Machine Learning

Lab 2. Basic regression and classification

The aim of the exercise is to gain practical familiarity with several machine learning models - linear regression, logistic regression, and support vector machines. The exercise is scheduled for 3x45 minutes.

Regression vs classification

Regression and classification are two of the main tasks that learning systems face. We saw an example of classification in the first lab, where the task was to predict the species of a flower based on several of its features. Generalizing this example, classification involves finding the class of an object - which is a categorical variable, based on a set of features that are real-valued.

Regression differs from classification in that we predict a continuous variable based on a set of other variables. Within the example of iris flowers, regression could involve predicting the width of the sepals based on the other features of the flower. Sepal width is not a categorical or discrete variable - it is a continuous variable that takes positive real values.

Linear regression

Now let's take a look at the simplest regression model, which is linear regression. We will try to perform the task mentioned above - predict the width of the sepals based on their length, for a selected species of iris. This is the simplest possible case – the training set consists of one **independent variable (predictor)**, based on which we predict the value of one **dependent variable**. In this case, the independent variable is the sepal length, and the dependent variable is sepal width. The training set consists of the corresponding pairs of these variables. The goal is to fit a linear function optimally, which will allow us to predict the width of the sepals based on their length. There are also other types of regression, such as multiple regression, where more independent variables are involved in the prediction process, and multiple dependent variables are predicted in multivariate regression. The linearity of this model, on the other hand, is based on the fact that the hypothesis function is linear:

$$h(x) = w_0 + w_1 x$$

where h(x) is a hypothesis that returns the value of the dependent variable, w_0 and w_1 are parameters (also often referred to as weights, especially in the case of neural networks), and x is the independent variable. This formula can be written in vector form, which takes into account multiple regression with n independent variables:

$$h(x) = \sum_{i=0}^{n} w_i x_i = \boldsymbol{w}^T \boldsymbol{x}$$

where **w** is a vector of parameters and **x** is a vector of variables. T denotes the transpose of a vector, which swaps its columns and rows - this allows for proper multiplication of the **w** and **x** vectors. To ensure that the above formula develops correctly, we also need to include a "virtual" variable $x_0 = 1$,

which is multiplied by the weight w_0. The latter is sometimes called the bias and is equivalent to the intercept of a linear function.

To perform the regression task described above, we first need to prepare and examine a training and test set.

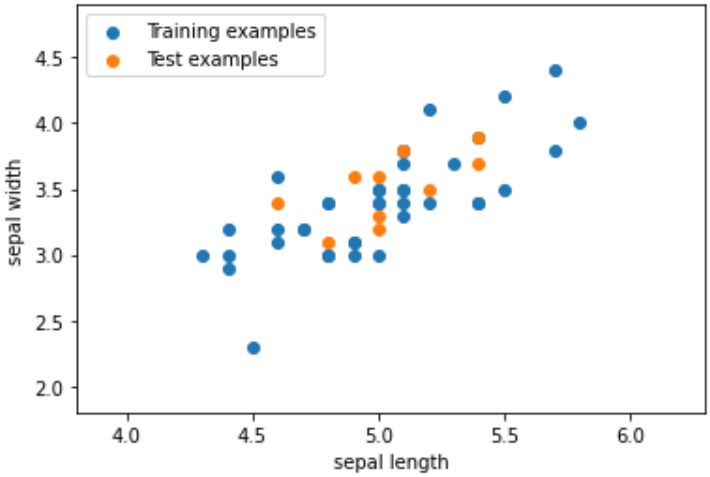

Figure 1. Visualisation of the dataset. Two features are considered – sepal length is chosen to be the independent variable and sepal width the dependent variable

```
from sklearn import datasets
import matplotlib.pyplot as plt
import numpy as np

iris = datasets.load_iris()
X = iris.data
y = iris.target

# take class 0 (iris setosa) samples only and first two features

X = X[y==0][:,0:2]
y = y[y==0]

# random sample ordering
random0 = np.random.choice(np.arange(0,50),50,replace=False)

# sample indices for the training and test sets
train_inds = random0[:40]
test_inds = random0[40:]

# feature value X and Y min-max ranges
x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5

# the plot
fig, ax = plt.subplots()
scatter = ax.scatter(X[train_inds, 0], X[train_inds, 1])
scatter = ax.scatter(X[test_inds, 0], X[test_inds, 1])
plt.xlabel("sepal length")
plt.ylabel("sepal width")
plt.legend(["Training examples", "Test examples"])
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
```

It is already apparent to the naked eye that there is a correlation between the length and width of circles, and the samples form a cloud of points that align in an elongated line. We will calculate the parameters of a linear function that minimize the mean squared error between this function and the values of the dependent variables in the training set.

```
from sklearn import linear_model
regr = linear_model.LinearRegression()
regr.fit(X[train_inds,0:1],X[train_inds,1:])

# parameters of the regression line: slope and intercept
print(regr.coef_, regr.intercept_)

# draw the regression line
line = np.arange(x_min,x_max+1)*regr.coef_[0] + regr.intercept_[0]
plt.plot(np.arange(x_min,x_max+1),line,'r')
```

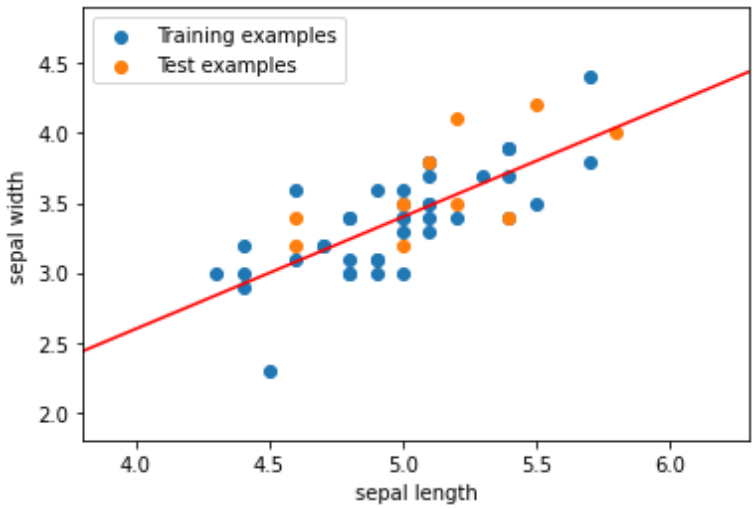

Figure 2. A linear fit (red line) to the training data (blue dots)

The learning algorithm found the parameters of the line visible in the figure. As we can see, the test samples marked in orange are located at a certain distance from the line. This means that there is a regression error - the line predicts a value that was different in reality.

Task 1: Calculate the total mean squared error between the predictions of the width of the sepals learned by the regression model and the actual values of this feature - for the training set and separately for the test set. Use the .predict() method to obtain predictions from the trained regression model.

Task 2: Modify the above example to predict the width of sepals based on two other features - the length of sepals (0) and the length of petals (2). To do this, fill in the multiple_regression.py program in places marked with: # !!!. Note the values of the learned parameters in the report and include the obtained graph.

The objective function and learning algorithms

How does automated finding of linear regression parameters representing the relationship between length and width of sepals actually work? Two important elements are involved in this task: the learning algorithm and the objective function (also known as the loss function or cost function). The

term "learning algorithm" essentially covers the entire "recipe" for the selected learning method or model. Its component, however, is the optimization algorithm, which operates on the objective function. The objective function measures how good the current model parameters are, updated during the learning process. One of the most popular objective functions is the mean squared error (MSE). In the case discussed above, MSE is calculated between the model predictions and the actual target values of sepal width.

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2$$

In the above formula, the upper indices (i) denote consecutive training examples, of which there are m. Therefore, we see that MSE is simply the difference between the prediction and the target for each training example, squared and then averaged over all examples. The 1/2 multiplier before the error is added here to simplify further calculations associated with the loss function, whose details will be presented in another exercise.

Learning largely involves optimizing the loss function. Often, this means minimizing its value, i.e., the model's errors. Optimization is performed using optimization algorithms, one of the most common of which is the gradient descent algorithm. We will also take a closer look at this algorithm in another exercise.

Logistic regression

Logistic regression, despite its name, is used for classification, as we saw in the first exercise. It is one of the most popular and widely used linear classifiers. Binary classification can be used to answer various questions, such as whether a given message is spam or not, whether an online transaction is legitimate or not, or whether a tumour is malignant or benign. However, we will stick to the example of the Iris dataset, reinterpreting the results obtained in the first exercise and tracing the difference between linear and logistic regression. The hypothesis in one-variable linear regression, $h(x) = w_0 + w_1 x$, allows for obtaining h(x) > 1 and h(x) < 0. In the case of binary classification, where we usually assume that the class labels are 0 and 1, such a linear model can cause problems. We want the classifier to always return a discrete value - the class label. Logistic regression allows us to bypass this problem and always obtain $0 \leq h(x) \leq 1$. This is achieved by using the standard logistic function, whose graph has a sigmoidal shape, passes through the point (0, 0.5), and whose range of values is contained in the interval (0, 1):

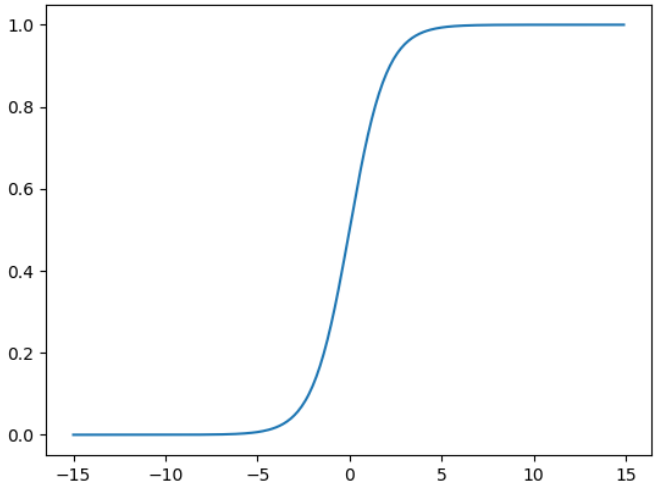

Figure 3. Graph of the logistic function

In logistic regression, the output of the linear model becomes the argument of the logistic function:

$$h(x) = g(\boldsymbol{w}^T\boldsymbol{x}) = \frac{1}{1+e^{-w^T x}}$$

where g(.) is the logistic function:

$$g(z) = \frac{1}{1+e^{-z}}$$

As can be seen, the hypothesis above now returns values in the range (0,1), but they are still continuous values, not discrete classes. However, the key is the interpretation of the values of the hypothesis function: we interpret it as the probability that the sample belongs to class 1. If h(x)=0.9, it means that the probability is 90% for class 1 and 10% for class 0. We can assume that the classifier indicates class 1 when $h(x) \geq 0.5$, and class when 0 $h(x) < 0.5$. Observing the logistic function graph, it is easy to see that for $\boldsymbol{w}^T\boldsymbol{x} < 0$, the classifier will return class 0, and for $\boldsymbol{w}^T\boldsymbol{x} \geq 0$ – class 1.

We now return to the concept of the decision boundary. In the figure below, there is an example from the previous exercise where a classifier was trained to distinguish *iris setosa* from *iris versicolor*. The question arises, where do the parameters of the visible linear decision boundary come from?

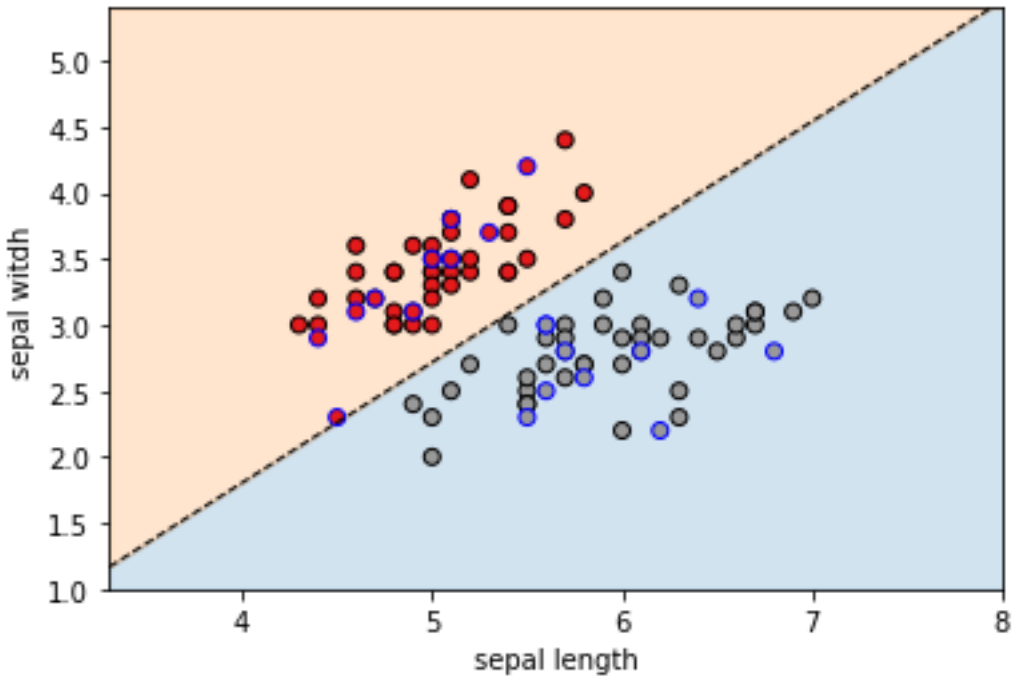

Figure 4. Decision boundary for logistic regression. Binary classification (iris setosa and versicolor)

The decision boundaries observed in the previous exercise do not take into account the logistic function; they are only related to its argument: $\boldsymbol{w}^T\boldsymbol{x}$. In this example, the decision boundary is the function $x_2 = f(x_1)$, where x_1 is the sepal length and x_2 is the sepal width.

Task 3: Draw a necessary conclusion from the information that for $\boldsymbol{w}^T\boldsymbol{x} < 0$ the classifier returns class 0, and for $\boldsymbol{w}^T\boldsymbol{x} \geq 0$ class 1. Then calculate (using variables) the parameters of the decision boundary. Write down the obtained formulas (and possibly their derivation).

Note that the result of these calculations was used in the code in the previous exercise to find the formula for the decision boundary. It is worth mentioning that the decision boundary in logistic regression does not have to be linear. However, this requires the introduction of higher-order factors, such as x12, and additional parameters, into the model.

Task 4: Expand the logistic_regression.py program by adding the calculation of the probability with which the test samples were assigned to classes. Use the .predict_proba() method, which can be called on an instance of the LogisticRegression() object, for this purpose. Paste the probabilities for successive samples, the class assigned by the classifier, and the actual class into the report.

Support vector machines

Another popular classification algorithm is Support Vector Machines (SVMs). To understand the essence of this type of classification, consider the case shown in the figure.

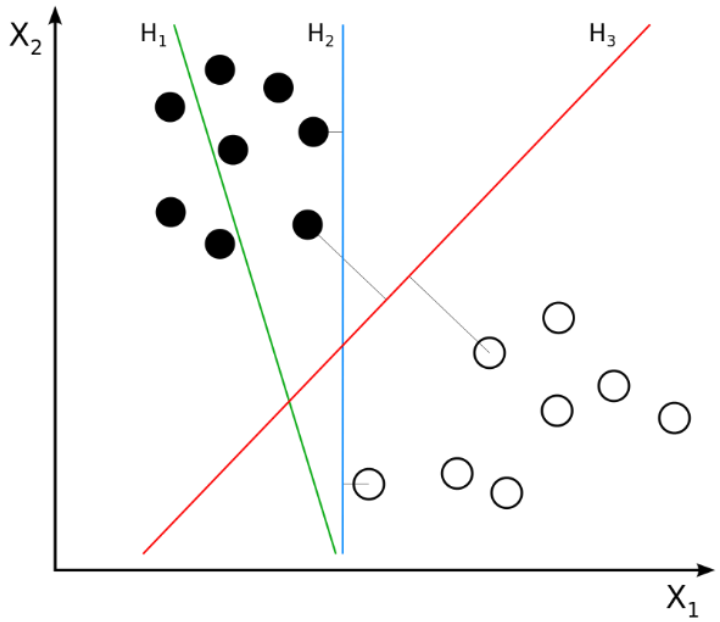

Figure 5. Three of infinitely many possible decision boundaries separating two abstract classes (white and black dots). By User: ZackWeinberg, based on PNG version by User: Cyc - This file was derived from: Svm separating hyperplanes.png, CC BY-SA 3.0

Three decision boundaries, H1, H2, and H3, are visible in the figure. It is easy to see which one "best" separates the samples of the two classes (black and white dots). Although both H2 and H3 separate the samples perfectly, it is easy to identify H3 as better suited to the data set. Remember that the trained system is supposed to be used for the classification of new samples in the future, whose location in space will be different from the samples in the training set. Therefore, the classifier should have good **generalization** ability - correct classification on new samples that were not used (and may not even be available) during training.

The hypothesis in the case of support vector machines looks like the logistic regression:

$$h(x) = f(\boldsymbol{w}^T \boldsymbol{x})$$

This time, however, the function f(.) is not a logistic function, but a perceptron function, in which the threshold is present:

$$h(x) = f(\boldsymbol{w}^T\boldsymbol{x}) = \begin{cases} 1 \; dla \; \boldsymbol{w}^T\boldsymbol{x} \geq 0 \\ -1 \; dla \; \boldsymbol{w}^T\boldsymbol{x} < 0 \end{cases}$$

So in the hypothesis, we have two differences: firstly, the course of the f(.) function is not sigmoidal, it is discontinuous and contains a sudden jump from -1 to 1, and secondly, the values representing the classes are -1 and 1 instead of 0 and 1 - this is a choice that simply facilitates further calculations. There are also differences in the objective function.

Support vector machines introduce the concept of margin. The geometric margin is the minimum distance of a sample from the decision boundary. To calculate it, one must calculate the length of the vector perpendicular to the decision boundary, such that the sample must be moved by the same length to reach the decision boundary. Support vector machines allow finding the decision boundary for which the margin is the largest, taking into account all samples from both classes. Therefore, they prefer decision boundaries such as H3 in the above figure, rather than such as H2 - which does not run farthest from examples of different classes. The support vectors are those data samples through which lines perpendicular to the decision boundary pass and that lie at a distance from the margin. These lines are somehow "supported" by the support points-vectors, hence the name. The figure below shows the example results of the described algorithm.

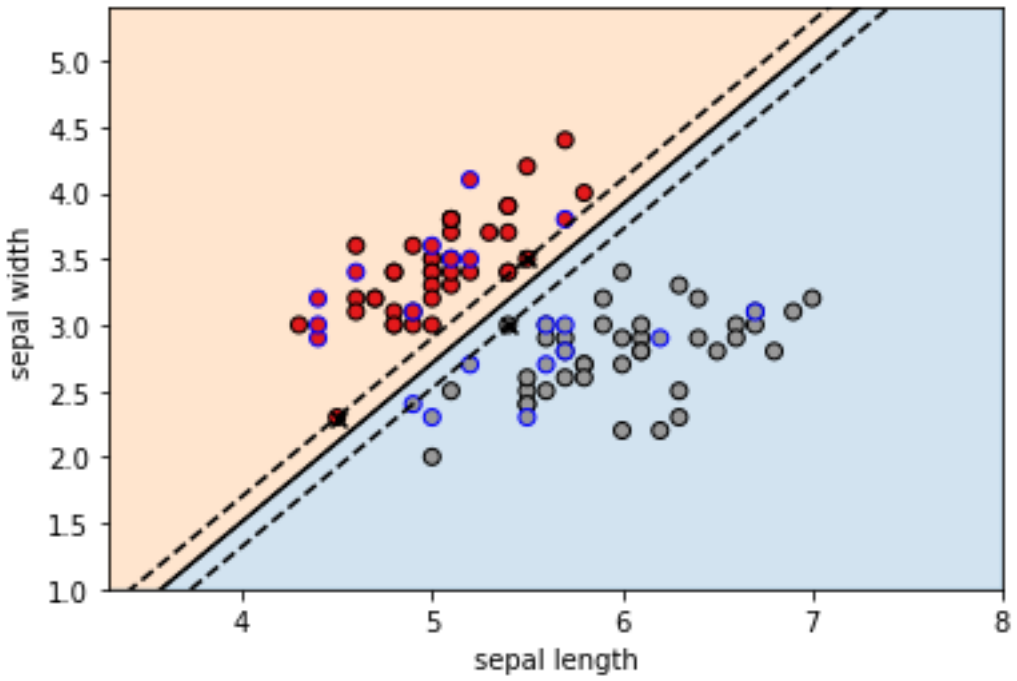

Figure 6. Support Vector Machines - an example of linearly separable data. The black line is the optimal decision boundary, the dashed lines are the lines indicating the margins. Support vectors are samples marked with black crosses that "support" margins. There are no training set samples inside the margins. Samples surrounded by blue borders belong to the test set (do not participate in searching for the decision boundary).

More about support vector machines, their mathematics and possibilities of their application for separating classes that are not linearly separable can be found here and here (Polish).

Task 5: add support vector machine classification to the previous logistic regression program (for the same training and test set). The appropriate function is SVC from the sklearn.svm module. When defining the classifier object, provide parameters: kernel='linear', C=1E10.

Caution: We set the kernel to 'linear' because we are looking for a linear decision boundary; the parameter C concerns the so-called regularization, which counteracts the phenomenon of overfitting - excessive adjustment of model parameters to training data. Save the graphs and draw your conclusions.

Support Vector Machines - an example of non-linearly separable data

The program nonlinear_machines.py contains an example of using Support Vector Machines for the classification of non-linearly separable data. This means that it is not possible to find a straight line that separates the class samples. In the example, we still consider binary classification and only two features. Moving from a linear to a non-linear decision boundary requires the use of the so-called kernel trick. The kernel is a function that transforms the feature space. In the transformed space, a linear decision function is fitted. After the inverse transformation to the original feature space, the linear boundary becomes non-linear.

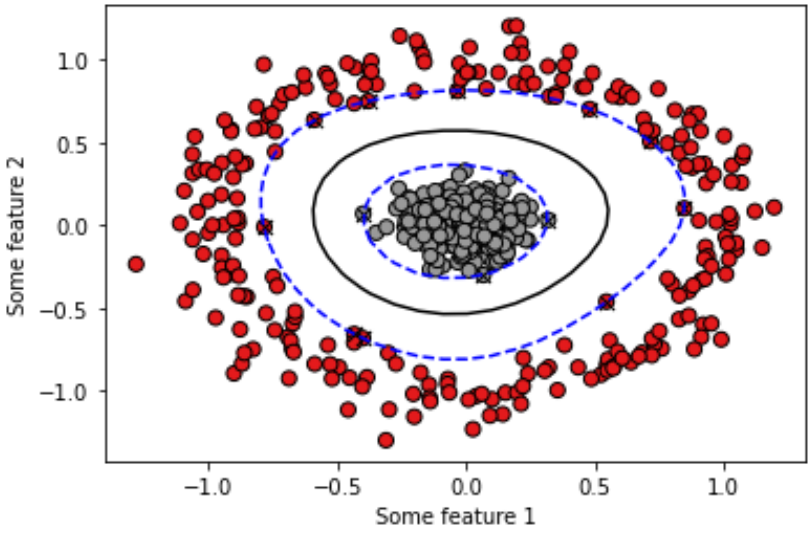

Figure 7. Expected output of the nonlinear_machines.py script, showing a nonlinear classification example using support vector machines

Task 6: Run, analyze, and try to understand the code of nonlinear_machines.py.

This instruction used:

Materials of AGH (Polish)

Materials of the University of Warsaw (Polish)

Machine Learning — Andrew Ng, Stanford University at Coursera, Lectures 6.1-6.3 (STRONGLY RECOMMENDED TO WATCH THIS COURSE ON BASIC MACHINE LEARNING)

This instruction was written in Polish by Jakub Jurek, then translated by ChatGPT.