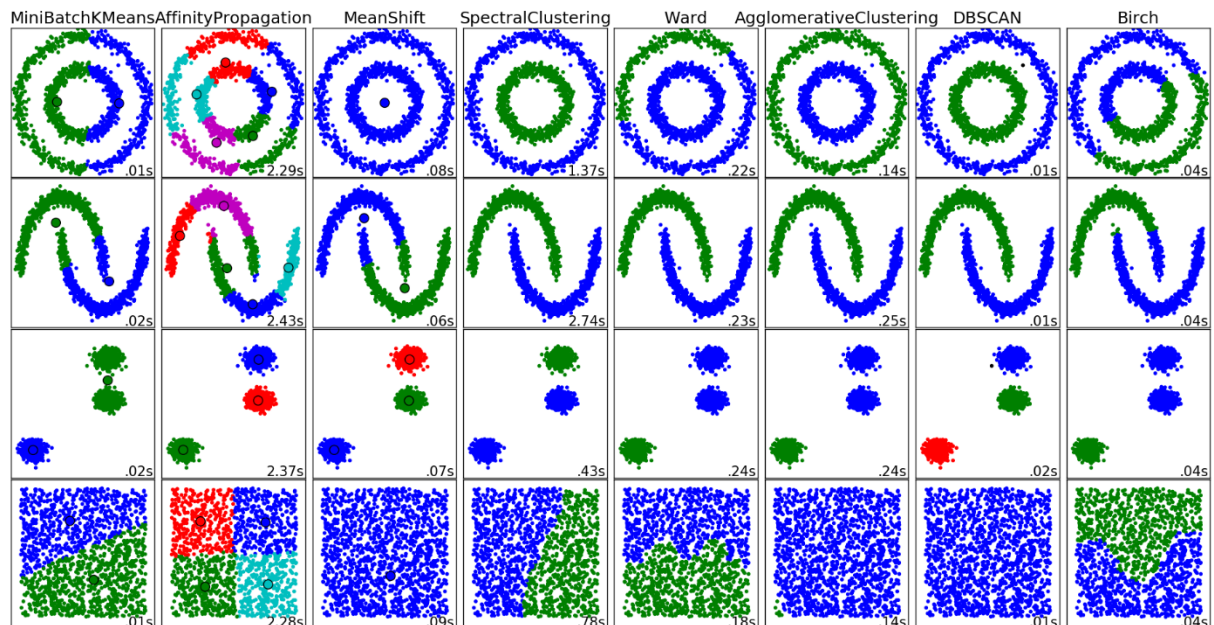
Practical work 3

Topic: Clustering Algorithms

Objective of the work: to consolidate knowledge, skills and abilities in working with clustering algorithms.

Methodological guidelines

The intuitive formulation of the clustering problem is quite simple and represents our desire to say: "Here I have a scattering of points. I see that they tend to cluster together in some sort of pile. It would be cool to be able to assign these points to these piles and, whenever a new point appears on the plane, tell which pile it falls into." This formulation clearly leaves much room for imagination, and this gives rise to a corresponding set of algorithms for solving this problem. The algorithms listed by no means describe this set completely, but they are examples of the most popular methods for solving the clustering problem.



Examples of clustering algorithms from the scikit-learn package documentation

Task 1. Implement the K- means algorithm

The K-means algorithm is probably the most popular and simple clustering algorithm and is very easy to represent in simple pseudocode:

1. Choose the number of clusters k that seems optimal to us for our data.
2. Sprinkle randomly k points (centroids) into our data space.
3. For each point in our data set, calculate which centroid it is closer to.
4. Move each centroid to the center of the sample we assigned to that centroid .
5. Repeat the last two steps a fixed number of times, or until the centroids "converge" (usually this means that their displacement relative to the previous position does not exceed some predetermined small value).

For a standard Euclidean metric for points lying on a plane, this algorithm is very easy to describe analytically and draw. Let's look at an example:

In [] :

```

# Let's start by pouring three clusters of points onto the plane
X = np . zeros ( (150, 2))

np . random . seed ( seed = 42)
X[ :50, 0] = np . random . normal ( loc = 0.0, scale = 0.3, size = 50)
X[ :50, 1] = np . random . normal ( loc = 0.0, scale = 0.3, size = 50)

X[ 50:100, 0] = np . random . normal ( loc = 2.0, scale = 0.5, size = 50)
X[ 50:100, 1] = np . random . normal ( loc =- 1.0, scale = 0.2, size = 50)

X[ 100:150, 0] = np . random . normal ( loc =- 1.0, scale = 0.2, size = 50)
X[ 100:150, 1] = np . random . normal ( loc = 2.0, scale = 0.5, size = 50)

plt . figure ( figsize = (5, 5))
plt . plot ( X[ :, 0], X[:, 1], " bo ");
In [ ] :
# Scipy has a great function that calculates distances
# between pairs of points from two arrays fed to it as input
from scipy . spatial . distance import cdist

# Let's add randomness and add three random centroids to start with
np . random . seed ( seed = 42)
centroids = np . random . normal ( loc = 0.0, scale = 1.0, size = 6)
centroids = centroids . reshape ((3, 2))

cent_history = []
cent_history . append ( centroids)

for i in range ( 3):
# We calculate the distances from observations to the centroids
distances = cdist ( X, centroids )
# We look at which centroid each point is closest to
labels = distances . argmin ( axis = 1)

# Let's put in each new centroid the geometric center of its points
centroids = centroids . copy ()
centroids[ 0, :] = np . mean (X[labels == 0, :], axis = 0)
centroids[ 1, :] = np . mean (X[labels == 1, :], axis = 0)
centroids[ 2, :] = np . mean (X[labels == 2, :], axis = 0)

cent_history . append ( centroids)
In [ ]:
# Now let's draw all this beauty
plt . figure ( figsize = (8, 8))
for i in range ( 4):

```

```
distances = cdist (X, cent_history [ i ])
labels = distances . argmin (axis = 1)
```

```
plt . subplot ( 2, 2, i + 1)
plt . plot ( X[ labels == 0, 0], X[labels == 0, 1], " bo ", label = "cluster #1")
plt . plot ( X[ labels == 1, 0], X[labels == 1, 1], "co", label = "cluster #2")
plt . plot ( X[ labels == 2, 0], X[labels == 2, 1], " mo ", label = "cluster #3")
plt . plot ( cent_history [ i ][:, 0], cent_history [ i ][:, 1], " rX ")
plt . legend ( loc = 0)
plt . title ( "Step { : }" . format( i + 1));
```

It's also worth noting that, although we considered the Euclidean distance, the algorithm will converge with any other metric as well. Therefore, for various clustering problems, depending on the data, you can experiment not only with the number of steps or the convergence criterion, but also with the metric by which we calculate the distances between points and cluster centroids . Another feature of this algorithm is that it is sensitive to the initial spatial position of the cluster centroids . In such situations, several consecutive runs of the algorithm followed by averaging the resulting clusters can help.

Task 2. Selecting the number of clusters for kMeans

Unlike classification or regression problems , in clustering it's more difficult to choose a criterion that would easily represent the clustering problem as an optimization problem. In the case of kMeans , a common criterion is the sum of the squared distances from points to the centroids of the clusters to which they belong.

$$J(C)=\sum_{k=1}^K\sum_{i \in C_k} \| x_i-\mu_k \|^2 \rightarrow \min_C ,$$

here C is the set of clusters of power K , μ_k is the centroid of cluster C_k .

Clearly, this makes sense: we want the points to be clustered near the centers of their clusters. But here's the problem: the minimum of this functionality will be achieved when there are as many clusters as points (that is, each point is a cluster of one element). To solve this problem (choosing the number of clusters), the following heuristic is often used: choose the number of clusters starting from which the described functional $J(C)$ decreases "no longer so rapidly." Or, more formally:

$$D(k)=|J(C_k)-J(C_{k+1})|/|J(C_{k-1})-J(C_k)| \rightarrow \min_k$$

Let's consider example .

In []:

```
from sklearn.cluster import KMeans
```

In []:

```
inertia = []
```

```
for k in range (1, 8):
```

```
    kmeans = KMeans ( n_clusters = k, random_state = 1) . fit(X)
```

```
    inertia . append ( np . sqrt ( kmeans . inertia_ ))
```

In []:

```
plt . plot ( range(1, 8), inertia, marker = "s")
```

```
plt . xlabel ( "$k$")
```

```
plt . ylabel ( "$J( C_k )$");
```

We see that $J(C_k)$ drops sharply when the number of clusters increases from 1 to 2 and from 2

to 3, and not so sharply when k changes from 3 to 4. This means that in this problem it is optimal to set 3 clusters.

Difficulties

means problem itself is NP-hard (see the article "[A Little More About P and NP](#)" on Habr), and for dimension d , the number of clusters k , and the number of points n , it can be solved in $O(ndk+1)$. Heuristics are often used to solve this kind of problem, for example MiniBatch K-means , which uses small portions of the dataset (called a batch) for training rather than the entire dataset, updates centroids using the average of all centroid updates over the entire history of all points related to it. A comparison of regular K-means and its MiniBatch implementation can be found in [the scikit-learn documentation](#) .

[implementation](#) in scikit-learn offers a host of convenient features, such as the ability to specify the number of runs via the `n_init` parameter , which will yield more stable cluster centroids in the case of skewed data. Furthermore, these runs can be run in parallel, without sacrificing computation time.

Task 3. Implement the AffinityPropagation clustering algorithm

Another example of a clustering algorithm. Unlike the K-means algorithm, this approach doesn't require us to predetermine the number of clusters into which we want to partition our data. The basic idea is that we want our observations to be clustered into groups based on how they "communicate," or how similar they are to each other.

For this purpose, we introduce some kind of "similarity" metric, defined by the fact that $s(x_i, x_j) > s(x_i, x_k)$ if observation x_i is more similar to observation x_j than to x_k . A simple example of such similarity would be the negative square of the distance $s(x_i, x_j) = -\|x_i - x_j\|^2$.

Now let's describe the "communication" process itself. To do this, we'll create two matrices initialized to zero. One of these, $r_{i,k}$, will describe how well the k th observation is suited to being a "role model" for the i th observation, relative to all other potential "role models," while the second, $a_{i,k}$, will describe how appropriate it would be for the i th observation to choose the k th as such a "role model." It sounds a bit confusing, but we'll see a simple example a bit later.

After this, the matrix data is updated in turn according to the rules:

$$\begin{aligned} r_{i,k} &\leftarrow s(x_i, x_k) - \max_{k' \neq k} \{ a_{i,k'} + s(x_i, x_{k'}) \} \\ a_{i,k} &\leftarrow \min(0, r_{i,k} + \sum_{i' \notin \{i,k\}} \max(0, r_{i',k})), i \neq k \\ a_{k,k} &\leftarrow \sum_{i' \neq k} \max(0, r_{i',k}) \end{aligned}$$

Task 4. Implement spectral clustering

Spectral clustering combines several of the approaches described above to obtain the maximum amount of profit from complex manifolds of smaller dimension than the original space.

For this algorithm to work, we need to define an adjacency matrix . This can be done in the same way as for AffinityPropagation above: $A_{i,j} = -\|x_i - x_j\|^2$. This matrix also describes a complete graph with vertices in our observations and edges between each pair of observations, with a weight corresponding to the similarity of these vertices. For our chosen metric and points lying on a plane, this will be intuitive and simple: two points are more similar if the edge between them is shorter. Now we would like to split our resulting graph into two halves such that the resulting points in the two graphs are, in general, more similar to other points within the resulting "own" half of the graph than to points in the "other" half. The formal name for such a problem is the Normalizedcutsproblem , and you can read more about it [here](#) .

Practical work 4

Topic: Decision trees, linear classifiers

Objective of the work: To learn how to apply the decision tree model to classification and regression problems.

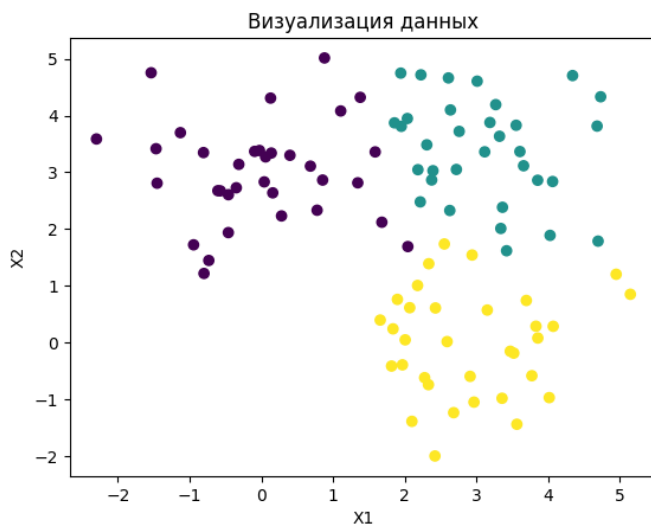
Methodological guidelines

Classification problem

Let's create and visualize the dataset :

```
X, y = make_blobs ( n_samples = 100, centers=[(0,3),(3,3),(3,0)],  
n_features =2, random_state =RANDOM_SEED,  
cluster_std=( 0.9,0.9,0.9))  
plt.scatter ( X[ :, 0], X[:, 1], c=y)  
plt.title ( ' Data Visualization ' )  
plt.xlabel ( 'X1')  
plt.ylabel ('X2')
```

We see a data set of three classes:



Just like perceptrons , decision trees can be easily applied to problems of varying dimensions.

The number of classes doesn't matter for the tree. Let's create and train the model:

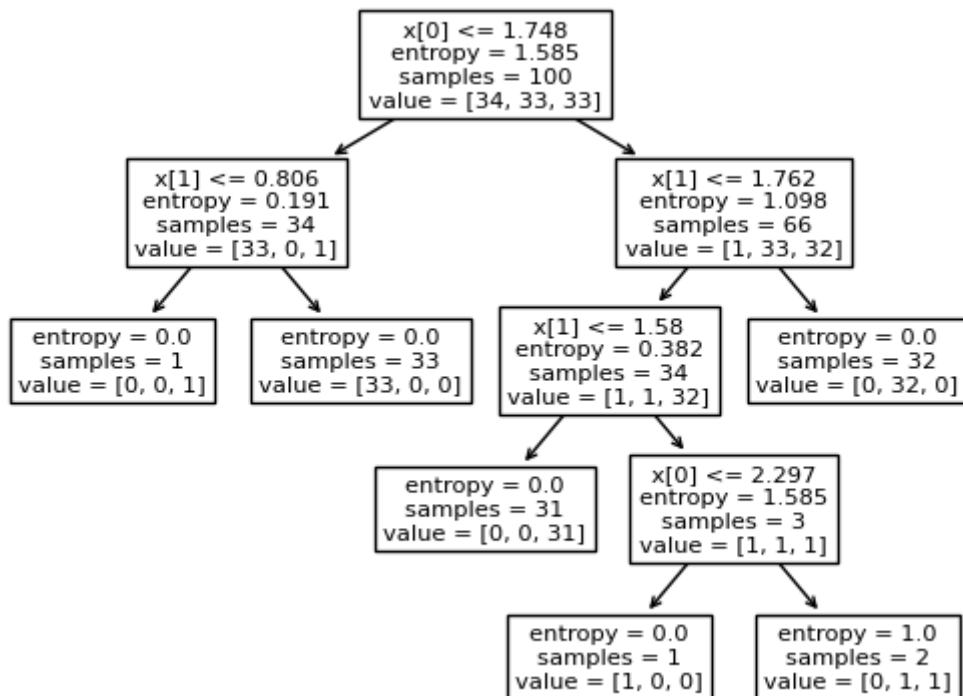
```
depth= 4  
clf_tree = DecisionTreeClassifier ( criterion='entropy', max_depth =depth,  
random_state = RANDOM_SEED)  
clf_tree.fit (X, y)
```

Here we specify the type of function that will be used to find the optimal sample split boundary. By default, the Gini criterion is used, but we will now use the information entropy criterion. We also specify the maximum tree depth. The tree will not "grow" beyond this number of levels. You can change these parameters yourself and analyze how this affects the model's performance

After training this model, we can plot the decision tree itself, that is, the model's complete internal structure. For this purpose, the object has a special method , `plot_tree ( )` :
`tree.plot_tree ( clf_tree )`

plt.show ()

This object has another method for visualizing the tree— as a text object. Find and apply it yourself. The method we used displays the tree as a graphic:



Interpret the information shown in this graph.

Now we can visualize the decision-making boundaries themselves:

```
X0 = np.linspace ( X[ :, 0].min()-1,X[:, 0].max()+1, X.shape [0])
```

```
X1 = np.linspace ( X[ :, 1].min()-1,X[:, 1].max()+1, X.shape [0])
```

```
X0_grid, X1_grid = np.meshgrid (X0, X1)
```

```
y_predict = clf_tree.predict( np.c_[X0_grid.ravel(),X1_grid.ravel()]).reshape(X0_grid.shape)
```

```
plt.pcolormesh ( X0_grid, X1_grid, y_predict )
```

```
plt.scatter ( X[ :, 0], X[:, 1], c=y, edgecolors = 'black', linewidth =1)
```

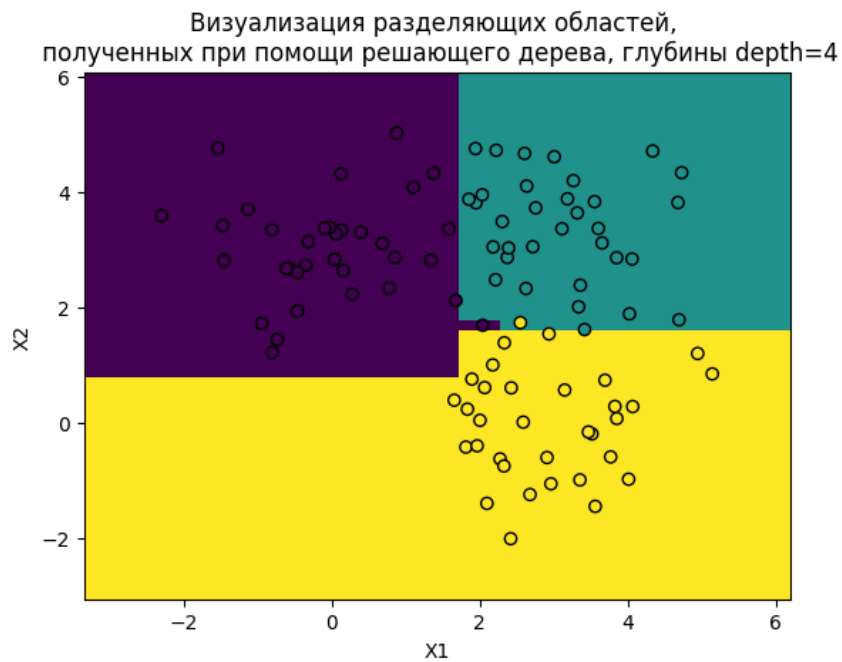
```
plt.title ( 'Visualization of separating regions obtained using a decision tree, depth = { } '. format (
depth ))
```

```
plt.xlabel ( 'X1')
```

```
plt.ylabel ( 'X2')
```

```
plt.show ()
```

We see a very characteristic tree boundary shape:



As always, we will evaluate the quality of the model's performance using metrics:

```
y_pred = clf_tree.predict(X)
print( confusion_matrix (y, y_pred ))
print( 'Accuracy =', accuracy_score (y, y_pred ))
print( 'F1_score =', f1_score(y, y_pred , average='micro'))
[[34 0 0]
 [ 0 33 0]
 [ 0 1 32]]
Accuracy = 0.99
F1_score = 0.99
```

We see that the model made only one error out of 100 examples, which is quite good. However, by increasing the depth of the tree, we can further improve its performance and achieve absolute accuracy.

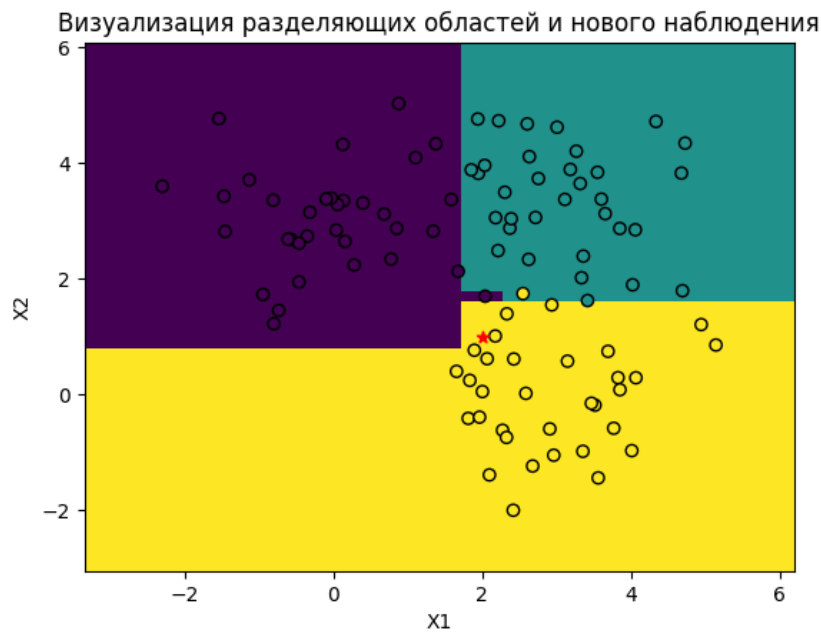
Now let's create a new observation:

```
observation_new = [[2, 1]]
```

I classify it :

```
clf_tree.predict ( observation_new )
```

Let's visualize it on a graph together with the decision boundary:



Regression problem

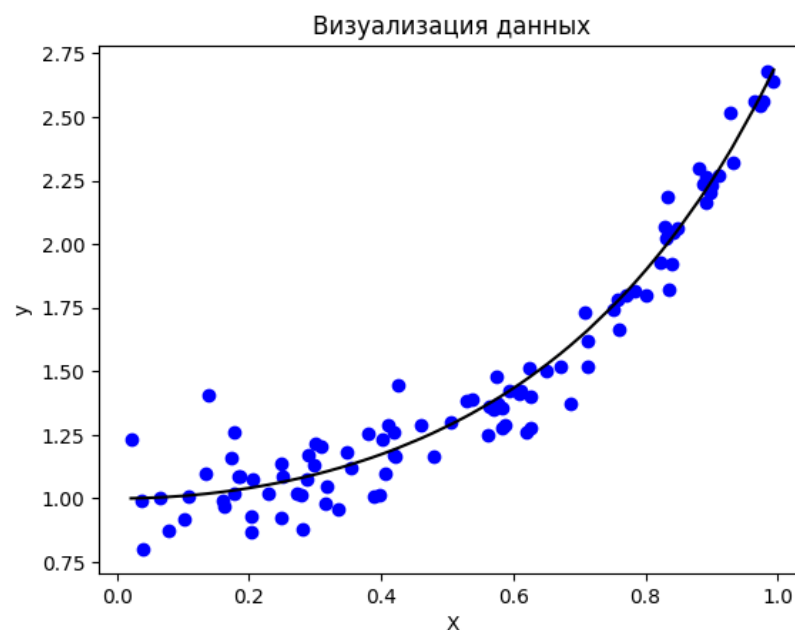
Let's generate a simple dataset for the regression problem. To do this, we'll obtain 100 ordered random numbers, then calculate the target variable values as the output of some function (for example , an exponential function) and add random noise to the result to simulate dispersion:

`n_samples = 100`

`X = np.sort ( np.random.rand ( n_samples ) )`

`y = np.exp (X**2 )+ np.random.normal (0.0, 0.1, X.shape [0])`

a dataset like this for paired regression (the line here is added only for information and clarity, it is not part of the data):



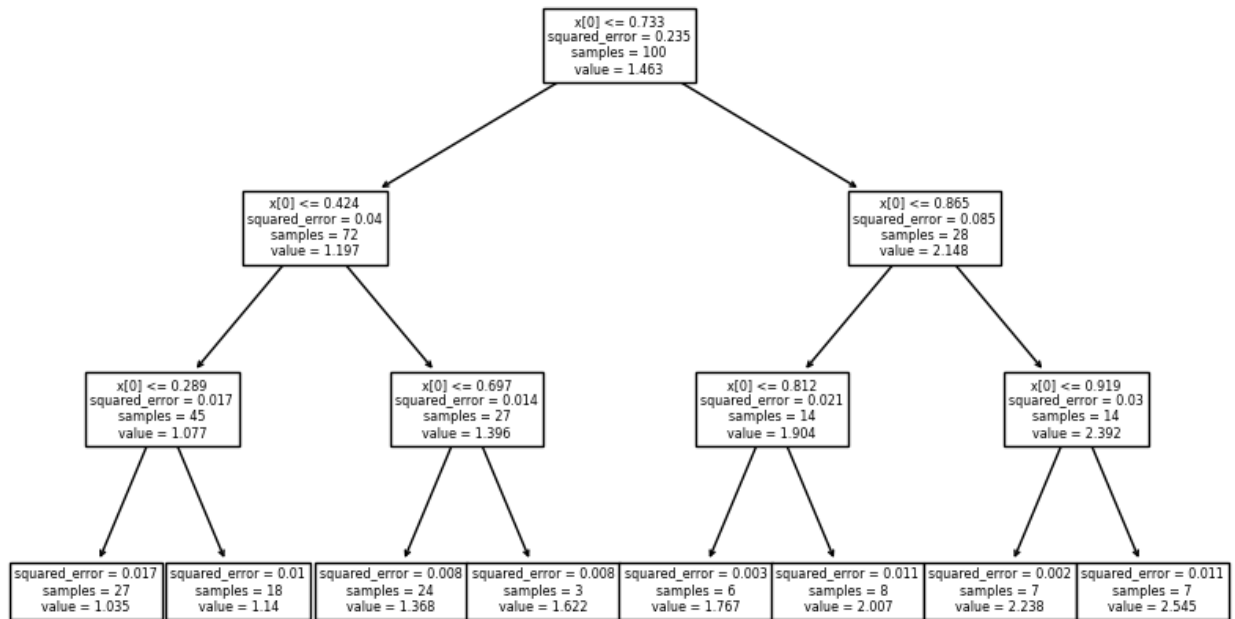
Let's create a regressor object based on decision trees and train it:

`depth= 3`

`reg_tree = DecisionTreeRegressor ( max_depth =depth, random_state =RANDOM_SEED)`

`reg_tree.fit (X, y)`

Let's output the resulting tree:

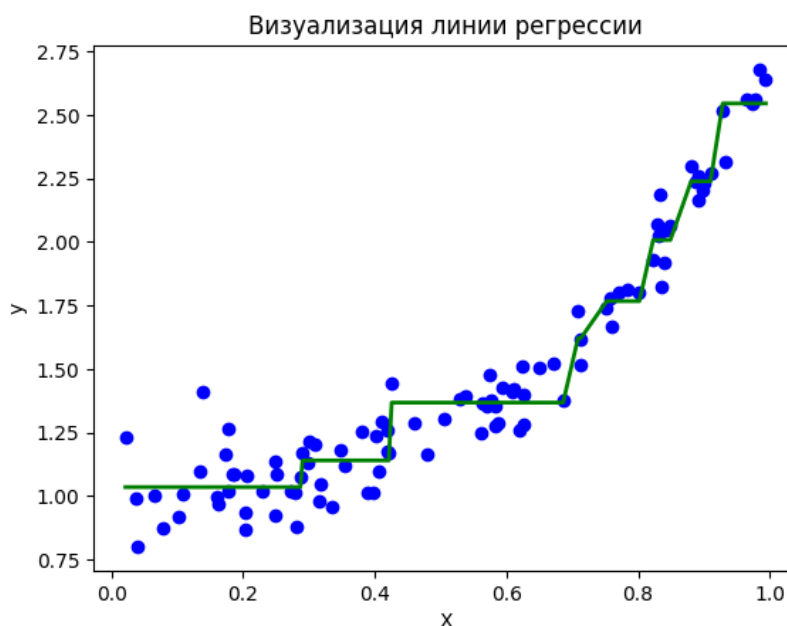


Interpret the information in this column yourself. We'll plot the regression line on the graph:
`plt.scatter(X, y, c="b")`

`plt.plot(X, y_pred_reg, "g", lw = 2)`

`plt.title('Visualization of the regression line')`
`plt.xlabel('X')`
`plt.ylabel('y')`

Again we see the characteristic piecewise linear line:



Now, as always, let's evaluate the quality of the model's work:

`print('r2= ', r2_score(y, y_pred_reg))`
`print('MSE= ', mean_squared_error(y, y_pred_reg))`

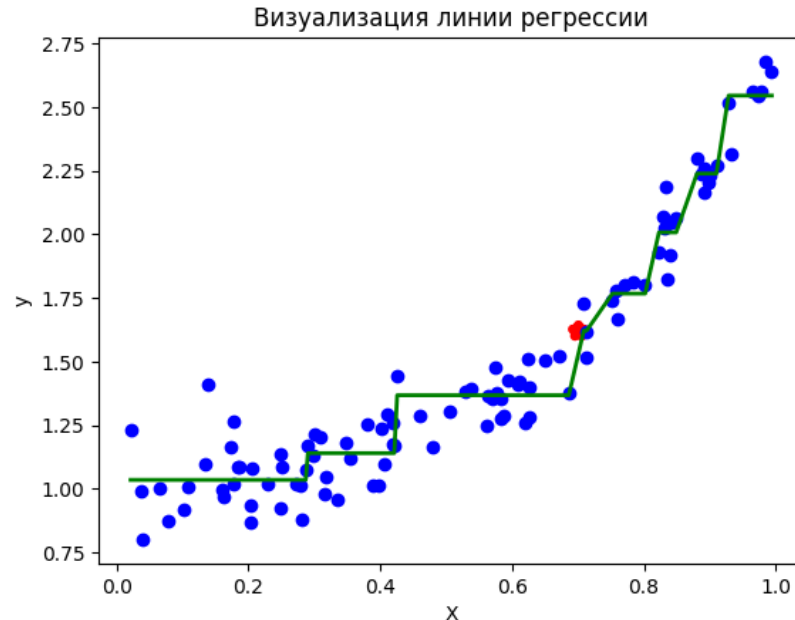
$r^2 = 0.9559340585697235$

$MSE = 0.010347452773751169$

These metrics indicate the model's fairly accurate performance. Only the graph reveals a certain arbitrariness and softness in the resulting predictions. Let's create a new prediction:

`observation_new = [[0.7]]`

And let's depict it on a graph:



Move this point manually. Note that there are areas where a slight change in the point's position does not change the prediction result.

Assignments for independent completion

1. Load the built-in `sklearn.datasets.load_iris` data, taking only the last two columns (petal length and width). Plot them in a scatterplot, color-coding each class.
2. a `sklearn.tree.DecisionTreeClassifier` decision tree classification model of depth 4 using entropy. Visualize the corresponding decision tree graph.
3. `sklearn.tree.DecisionTreeClassifier` decision tree classification model with different depths (1, 2, 3, 4, 10) using entropy and visualize the resulting splitting regions in each case.
4. Derive the necessary metrics to evaluate the performance of models with different depths. Determine which model classifies the data better.
5. Download the entire dataset `load_iris`. Train a `sklearn.tree.DecisionTreeClassifier` decision tree classification model of depth 4 using entropy. Visualize the corresponding decision tree graph. Evaluate the model's performance.
6. Load the built-in `sklearn.datasets.california_housing` data, using only the `AveBedrms` column as the only feature. Plot the data in a scatterplot with the feature values on one axis and the target variable on the other.
7. Train the decision tree regression model `sklearn.tree.DecisionTreeRegressor` with `random_state=0` and the rest of the hyperparameters as default.
8. Visualize the corresponding decision tree graph and the resulting piecewise regression line.
9. Evaluate the model's performance. Create a new observation and make a prediction based on it.
10. Download all `fetch_california_housing` dataset. Train the same model. Visualize the corre-

sponding decision tree graph and evaluate the model's performance.

Security questions

1. Why do decision boundaries in decision trees have such a characteristic shape?
2. How does tree depth affect model complexity?
3. Why can the depth of a tree vary on different branches?
4. What is a criterion in decision trees and how does it affect the performance of the model?

Additional tasks

1. Test the model's performance on unnormalized data with widely varying values. Determine the applicability of the decision tree model without normalization.
2. Apply decision tree to a large dataset (at least 10,000 rows and 10 columns) of your choice.
3. Repeat the modeling with a different criterion. Find datasets on which different criteria perform better.
4. Use a random forest instead of decision trees on the first dataset . Visualize its decision boundary.

Lab 5

Topic: Neural Networks

Objective: To become familiar with the perceptron as a supervised learning model in the sklearn library .

Methodological guidelines

To train the perceptron, let's create a dataset. We'll use familiar data generation functions. Immediately after creation, we'll visualize this dataset in a scatterplot:

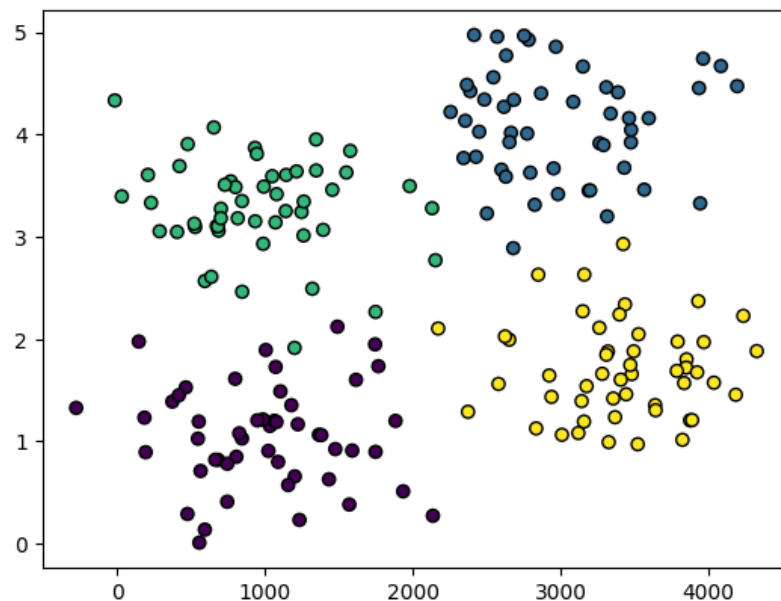
```
blob_centers = ([1, 1], [3, 4], [1, 3.3], [3.5, 1.8])
```

```
X, y = make_blobs ( n_samples = 200,  
                    centers= blob_centers ,  
                    cluster_std = 0.5,  
                    random_state = 0)
```

```
X[:, 0] *= 1000
```

```
plt.scatter ( X[:, 0], X[:, 1], c=y, edgecolors='black', linewidth=1)
```

You should see a dataset consisting of four clusters belonging to four classes:



So, we're dealing with a multiple classification problem. This isn't a problem for a perceptron ; a neural network is a very versatile model, capable of working natively and in multiple-class mode. Let's create the first version of our neural network :

```
clf = MLPClassifier ( hidden _ layer _ sizes =(6,),  
                    random_state = 1).fit(X, y)
```

Note that when creating a model, we specify the number of layers and the number of neurons in each layer using the *hidden_layer_sizes tuple* . In this case, we have one hidden layer with 6 neurons. If we don't specify this argument, its default value is (100,), meaning one layer with 100 neurons. This is quite large for a first model.

Since we've already trained the model, we can view the neuron weight values, or the network's internal parameters . They are all stored in the `coefs_` field , *similar* to other machine learning models in `sklearn` . For example, you can display the weights separately for each layer:

```
print ( "Weights between input and hidden layers:")
print ( clf.coefs_[0])
print ( "\nWeights between hidden and output layer:")
print ( clf.coefs_[1])
Weights between input and hidden layer:
[[-0.13186096 0.36276684 -0.87753229 -0.3544923 -0.62342741 -0.69458326]
 [-0.55527141 -0.28633324 -0.1793465 0.07807555 -0.14370807 0.32689274]]
```

Weights between hidden and output layers:

```
[[-0.4955391 -0.41463226 0.41301641 0.65040935 -0.25197168 0.26165608]
 [ 0.5134373 0.53925562 -0.57325313 -0.6379063 -0.4542433 0.51596208]
 [-0.55456344 -0.10522807 0.63360879 0.03543427 0.28300565 -0.25076137]
 [ 0.25356345 0.4604906 -0.66711811 0.3423877 0.70293671 0.33948522]
 [-0.29956981 0.3969661 -0.54770982 -0.0851425 0.58948128 -0.28122352]
 [-0.28935468 -0.51007606 -0.6656005 0.24156416 -0.39625967 -0.3203444 ]]
```

Or for each neuron separately:

```
for i in range( len ( clf.coefs_ )):
    number_neurons_in_layer = clf.coefs_[ i ].shape[1]
    for j in range( number_neurons_in_layer ):
        weights = clf.coefs_[ i ][:,j]
    print( i , j, weights, end=", ")
    print ( )
    print ( )
0 0 [-0.09809877 -0.58903611],
0 1 [ 0.33667051 -0.31219238],
0 2 [-0.82024119 -0.22424296],
0 3 [-0.29720839 0.11423652],
0 4 [-0.56852789 -0.18324369],
0 5 [-0.66056109 0.37321781],

1 0 [-0.51316658 -0.24370005 -0.59713359 -0.5769702 0.34285864 -0.7006119 ],
1 1 [-0.43365467 0.34328608 -0.67678056 -0.10221922 -0.25645017 0.35515533],
1 2 [ 0.43189995 0.57835494 -0.46609801 0.75476969 0.33417991 0.80295119],
1 3 [0.68148414 0.57889271 0.54011704 0.00610289 0.47281254 0.33880635],
```

Note that, judging by the output, we have four neurons in the output layer. This is because there are four classes in the dataset . The number of classes always determines the number of neurons in the output layer. And we have two neurons in the input layer. This is because there are only two attributes in the dataset .

bias neurons , as in other models, are stored in a separate field:

```
print ( "Bias weights for the hidden layer:")
print ( clf.intercepts_[0])
print ( "\nBias weights for the output layer:")
```

```
print ( clf.intercepts _[1])
Bias weights for the hidden layer:
[-0.55752645 0.60978582 -0.8640854 0.33258994 -0.18652436 0.05696655]
```

```
Bias weights for the output layer:
[-0.29413473 0.49414359 -0.60792984 -0.11888525]
```

```
Let's analyze how well our model performs. For this, we'll use the familiar classification matrix:
confusion_matrix (y, clf.predict (X))
array ( [[ 1, 0, 49, 0],
[ 0, 0, 50, 0],
[ 1, 0, 49, 0],
[ 0, 0, 50, 0]])
```

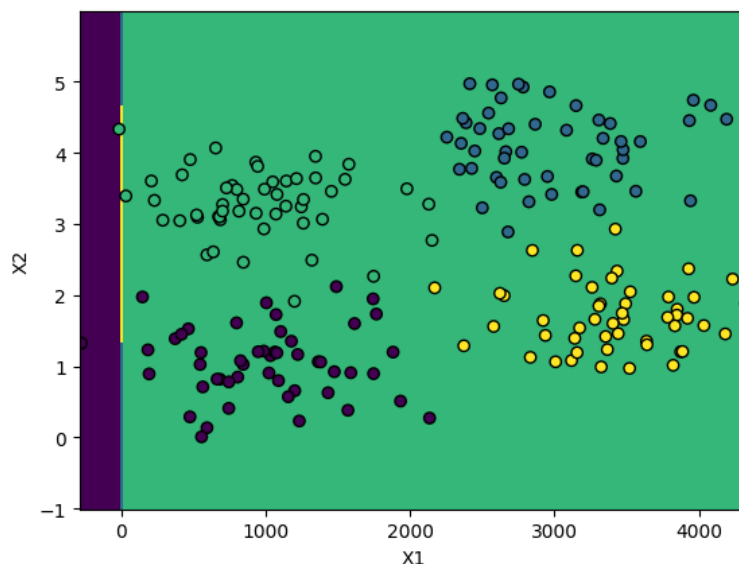
We see that the model seems to be performing poorly and is making many errors. To confirm this, let's plot the classification on a graph:

```
X0 = np . linspace ( X [ :, 0]. min ()-1, X [ :, 0]. max ()+1, X . shape [0])
X1 = np.linspace ( X[ :, 1].min()-1,X[:, 1].max()+1, X.shape [0])
X0_grid, X1_grid = np.meshgrid (X0, X1)
```

```
y_predict = clf.predict( np.c_[X0_grid.ravel(),X1_grid.ravel()]).reshape(X0_grid.shape)
plt.pcolormesh ( X0_grid, X1_grid, y_predict )
```

```
plt.scatter ( X[ :, 0], X[:, 1], c=y, edgecolors ='black', linewidth =1)
plt.xlabel ( 'X1')
plt.ylabel ( 'X2')
plt.show ()
```

Using this code as an example, learn how to represent a decision boundary for multiple classification. We see a rather strange picture:



The model assigns almost all points to a single class. And the decision boundary is almost always vertical. What did we do wrong? To answer this question, we need to look at the coordinate axes. The problem is that the attributes in the source data have very different values. X1 is measured in thousands, while X2 is measured in units. This can be a problem for some machine learn-

ing models, and the perceptron is one of them. For a perceptron to function correctly , the data must be normalized.

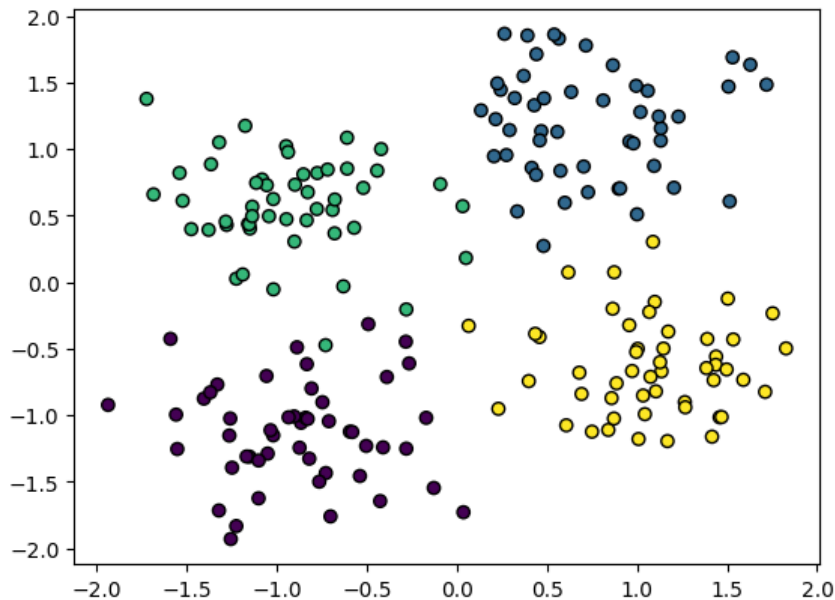
To do this, we'll import and use a data normalization object. For example , like this :

```
from sklearn.preprocessing import StandardScaler
```

```
X_scaled = scaler.transform (X)
```

```
X_scaled = scaler.transform (X)
```

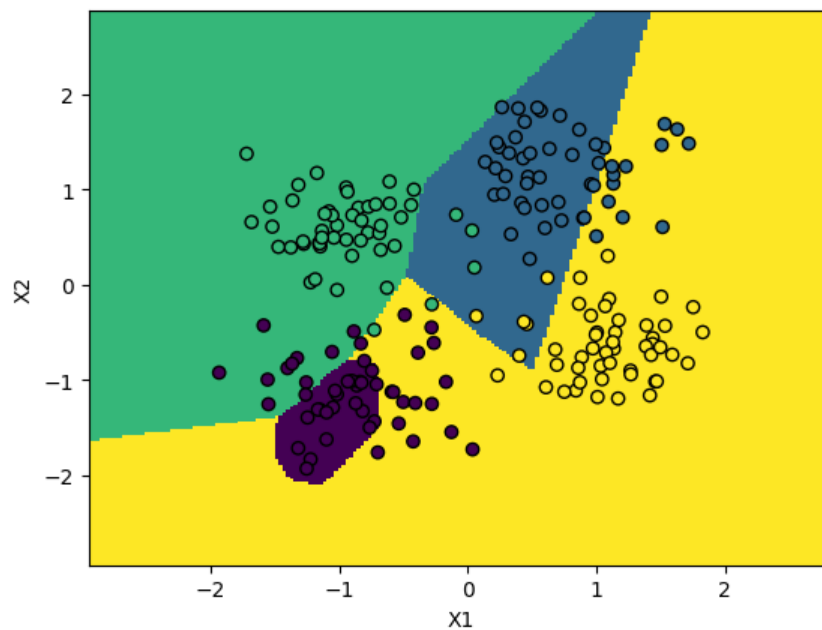
Let's plot the points again to see how normalization transformed our dataset :



The distribution itself doesn't seem to have changed, but pay attention to the values along the axes. Our dataset now has the same dimensionality. The data appears to have been "shrunk" horizontally. Let's see if this affects the quality of training. Let's retrain the model and output the classification matrix:

```
array ( [[22, 0, 11, 17],  
[ 0, 38, 0, 12],  
[ 0, 3, 45, 2],  
[ 0, 5, 0, 45]])
```

We see a completely different picture. It's gotten significantly better. For greater clarity, let's plot the classification on a graph:



We see that the regions of the corresponding classes are now much closer to the training set points. However, the model still makes quite a few errors. How can we simply improve it? You may have noticed that during training, the model issued a warning about early termination. We've encountered this situation before. We can simply increase the iteration limit of the training algorithm:

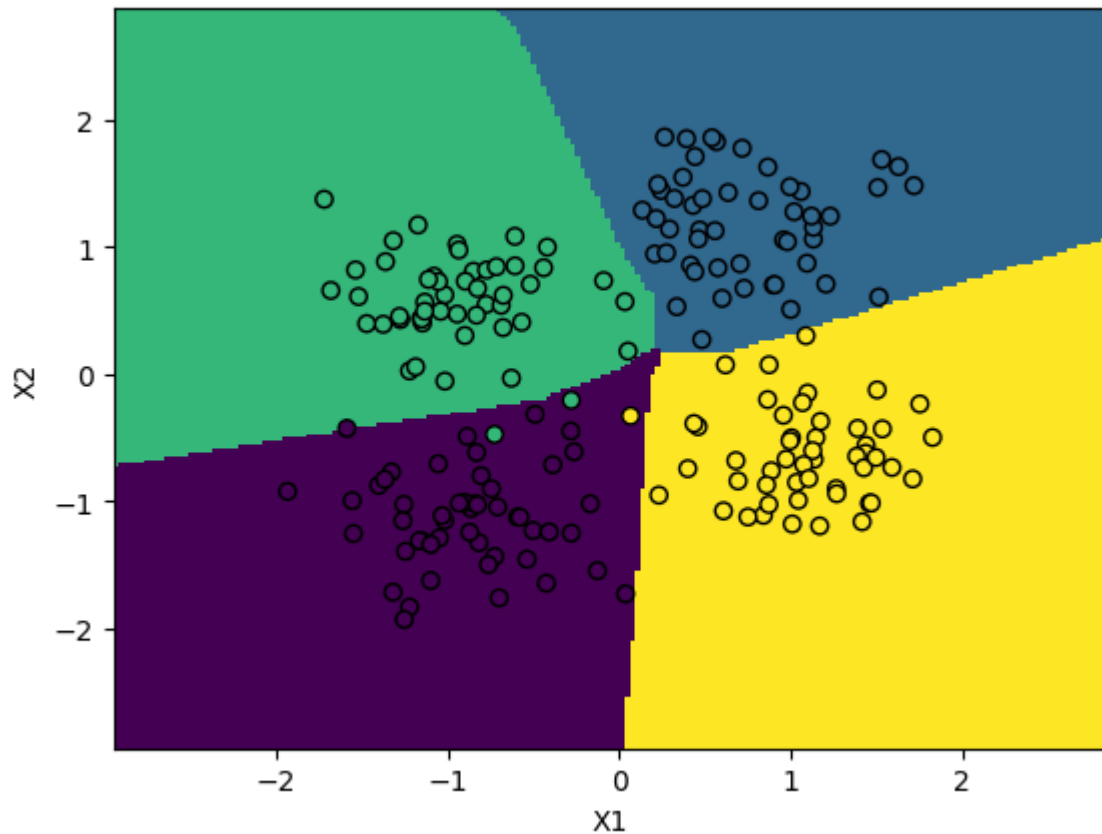
```
clf = MLPClassifier ( hidden_layer_sizes =(6,), max_iter =10_000, verbose=True).fit( X_scaled ,
y)
```

We'll also use another argument of the perceptron object constructor here : *verbose* . It displays detailed information about the model's training. Analyze the resulting information and draw a conclusion.

Now we can output the classification matrix. It's immediately clear that things have improved significantly again:

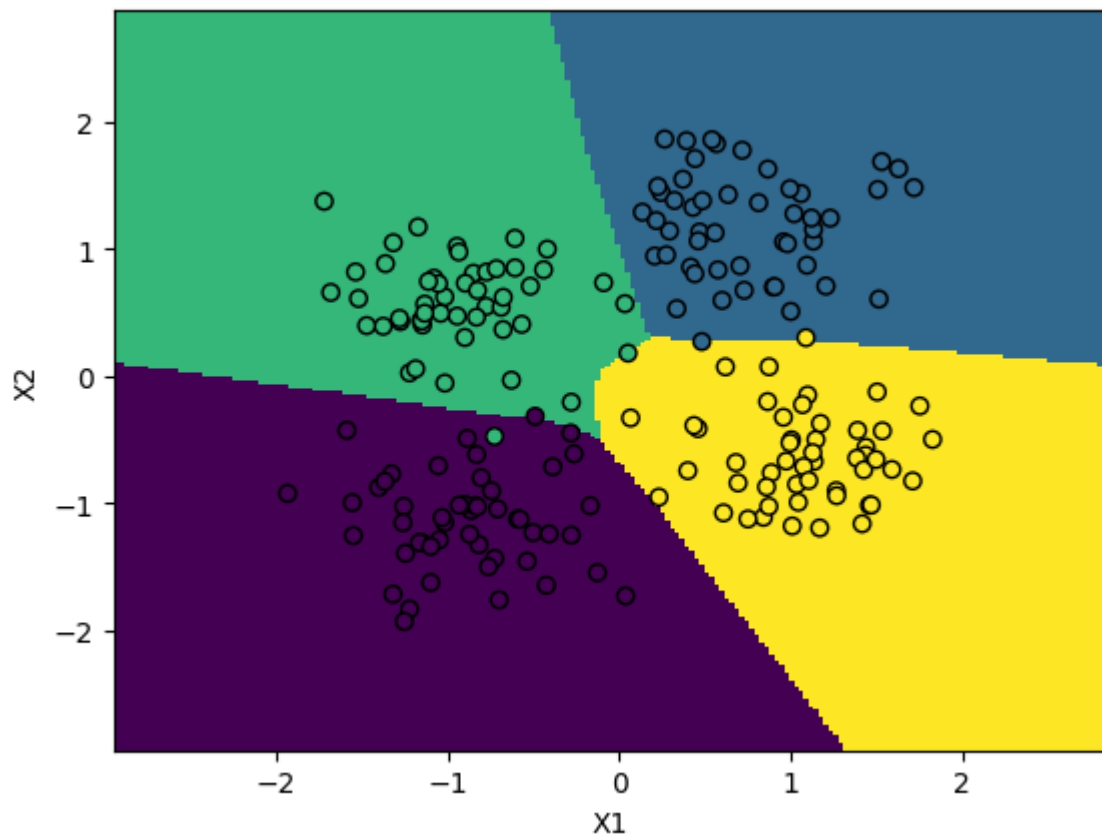
```
array ( [[50, 0, 0, 0],
[ 0, 50, 0, 0],
[ 2, 3, 45, 0],
[ 0, 1, 0, 49]])
```

This conclusion is confirmed by the visualization of the model:



To further improve accuracy, you can increase the number of hidden layers. This will create a deep neural network . For example, let's train a perceptron with three layers:

```
clf = MLPClassifier ( hidden_layer_sizes =(6, 6, 6), max_iter =10_000).fit( X_scaled , y)
```



The graph shows that the decision boundary matches the training set even better. Construct a

classification matrix for this model yourself.

Assignments for independent completion

1. Create a single-layer perceptron with 1, 2, 10, and 100 neurons. Compare their accuracy and determine whether the number of neurons is sufficient.
2. Create and evaluate a model with two, three, and ten hidden layers, each with the same number of neurons. Compare their accuracy and determine whether the number of layers is sufficient.
3. For a deep model, output the weights of all neurons in all layers. Output the values of the bias weight vectors.
4. Build and evaluate a model with a larger number of neurons and layers. Measure training time and compare it with training times for simpler models.
5. Build and evaluate a perceptron classification model on the dataset you used in the classification quiz (if you didn't take it, use any dataset from the "realworlddatasets" section of the sklearn library).
6. Build and evaluate a regression model using a perceptron on the dataset you used in the regression quiz.

Security questions

1. What is called a deep neural network?
2. What is neural network architecture?
3. How does the number of neurons and layers affect the quality of modeling?
4. How does a neural network solve multiple classification problems?
5. When should a perceptron be used ?

Additional tasks

1. Repeat the simulation on one of the datasets, but using TensorFlow, PyTorch, or Keras. Compare the results with the perceptron implementation in the sklearn library.
2. Experiment with different activation functions and optimization methods. Find datasets on which different activation functions and optimization methods perform better.

Practical work 6

Topic: Regression Analysis

The purpose of the work

Get acquainted with the basic techniques for working with regression models in scikit-learn .

Tasks to complete

1. Download the built-in California real estate price dataset .
2. Build a regression model to predict the price of a particular property.
3. Evaluate the quality of the constructed model using visualization and the coefficient of determination.
4. Build an alternative polynomial model and compare it with the previous one.

Methodological guidelines

a dataset of California real estate prices as our input data . This is a well-known training dataset. It's built into the *sklearn library* , so there's no need to upload or download it separately. To get started, let's import the required standard libraries:

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import pandas as pd
```

First, let's load the original dataset:

```
from sklearn.datasets import fetch_california_housing
```

```
california = fetch_california_housing ( )
```

First, we need to understand the structure of the data we received. To do this, let's display the type of the received object:

```
type ( california )
```

```
sklearn.utils.Bunch
```

This is a special data type in the *sklearn library* , similar in structure to a regular dictionary. So, let's look at the keys in this dictionary:

```
california . keys ()
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'feature_names', 'DESCR'])
```

Of particular interest here are *the data* and *target fields* , which contain the source attributes and the vector of values of the target variable. Let's derive their type :

```
print( type( california.data ), type( california.target ) )
```

```
< class ' numpy.ndarray '><class ' numpy.ndarray '>
```

Since these are *numpy arrays* , you can look at their shape:

```
print( california.data.shape , california.target.shape )
```

```
(20640, 8) (20640,)
```

This means the data contains over 20,000 rows and 8 attributes. You can also display the dataset description for additional information.

Now we can work with the data in a variety of ways. For ease of analysis, we'll combine all the arrays into a dataframe :

```
data = pd.DataFrame ( california.data , columns = california.feature_names )
```

```
data[ 'Price' ] = california.target
```

```
data.head ()
```

index	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Price
0	8.3252	41.0	6.984126984126984	1.0238095238095237	322.0	2.5555555555555554	37.88	-122.23	4.526
1	8.3014	21.0	6.238137082601054	0.9718804920913884	240.0	2.109841827768014	37.86	-122.22	3.585
2	7.2574	52.0	8.288135593220339	1.073446327683616	496.0	2.8022598870056497	37.85	-122.24	3.521
3	5.6431	52.0	5.8173515981735155	1.0730593607305936	558.0	2.547945205479452	37.85	-122.25	3.413
4	3.8462	52.0	6.281853281853282	1.0810810810810811	565.0	2.1814671814671813	37.85	-122.25	3.422

Let's check the data for missing values:

```
data.info()
```

```
< class ' pandas.core.frame.DataFrame ' >
```

```
RangeIndex : 20640 entries, 0 to 20639
```

```
Data columns (total 9 columns):
```

```
# Column Non-Null Count Dtype
```

```
-----
```

```
0 MedInc 20640 non- null float64
```

```
1 HouseAge 20640 non- null float64
```

```
2 AveRooms 20640 non- null float64
```

```
3 AveBedrms 20640 non- null float64
```

```
4 Population 20640 non-null float64
```

```
5 AveOccup 20640 non- null float64
```

```
6 Latitude 20640 non- null float64
```

```
7 Longitude 20640 non- null float64
```

```
8 Price 20640 non- null float64
```

```
dtypes : float64(9)
```

```
memory usage : 1.4 MB
```

We see that there are no gaps in the data. Furthermore, it's clear that all data is expressed on numerical scales. This means that this dataset doesn't require any special processing; it's already quite clean. Now we can display the basic statistics for the dataset :

```
data.describe ( ). round (2)
```

index	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Price
count	20640.0	20640.0	20640.0	20640.0	20640.0	20640.0	20640.0	20640.0	20640.0
mean	3.87	28.64	5.43	1.1	1425.48	3.07	35.63	-119.57	2.07
std	1.9	12.59	2.47	0.47	1132.46	10.39	2.14	2.0	1.15
min	0.5	1.0	0.85	0.33	3.0	0.69	32.54	-124.35	0.15
25%	2.56	18.0	4.44	1.01	787.0	2.43	33.93	-121.8	1.2
50%	3.53	29.0	5.23	1.05	1166.0	2.82	34.26	-118.49	1.8
75%	4.74	37.0	6.05	1.1	1725.0	3.28	37.71	-118.01	2.65
max	15.0	52.0	141.91	34.07	35682.0	1243.33	41.95	-114.31	5.0

Now let's identify the target variable and factors:

```
y = data[ 'Price']
```

```
X = data.drop ( 'Price', axis=1)
```

Let's derive the form of the resulting arrays:

```
y.shape , X.shape
```

```
((442,), (442, 10))
```

Let's begin training and evaluating the model. We'll import linear regression from *the sklearn library's set of linear models*:

```
from sklearn.linear_model import LinearRegression
```

```
model = LinearRegression ()
```

```
model.fit (X, y)
```

As in previous works, we will derive the coefficients of the model, since in linear models they have some meaning:

```
print( "Coefficients: \n", model.coef _)
```

Coefficients:

```
[ 4.36693293e -01 9.43577803e-03 -1.07322041e-01 6.45065694e-01
```

```
-3.97638942 e -06 -3.78654265 e -03 -4.21314378 e -01 -4.34513755 e -01]
```

Let's derive the coefficients along with the names of the corresponding attributes:

```
_ = [ print( k, v) for k, v in zip( X.columns , model.coef _)]
```

```
MedInc 0.4366932931343245
```

```
HouseAge 0.009435778033237972
```

```
AveRooms -0.10732204139090447
```

```
AveBedrms 0.645065693519812
```

```
Population -3.976389421211576e-06
```

```
AveOccup -0.003786542654971
```

```
Latitude -0.42131437752714385
```

Longitude -0.43451375467477743

Analyze this data along with the interpretation of the dataset attributes and draw conclusions about which factors influence real estate prices in California.

As in the linear regression model, this vector does not include a free coefficient. It is stored in a separate class field:

```
print ( " Intercept :\n ", model . intercept_ )
```

Intercept :

-36.94192020718441

Let's make the model's predictions and display the first few points:

```
y_pred = model.predict (X)
```

```
print ( y_pred [:5])
```

[4.13164983 3.97660644 3.67657094 3.2415985 2.41358744]

For comparison, we will derive the actual corresponding values of the target variable:

```
print( y[:5])
```

0 4.526

1 3.585

2 3.521

3 3.413

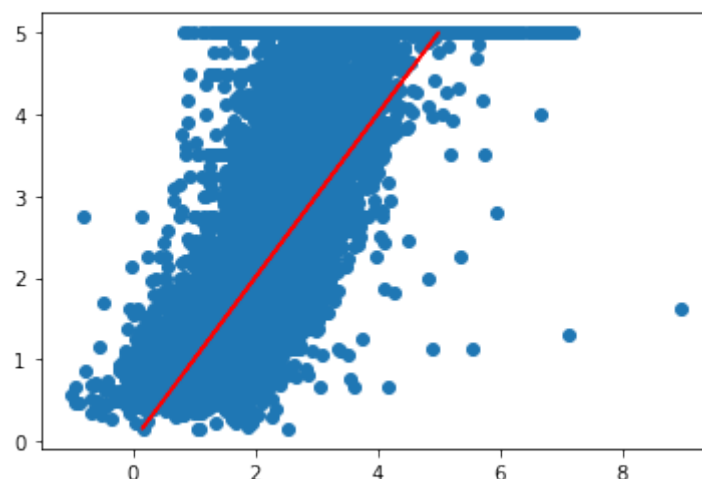
4 3.422

Name: Price, dtype : float64

Of course, analyzing data this way is inconvenient. It's better to plot a graph showing the relationship between the actual and predicted values of the target variable:

```
plt.scatter ( y_pred , y)
```

```
plt.plot (y, y, c='r')
```



In this graph, the closer the points are to the centerline, the more accurate the model's forecasts. In this case, the spread is quite large. To evaluate the model's effectiveness numerically, we again turn to the built-in metric implemented by *the score method* —the coefficient of determination:

```
model.score (X, y)
```

0.606232685199805

A level of 0.6 indicates that the model could be more accurate. Let's try building a different model—a polynomial regression. It's hoped that introducing polynomial features can significantly improve the model's accuracy:

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures (5). fit_transform (X)
```

In this case, we'll use fifth-degree polynomial features. You can experiment with other degrees of polynomial. Let's construct a prediction for analysis:

```
polynomial = LinearRegression ()
```

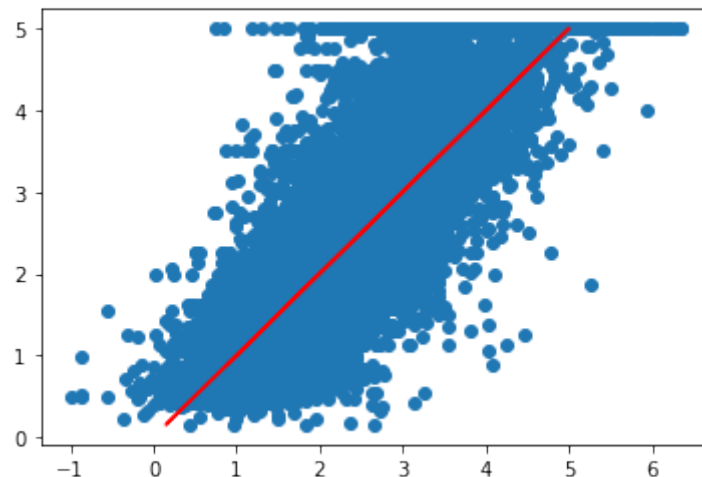
```
polynomial.fit ( poly, y)
```

```
y_pred_poly = polynomial.predict ( poly)
```

From this data, you can construct a graph similar to the previous one:

```
plt.scatter ( y_pred_poly , y)
```

```
plt.plot (y, y, c='r')
```



This graph suggests that the model has become somewhat more accurate. But this improvement can be seen more concretely using the selected metric:

```
polynomial.score ( poly , y)
```

```
0.7406575543454206
```

The metric shows that the second model is about 14 percentage points better than the first.

Security questions

1. What is the difference between using different regression models in a library ? *sklearn* from classification models?
2. What does the coefficient of determination for a regression model show?
3. What is the significance of linear regression coefficients?
4. What attributes does a linear regression object have?

Assignments for independent completion

1. What other information can be inferred from a trained model? Try changing the arguments when creating the model and see how this affects the prediction quality.
2. Try applying other regression models to the same problem. For each, display a regression visualization and accuracy score. We recommend exploring the following models:
 - i. Support vector machine
 - a. Without core
 - b. With a Gaussian kernel
 - c. With a polynomial kernel
 - ii. Nearest neighbor method

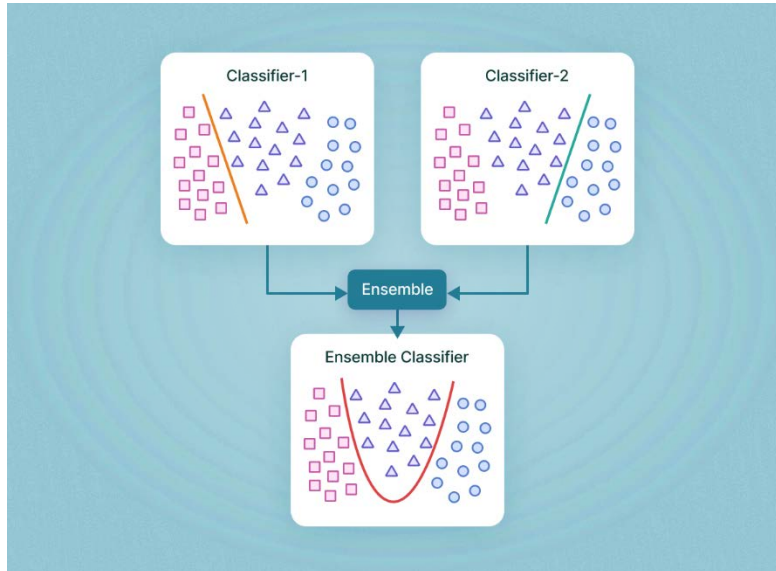
- iii. Multilayer perceptron
 - iv. Decision tree
 - v. (*) Other methods:
 - a. Ridge regression
 - b. Lasso Regression
 - c. ElasticNet Regression
 - d. Random Forest
 - e. Begging
 - f. Other models on request
3. Write a function that automatically trains all of the listed models and produces an accuracy score for each.
 4. Repeat the entire analysis for a different dataset - built into *sklearn* dataset *diabetes* .

Lab 7

Topic: Ensemble methods

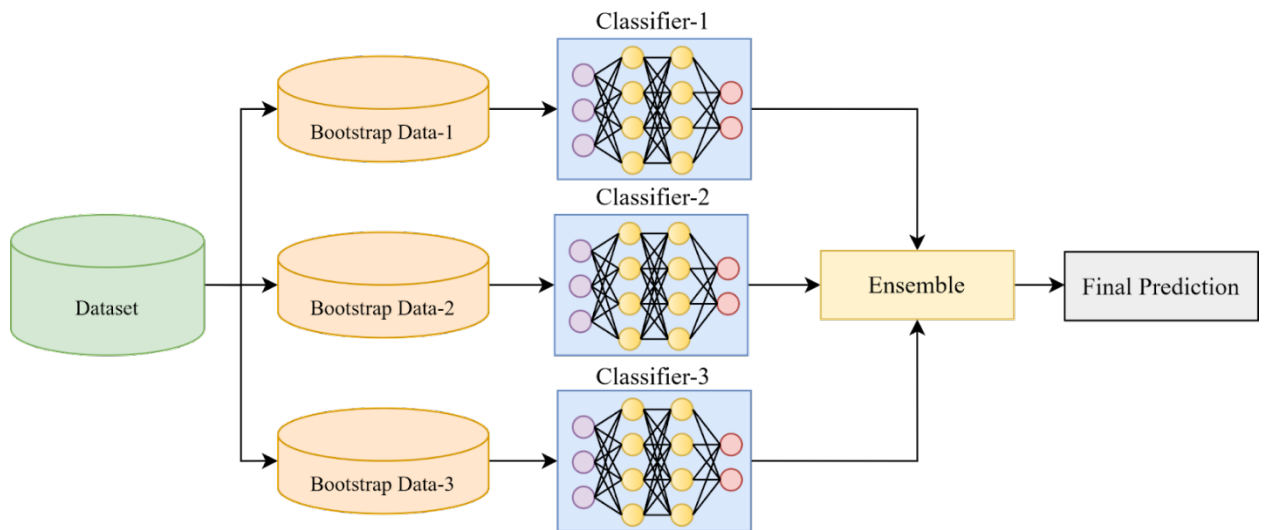
The purpose of the work: to consolidate knowledge, skills and abilities in working with ensemble methods

Methodological guidelines



1. Ensemble learning is a technique for training multiple models on the same task to improve predictive performance.
2. An ensemble is a set of multiple machine learning models and a way to combine and train them.
3. Ensembling is a way to improve efficiency at the cost of increasing computational complexity.
4. Used in both supervised and unsupervised learning.
5. The ensemble functions as a single model with an input and output. The training technique is similar.
6. Ensembles can also overfit and underfit .
7. The law of diminishing returns applies to the construction of an ensemble.

Task 1. Begging



```

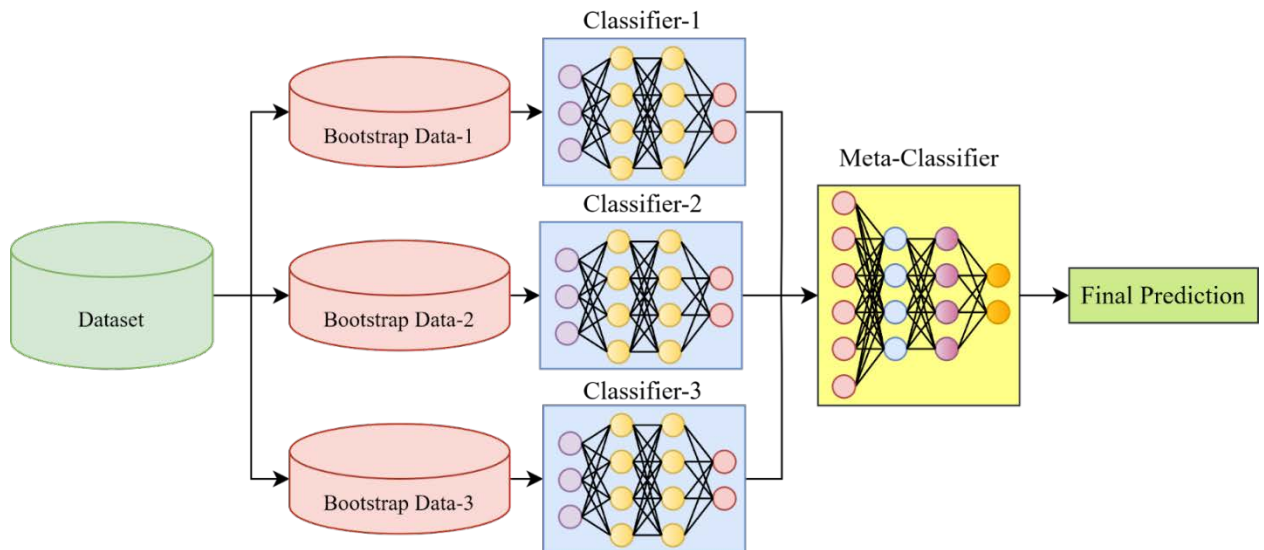
1  clf1 = LogisticRegression ( random_state =1)
2  clf2 = RandomForestClassifier ( n_estimators =50, random_state =1)
3  clf3 = GaussianNB ()
4
5  eclf = VotingClassifier (
6  estimators=[(' lr ', clf1), (' rf ', clf2), (' gnb ', clf3)],
7  voting =' hard ')
8
1  from sklearn.ensemble import BaggingClassifier
2  from sklearn.tree import DecisionTreeClassifier
3
4  tree = DecisionTreeClassifier ()
5  bagging_clf = BaggingClassifier ( base_estimator = tree,
6  n_estimators = 1500,
7  random_state = 42)
8  bagging_clf.fit ( X_train , y_train )

```

Conclusions:

1. Bagging (bootstrap aggregating) is a method of training several models on subsamples of the training set.
2. The training set is divided into subsets with repetitions. A separate model is trained on each subset.
3. Combination of predictors is carried out by averaging or voting.
4. Begging without resampling is a simple voting ensemble.
5. Voting can be hard or soft.
6. Significantly reduces model variability, thus combating overfitting (errors are averaged out).
7. The resulting model is less sensitive to anomalies and errors in the data.
8. Predictor training can be parallelized.
9. Disadvantages: loss of interpretability, computational complexity.

Task 2. Stacking

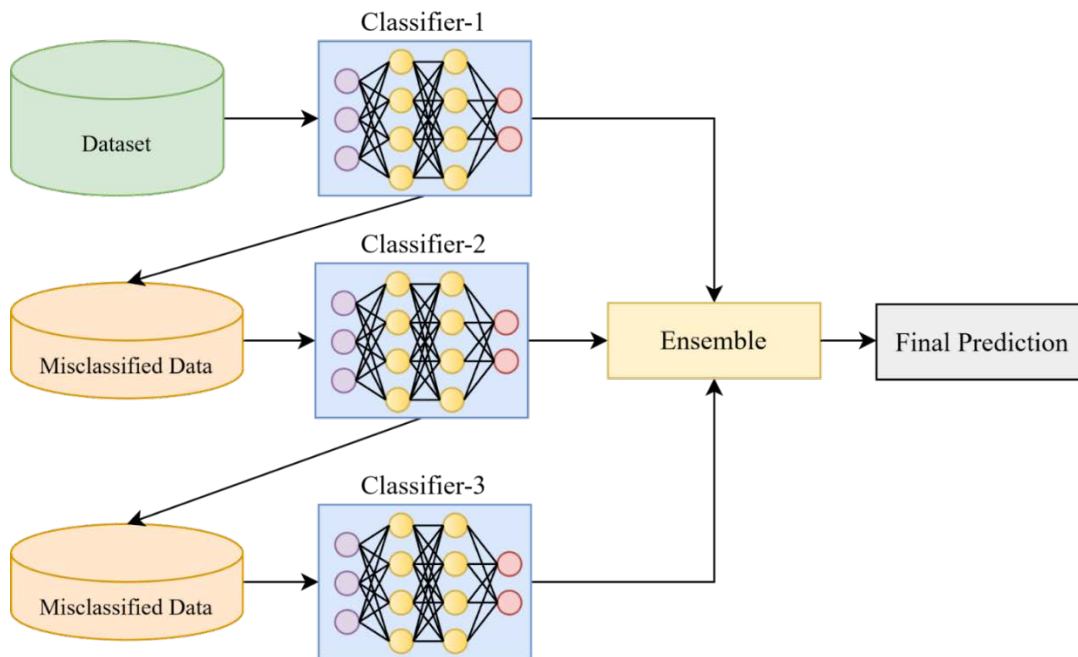


```
1 from sklearn.ensemble import GradientBoostingRegressor
2 from sklearn.ensemble import StackingRegressor
3 final_estimator = GradientBoostingRegressor (
4     n_estimators =25, subsample=0.5,
5     min_samples_leaf =25, max_features =1,
6     random_state = 42)
7 reg = StackingRegressor (
8     estimators=estimators,
9     final_estimator = final_estimator )
```

Conclusions:

1. Stacking (Stackedgeneralization) - similar to bagging , but with the addition of a meta-predictor (second-order model) .
2. The metapredictor uses the outputs of ensemble predictors as input.
3. Stacking involves splitting the training set into K folds , as in cross - validation .
4. Due to the combination of advantages of different models, it is able to significantly increase accuracy.
5. Takes longer to learn due to limited parallelization .
6. Prone to overfitting due to the presence of a second-order model.

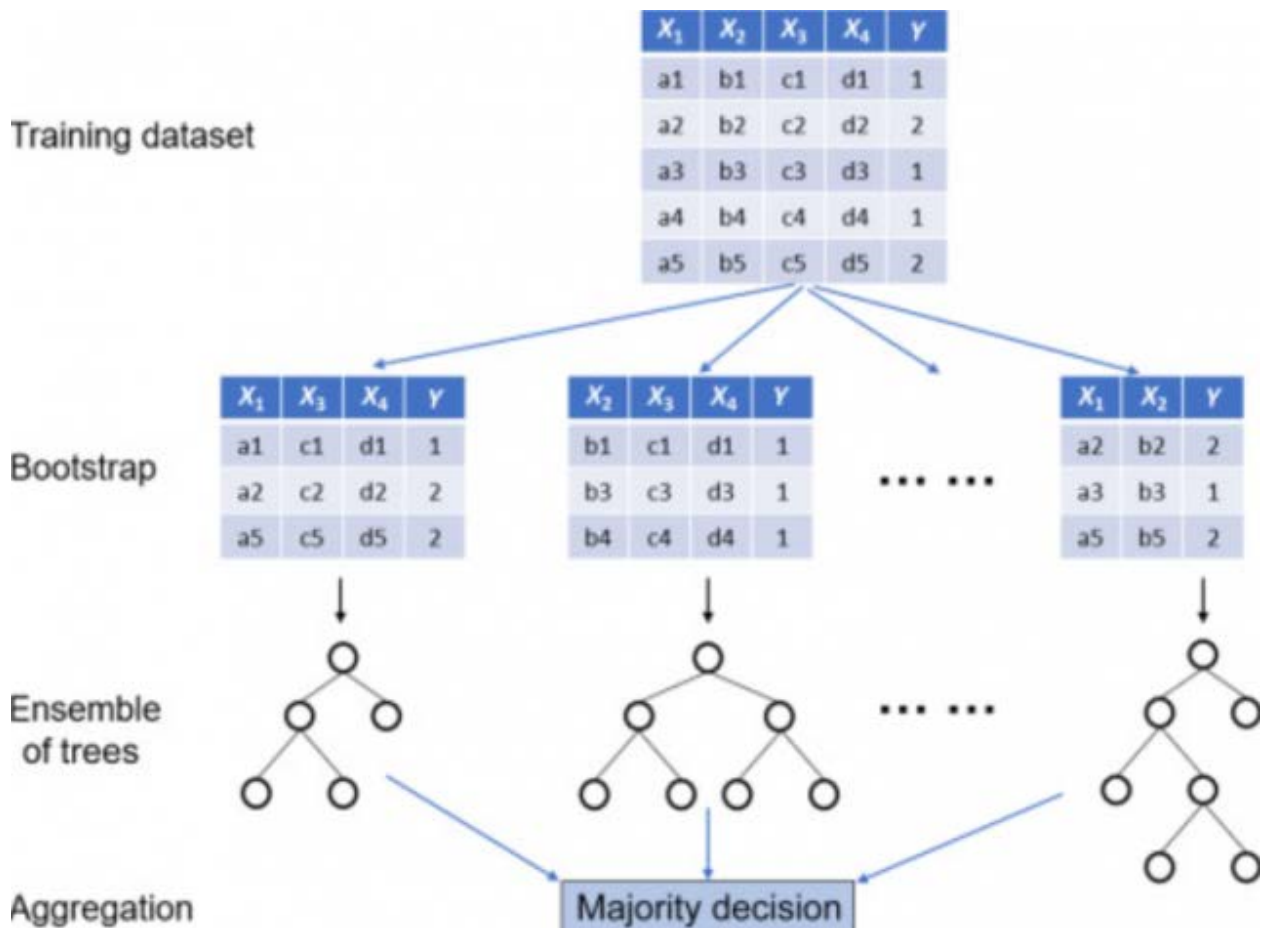
Task 3. Boosting



Conclusions:

1. The idea of boosting is to sequentially analyze data using models from the ensemble.
2. The first ensemble model is trained on the entire sample.
3. The next model receives only the data on which the previous model was wrong.
4. Subsequent models may correct the mistakes of previous ones.
5. Models are assigned weights based on their performance.
6. May reduce model bias.
7. Simple models with high bias and low variance are usually taken as predictors.
8. Due to sequential data processing, it is computationally complex.
9. Sensitive to outliers and anomalies.

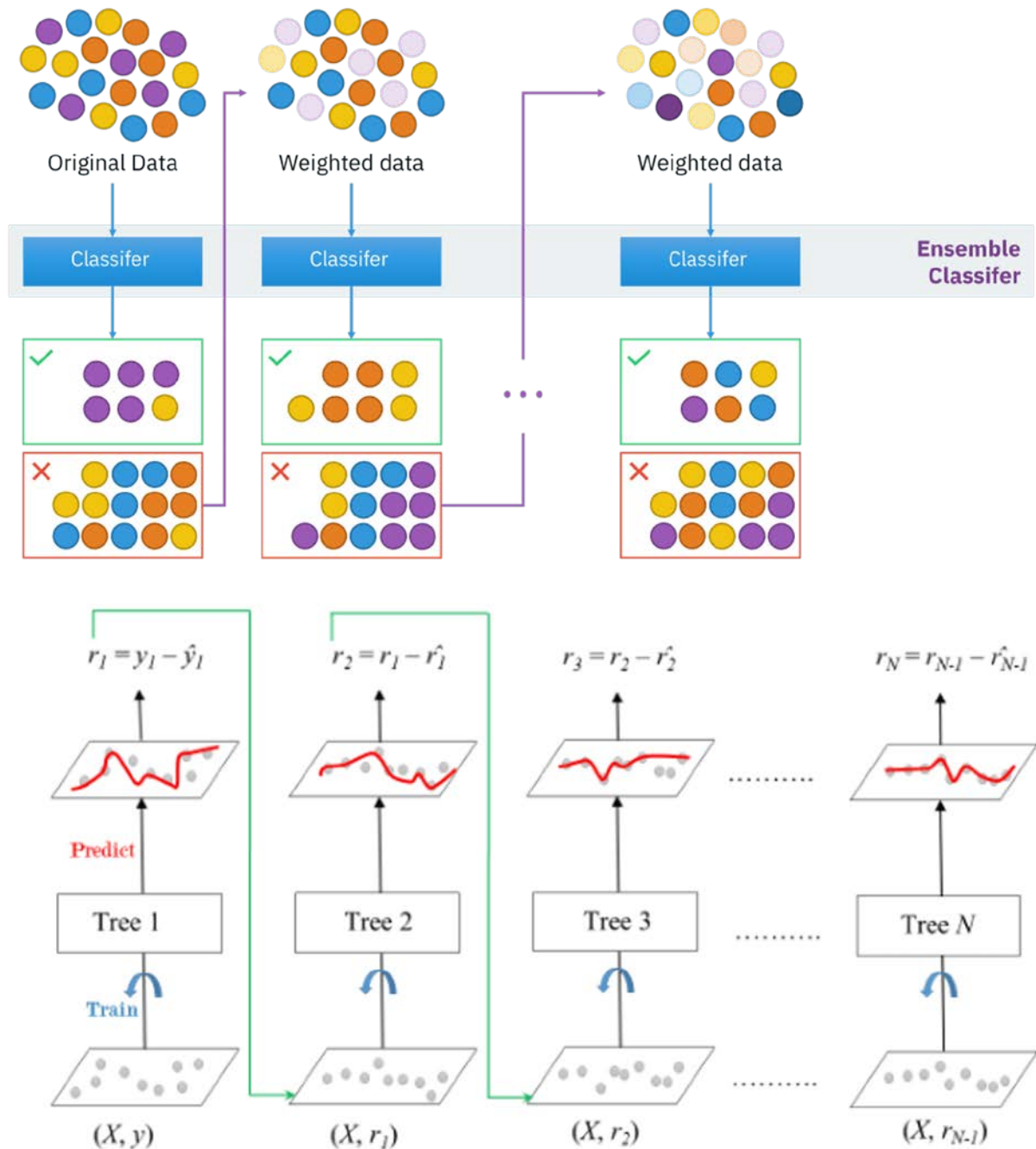
Task 4. Random Forest



Conclusions:

1. Random forest is a bagging over a set of decision trees.
2. Decision trees have high variance, which can be reduced by bagging .
3. Random forests typically use both row-wise and column-wise sampling .
4. Used for both classification and regression.
5. Less susceptible to overtraining.
6. The greater the number of trees, the greater the regularization effect.
7. You can customize the maximum number of features for individual trees.

Task 5. Gradient Boosting



Conclusions:

1. Adaptive boosting (AdaBoost) assigns weights to training sample points based on whether they were correctly identified by the first classifiers.
2. Subsequent classifiers focus on “difficult” cases proportionally to the weights.
3. Adaptive boosting is often applied to binary classification problems.
4. Gradient Boosting (GBM) passes on to subsequent models the amount of bias from previous models.
5. Gradient boosting uses decision trees as individual predictors.
6. XGBoost is a parallelizable, highly efficient implementation of gradient boosting .
7. XGBoost can handle large amounts of data.
8. XGBoost is prone to overfitting, but uses regularization.
9. XGBoost is quite demanding in terms of RAM.

10. Other famous implementations - LightGBM , CatBoost

Practical work 8

Topic: Stochastic Search

Objective of the work: consolidation of knowledge, skills, and abilities in working with stochastic gradient descent

Methodological guidelines

Gradient Descent (SGD) is a simple yet highly effective approach for training linear classifiers and regressors with convex loss functions, such as (linear) support vector machines and logistic regression. Although SGD has been around in the machine learning community for a long time, it has only recently received significant attention in the context of large-scale learning.

SGD has been successfully applied to large-scale and sparse machine learning problems, often encountered in text classification and natural language processing. Given the data sparsity, classifiers in this module easily scale to problems with more than 10^5 training examples and more than 10^5 features.

Strictly speaking, SGD is just an optimization technique and doesn't correspond to a specific family of machine learning models. It's simply a way to train a model. Often, an instance of `SGDClassifier` or `SGDRegressor` will have an equivalent model in the scikit-learn API, potentially using a different optimization technique. For example, using `SGDClassifier ( loss = 'log_loss '` results in logistic regression, that is, a model equivalent to `LogisticRegression`, which is fitted using SGD instead of one of the other algorithms in `LogisticRegression`. Similarly, `SGDRegressor ( loss = 'squared_error ', penalty = 'l2')` and `Ridge` solve the same optimization problem using different means.

The advantages of stochastic gradient descent are:

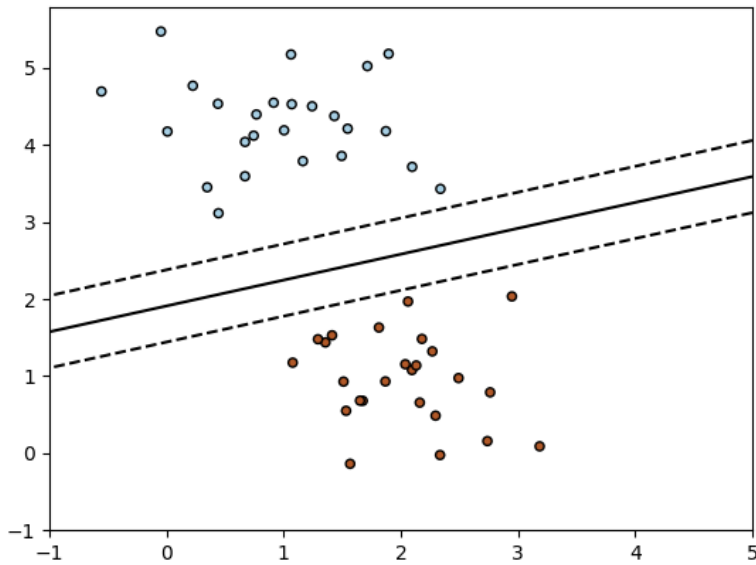
- Efficiency.
- Ease of implementation (many possibilities for code customization).

The disadvantages of stochastic gradient descent include:

- SGD requires a number of hyperparameters, such as the regularization parameter and the number of iterations.
- SGD is sensitive to feature scaling.

Task 1. Stochastic gradient descent for classification problem

SGDClassifier class implements a standard stochastic gradient descent learning procedure that supports various loss functions and penalties for classification. The decision boundary of an **SGDClassifier** **trained** with `hinge` loss, which is equivalent to a linear SVM, is shown below.



Like other classifiers, SGD must be equipped with two arrays: an X array of shape (n_samples , n_features) containing the training samples, and a y array of shape (n_samples ,) containing the target values (class labels) for the training samples:

```
>>> fromsklearn.linear_modelimport SGDClassifier
>>> X = [[ 0., 0.], [1., 1.]]
>>> y = [0, 1]
>>> clf = SGDClassifier ( loss="hinge", penalty="l2", max_iter =5)
>>> clf.fit (X, y)
SGDClassifier ( max_iter = 5)
```

Once trained, the model can be used to predict new values:

```
>>> clf.predict ([[2., 2.]])
array ([1])
```

SGD fits a linear model to the training data. The coef attribute contains the model parameters:

```
>>> clf.coef _
array ( [[9.9..., 9.9...]])
```

intercept attribute contains the intercept (also known as offset or deviation):

```
>>> clf.intercept _
array ( [-9.9...])
```

Whether the model uses an intercept, i.e. a shifted hyperplane, is controlled by the fit_intercept parameter .

The signed distance to the hyperplane (computed as the dot product between the coefficients and the input sample plus the intercept) is given by **SGDClassifier.decision_function** :

```
>>> clf.decision _function ([[2., 2.]])
array ( [29.6...])
```

A specific loss function can be specified using the loss parameter . SGDClassifier supports the following loss functions:

- loss =" hinge ": (soft-margin) linear support vector machine,
- loss =" modified_huber ": smoothed hinge loss,
- loss =" log_loss ": logistic regression,
- and all regression losses are lower. In this case, the target is coded as -1 or 1, and the problem is treated as a regression problem. The predicted class corresponds to the sign of the predicted target.

The first two loss functions are lazy, they only update the model parameters if an example violates the margin constraint, which makes training very efficient and can lead to sparse models (i.e. with a lot of zero coefficients) even if the L2 penalty is used.

Using `loss = "log_loss"` or `loss = "modified_huber"` enables the `predict_proba` method, which produces a vector of probability estimates $P(y|x)$ per sample x :

```
>>> clf = SGDClassifier ( loss="log_loss", max_iter=5).fit(X, y)
```

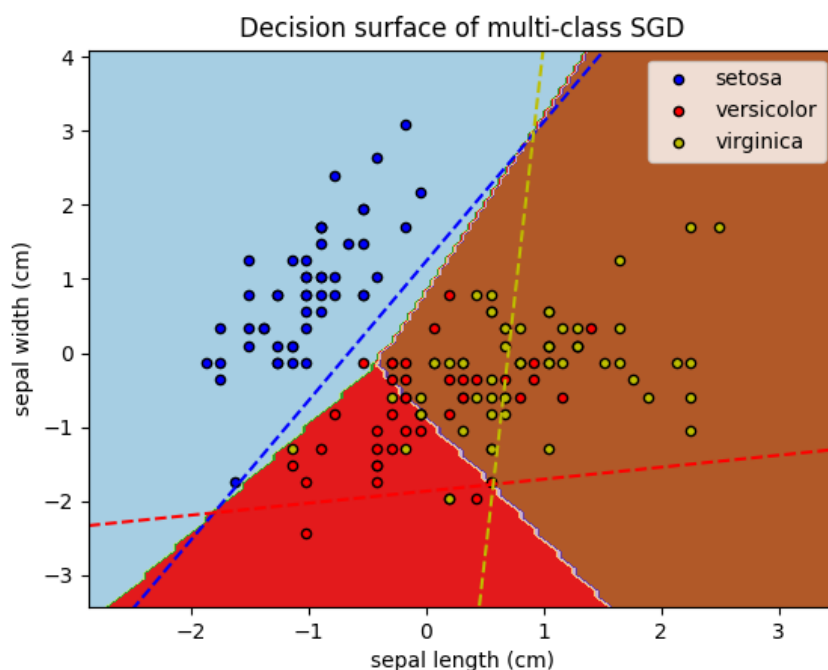
```
>>> clf.predict_proba ([[1., 1.]])
```

```
array ( [[0.00..., 0.99...]])
```

A specific penalty can be specified using the `penalty` parameter. SGD supports the following penalties:

- `penalty = "l2"`: L2 penalty rate on `coef_`.
- `penalty = "l1"`: L1 penalty rate on `coef_`.
- `penalty = "elasticnet"`: convex combination of penalties L1 and L2; $(1 - l1_ratio) * L2 + l1_ratio * L1$.

By default, `penalty = "l2"` is set. The L1 penalty leads to sparse solutions, causing most coefficients to vanish. The elastic net addresses some of the shortcomings of the L1 penalty in the presence of highly correlated features. The `l1_ratio` parameter controls the convex combination of the L1 and L2 penalties. **SGDClassifier** supports multi-class classification by combining multiple binary classifiers using a one-versus-all (OVA) approach. For each K class, a binary classifier is trained that distinguishes between this and all $K-1$ other classes. During testing, we calculate a confidence score (i.e., signed distances to the hyperplane) for each classifier and select the class with the highest confidence. The figure below shows the OVA approach on the Iris dataset. The dashed lines represent the three OVA classifiers; the background colors indicate the decision surface generated by the three classifiers.



In the case of multi-class classification, `coef_` is a two-dimensional array of shapes $(n_classes, n_features)$, and `intercept_` is a one-dimensional array of shapes $(n_classes,)$. The i -th row of `coef_` contains the weight vector of the OVA classifier for the i -th class; classes are indexed in ascending order (see the `classes_` attribute).

Note that in principle, since they allow for a probabilistic model, `loss = "log_loss"` and `loss = "modified_huber"` are more suitable for one-versus-all classification.

SGDClassifier supports weighted classes and weighted instances via the `class_weight` and `sample_weight` training parameters. For more information, see the examples below and the **SGDClassifier.fit**

docstring .

SGDClassifier supports averaged SGD (ASGD). Averaging can be enabled by setting `average = True` . ASGD performs the same updates as regular SGD, but instead of using the latest coefficient value as the `coef` attribute (i.e., the value of the last update), `coef` is set to **the average** coefficient value across all updates. The same is done for the intercept attribute . When using ASGD, the learning rate can be higher and even constant, which leads to faster training on some datasets.

For classification with logistic loss, another variant of SGD with an averaging strategy is available - the StochasticAverageGradient (SAG) algorithm, available as an algorithm in **LogisticRegression** .

Task 2. Stochastic gradient descent for a regression problem

SGDRegressor class implements a simple stochastic gradient descent learning procedure that supports various loss functions and penalties for training linear regression models. **SGDRegressor** is well suited for regression problems with large numbers of training samples (>10,000); for other problems, we recommend **Ridge** , **Lasso** , or **ElasticNet** .

A specific loss function can be specified via the `loss` parameter . SGDRegressor supports the following loss functions:

- `loss = "squared_error"`: Ordinary least squares,
- `loss = "huber"`: hingeloss for robust regression,
- `loss = "epsilon_insensitive"`: support vector linear regression.

For robust regression, you can use the `hingeloss` and `epsilon_insensitive` functions . The width of the insensitivity region is specified using the `epsilon` parameter . This parameter depends on the scale of the target variables.

`penalty` parameter determines the regularization used (see the description above in the classification section).

SGDRegressor also supports averaged SGD (here again see the description above in the classification section).

For regression with quadratic loss and l2 penalty, another variant of SGD with an averaging strategy is available - the StochasticAverageGradient (SAG) algorithm, available as an algorithm in **Ridge** .

MINISTRY OF SCIENCE AND HIGHER EDUCATION OF THE RUSSIAN FEDERATION
FEDERAL STATE AUTONOMOUS EDUCATIONAL
INSTITUTION OF HIGHER EDUCATION
NORTH CAUCASIAN FEDERAL UNIVERSITY

Methodological guidelines
on organizing and conducting independent work
in the discipline
**" MACHINE LEARNING THEORY / ТЕОРИЯ МАШИННОГО
ОБУЧЕНИЯ "**
for students in the field of study
02.04.01 Mathematics and Computer Science
direction (profile)
"Training Program for Electronic Information"

General Provisions

Independent work is the planned educational, educational and research, scientific research work of students, carried out outside of class (classroom) time on assignment and with the methodological guidance of the teacher, but without his direct participation (with partial direct participation of the teacher, leaving the leading role to the work of the students).

Student independent work (SIW) at a university is an important form of student learning and research activity. Student independent work plays a significant role in the rating system of learning.

The main types of independent work for students include:

- formation and assimilation of the content of lecture notes based on the educational literature recommended by the lecturer, including information educational resources (electronic textbooks, electronic libraries, etc.);
- writing reports;
- preparation for seminars, practical and laboratory work, their design;
- compilation of an annotated list of articles from relevant journals in the fields of knowledge (pedagogical, psychological, methodological, etc.);
- implementation of educational and research work, project activities;
- preparation of practical developments and recommendations for solving problematic situations;
- completing homework assignments in the form of solving individual problems, performing typical calculations, computer calculations and individual work on individual sections of the discipline content, etc.;
- computer-based current self-monitoring and progress monitoring based on electronic training and assessment tests;
- completion of coursework (projects) within the framework of disciplines;
- completion of final qualifying work, etc.

The methodology for organizing students' independent work depends on the structure, nature, and characteristics of the discipline being studied, the number of hours for its study, the type of assignments for students' independent work, the individual qualities of students, and the conditions of educational activities.

The process of organizing students' independent work includes the following stages:

- preparatory (defining goals, drawing up a program, preparing methodological support, preparing equipment);
- basic (implementation of the program, use of techniques for searching for information, assimilation, processing, application, transfer of knowledge, recording of results, self-organization of the work process);
- final (assessment of significance and analysis of results, their systematization, assessment of the effectiveness of the program and work methods, conclusions on areas of labor optimization).

Independent work on the course "**Machine Learning Theory / Теория машинного обучения**" is aimed at developing the following **competencies** :

ОПК-1. Capable of identifying, formulating, and solving current and significant problems in applied and computer mathematics.

ПС-5. Able to present and adapt mathematical knowledge, programming methods, and information technology in various ways, taking into account the audience's level.

1. The purpose and objectives of independent work

The main goal of organizing and implementing independent work coincides with the goal of student education – the formation of an appropriate set of competencies for a future bachelor's degree in the field of study "Mathematics and Computer Science".

When organizing independent work, developing the ability to work independently to acquire knowledge, skills, and the ability to organize educational and research activities is an important and essential condition. The goal of students' independent work is to master fundamental knowledge, professional skills, and relevant professional activities, as well as experience in creative and research activities. Students' independent work promotes the development of independence, responsibility, self-discipline, and a creative approach to solving academic and professional problems.

The objectives of the SRS are:

- systematization and consolidation of the acquired theoretical knowledge and practical skills of students;
- deepening and expanding theoretical knowledge;
- development of skills in using regulatory, legal, reference documentation and specialized literature;
- development of students' cognitive abilities and activity: creative initiative, independence, responsibility and organization;
- development of independent thinking, abilities for self-development, self-improvement and self-realization;
- development of research skills;
- use of material collected and obtained during independent study at seminars, in practical and laboratory classes, when writing coursework and final qualifying papers, for effective preparation for final tests and exams.

2. Technological map of independent student work

Codes of implemented competencies	Type of student activity	Assessment tools and technologies	Volume of hours, including (academic)		
			SRS	Contact work with the teacher	Total
1st semester					
OPK-1, PC-5	Independent study of literature	Interview	8	2	10
About PC-1, PC-5	Preparing for laboratory classes	Defense of laboratory work	10	2	12
OPK-1, PC-5	Preparing for the exam	Exam questions	52	2	54
Total for 2nd semester			70	6	76
Total			70	6	76

3. Procedure for completing independent work by a student

3.1. Methodological recommendations for working with educational literature

When working with books, you need to select literature, learn to read it correctly, and keep records. To select literature, the library uses alphabetical and systematic catalogs.

It is important to remember that rational skills in working with a book always mean great savings of time and effort.

The instructor teaching the course recommends the appropriate selection of textbooks. Required literature may also be listed in the course's teaching materials.

When studying the material in a textbook, you should move on to the next question only after correctly understanding the previous one, describing on paper all the calculations and computations (including those that are omitted from the textbook or given in the lecture for independent derivation).

When studying any discipline, independent individual work plays a large and important role.

Particular attention should be paid to defining the course's key concepts. Students should thoroughly analyze examples that illustrate these definitions and be able to construct similar examples independently. It's important to develop a clear understanding of what you're learning. Taking supporting notes is helpful. When studying the material from the textbook, it's helpful to supplement your lecture notes in the designated margins in your notebook. You should also note any questions you've identified for consultation with the instructor.

It is recommended to highlight the conclusions obtained as a result of the study in the notes so that they are better remembered when rereading the notes.

Experience shows that many students find it helpful to create a reference sheet containing the most important and frequently used formulas and concepts. Such a sheet helps students remember formulas and key points of the lecture, and can also serve as a constant reference for them.

Reading scientific text is part of cognitive activity. Its goal is to extract the necessary information from the text. The effectiveness of the action largely depends on how consciously the reader approaches the printed word (finding the necessary information, assimilating it in whole or in part, critically analyzing the material, etc.).

There are **four main attitudes towards reading scientific text** :

information retrieval (the task is to find and highlight the desired information)

assimilative (the reader's efforts are aimed at fully comprehending and remembering both the information presented by the author and the entire logic of his reasoning)

analytical-critical (the reader strives to critically comprehend the material, having analyzed it, and determined his attitude toward it)

creative (creates in the reader a readiness in one form or another - as a starting point for his own reasoning, as an image for action by analogy, etc. - to use the author's judgments, his train of thought, the result of observation, the developed methodology, to supplement them, to subject them to new testing).

The main types of systematic recording of what has been read:

Annotation is an extremely brief, coherent description of a book (article) viewed or read, its content, sources, nature and purpose;

Planning is a brief logical organization of the text, revealing the content and structure of the material being studied;

Thesis is a concise reproduction of the author's main assertions without using factual material;

Quoting is the verbatim copying of excerpts and extracts from the text that most significantly reflect a particular idea of the author;

Note-taking is a brief and sequential presentation of the content of what has been read.

A summary is a complex way of presenting the contents of a book or article in a logical sequence. A summary integrates previous notes and allows for a comprehensive understanding of the book or article's content. Therefore, the ability to create an outline, make notes, take excerpts, and other notes determines the technique of summary writing.

Guidelines for writing notes:

1. Read the text carefully. Look up any unfamiliar words in reference books. When taking notes, be sure to include reference information in the margins of your notes;
2. Highlight the main points, make a plan;
3. Briefly formulate the main points of the text, note the author's argumentation;
4. Take notes on the material, strictly following the outline. When taking notes, try to express your thoughts in your own words. Keep your notes clear and concise.
5. Write quotes correctly. When quoting, keep in mind the brevity and significance of the idea.

In the summary, it is advisable to present not only thesis statements but also their evidence. When formatting the summary, strive for the conciseness of each sentence. The author's thoughts should be presented concisely, with care taken to maintain style and expressiveness. The number of additional elements in the summary should be logically justified, and entries should be arranged in a specific sequence consistent with the logical structure of the work. Leave margins for clarification and addition.

Mastering note-taking skills requires a student to be determined and to work independently on a daily basis.

3.2. Methodological recommendations for preparation for practical and laboratory classes

To maximize the benefit of practical and laboratory classes, it's important to remember that exercises and problem solving are based on lecture material and typically involve a detailed analysis of specific topics covered in the lecture course. It's important to emphasize that only after the lecture material has been mastered from a specific perspective (namely, the perspective from which it is presented in lectures) will it be reinforced in practical classes, both through discussion and analysis of the lecture material and through solving problem situations and tasks. Under these conditions, students will not only master the material well but also learn to apply it in practice, and will also receive additional incentive (and this is very important) to actively engage with the lecture.

When independently solving problems, it's important to justify each step of the solution based on the theoretical principles of the course. If a student sees several possible solutions to a problem (problem), they should be compared and the most efficient one selected. It's helpful to create a brief plan for solving the problem (problem) before beginning calculations. Solutions to problematic problems or examples should be presented in detail, with calculations arranged in strict order, separating auxiliary calculations from the main ones. Solutions should be accompanied by comments, diagrams, drawings, and illustrations, if necessary.

It's important to remember that the solution to each learning problem should be followed through to the final logical answer required by the conditions, and, if possible, with a conclusion. The resulting answer should be verified using methods specific to the nature of the problem. It's also helpful (if possible) to solve the problem in several ways and compare the results. Problem solving of this type should be continued until solid solving skills are acquired.

3.3. Guidelines for self-assessment of knowledge

After studying a specific topic using notes in the notes and textbook, as well as solving a sufficient number of relevant problems in practical classes and independently, the student is advised to conduct a self-assessment of the acquired knowledge by answering test questions on the topic studied.

If necessary, you need to carefully review the material again.

Sometimes, a student's lack of understanding of a particular topic only becomes apparent upon further study. In this case, it's necessary to go back and review the poorly understood material. An important criterion for mastering theoretical material is the ability to solve problems or pass tests on the material covered. However, it's important to remember that a correct solution to a problem can result from applying rote memorized formulas without understanding the essence of the theoretical principles.

3.4. Methodological recommendations for preparation for exams and tests

The study of many general and specialized disciplines culminates in an exam. Preparation for the exam helps consolidate, deepen, and generalize the knowledge acquired during the course of study, as well as apply it to solving practical problems. By preparing for the exam, students address existing gaps in their knowledge, deepen, systematize, and organize their understanding. During the exam, students demonstrate what they have acquired during their studies in a specific subject.

An exam session is a series of exams set by the curriculum. There is a 3-4 day interval between exams. Don't assume that 3-4 days is enough to successfully prepare for the exams.

During these 3-4 days, students should systematize their existing knowledge. At the pre-exam consultation, they will be introduced to the basic requirements and have any questions answered. Therefore, attendance at the consultations is mandatory.

The requirements for preparing for exams are the same as for classes during the semester, but they must be adhered to more strictly. First, maintaining a daily routine is crucial: sleep at least 8 hours a day, and classes end no later than 2-3 hours before bedtime. The optimal time for classes is morning and afternoon. During breaks between classes, walks in the fresh air and gentle exercise are recommended. Second, having good lecture notes is essential. Even if a lecture is missed, it is necessary to make it up (copy it in the department), reflect on it, and resolve any questions that arise to ensure conscious memorization of the material. Third, when preparing for exams, students should have a good textbook or a summary of the literature read as directed by the instructor during the semester. Reference sheets can be used effectively here.

First, review all the material for the course you're taking, noting any difficult questions. Be sure to thoroughly understand them. Finally, it's a good idea to review the key concepts again, using the reference sheets.

Systematic preparation for classes throughout the semester will allow you to use the time during the exam session to systematize your knowledge.

Exam questions

in the discipline "**Machine Learning Theory / Теория машинного обучения**"

1. Types of machine learning problems
2. The subject and objectives of machine learning and data analysis.
3. Basic principles, objectives and approaches, application in various fields of science and industry.
4. The main stages of the evolution of machine learning algorithms.
5. Metric classifiers
6. General view of the metric classifier.

7. K-Nearest Neighbors Algorithm.
8. Algorithms for selecting standards.
9. Clustering algorithms
10. Clustering algorithms with a fixed number of clusters.
11. Density clustering algorithms.
12. Hierarchical clustering.
13. Decision trees
14. Rules and quality analysis (precision, recall). ROC curve analysis. Decision tree construction algorithm. Information gain criterion and Gini criterion. Decision tree forests.
15. Linear classifiers
16. Perceptron and separating hyperplane. Transition to higher-dimensional space. Support vector machine
17. Neural networks and deep learning
18. Logistic regression.
19. Gradient descent.
20. Neural networks and the gradient backpropagation algorithm.
21. Deep learning, convolutions and pooling .
22. Regression analysis
23. Linear regression.
24. Polynomial regression.
25. Bias and dispersion.
26. Ridge regression.
27. Ensemble methods
28. Vote.
29. Bootstrapping .
30. Boosting , adaptive boosting , gradient boosting .
31. Stochastic search
32. Monte Carlo search.
33. Simulated annealing algorithm.
34. Genetic algorithm

Monitoring students' independent work

Independent work is monitored by the teacher in the classroom.

The following types of control are provided: interview, assessment of laboratory work.

Detailed criteria for assessing competencies are provided in the Fund of Assessment Tools for Conducting Current and Interim Assessments.

List of literature for completing independent work

List of key literature:

1. Tatarnikova T. M. Intelligent data analysis: textbook / T. M. Tatarnikova. - Moscow, Vologda: Infra-Engineering, 2024. - 172 p. - ISBN 978-5-9729-1772-3. - Text: electronic // Digital educational resource IPR SMART: [site]. - URL: <https://www.iprbookshop.ru/143351.html>
2. Stepashkina A. S. Numerical methods and machine learning in metrology: textbook / A. S. Stepashkina. - Moscow, Vologda: Infra-Engineering, 2024. - 144 p. - ISBN 978-5-9729-1954-3. - Text: electronic // Digital educational resource IPR SMART: [site]. - URL: <https://www.iprbookshop.ru/143592.html>

List of additional literature:

1. Vasiliev E. P. Intelligent data analysis in decision-making technologies: a tutorial / E. P. Vasiliev, V. I. Oreshkov. - Ryazan: Ryazan State Radio Engineering University, 2023. - 180 p. - ISBN 978-5-7722-0344-6. - Text: electronic // Digital educational resource IPR SMART: [site]. - URL: <https://www.iprbookshop.ru/134854.html>

2. Zamyatin A. V. Intelligent data analysis: a tutorial / A. V. Zamyatin. - Tomsk: Publishing House of Tomsk State University, 2020. - 194 p. - ISBN 978-5-94621-898-6. - Text: electronic // Digital educational resource IPR SMART: [site]. - URL: <https://www.iprbookshop.ru/116889.html>

3. Palmov S. V. Intelligent data analysis: a tutorial / S. V. Palmov. - Samara: Volga Region State University of Telecommunications and Informatics, 2017. - 127 p. - Text: electronic // Digital educational resource IPR SMART: [website]. - URL: <https://www.iprbookshop.ru/75376.html>

List of resources of the information and telecommunications network "Internet" necessary for mastering the discipline

- <https://www.kaggle.com/> - an online platform for data science projects
- UCI Machine Learning Repository - a repository of datasets for machine learning - <http://archive.ics.uci.edu/ml/>
- Machine Learning Course <https://habr.com/company/ods/blog/322626/>
- A resource dedicated to machine learning, pattern recognition, and data mining. - <http://machinelearning.ru>