# Text Classification

The stages of working in the text Classification:

1. **Import necessary libraries** (e.g. pandas, Numpy, sklearn, matplotlib).
2. **Data processing:** Data extraction, deleting values ​​that are not useful to us and can create a mess for the models, Data reorganization, Data distribution for training and testing.
3. **Vectorizing:** Converting the data frame from text to numbers.
4. **Modeling:** Choosing the model, we will use for the classification task, and implementing it in the code.
5. **Confession matrix:** After training the model and examining it, we will build a matrix that tells us how many times the model was right or wrong in the prediction, and according to this we will know the level of our model in relation to the task.
6. **Test:** We will put a random input into the model that is similar to the data it learned, and we will see if the model manages to correctly classify the input.

**Import necessary libraries**

For the data manipulations we used in pandas and Numpy libraries.

* Numpy: "NumPy" can be used to perform a wide variety of mathematical operations on arrays.
* Pandas: "Pandas" as an open-source software library built on top of Python specifically for data manipulation and analysis.

We used the "sklearn" library to perform some tasks from the project such as: Vectorization, modeling, dividing the data, and applying the confession matrix.

* Sklearn: The "sklearn" library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.

In order to see graphs at each stage of the model learning process, we used Matplotlib.

* Matplotlib: "Matplotlib" is a comprehensive library for creating static, animated, and interactive visualizations in Python.

**Data**

I used the data project from the Kaggle site.

The data is 1.6 million tweets from Twitter, each tweet is classified according to sentiment, if the tweet is negative its classification is "0", if the tweet is positive its classification is "4".

The database is divided into six columns: 'target', 'id', 'date', 'flag', 'user', 'text'.

**'target':** In the 'target' column there are two classes: "0" for a negative tweet, "4" for a positive tweet. this column will represent vector Y in the learning process.

**'id':** In the 'id' column there are the id of the tweet. In the training and classification process we did not use this information.

**'date':** In the 'date' column there are the date of the tweet. In the training and classification process we did not use this information.

**'flag':** The query (lyx). If there is no query, then this value is NO\_QUERY. In the training and classification process we did not use this information.

**'user':** In the 'date' column there are the user that tweeted. In the training and classification process we did not use this information.

**'text':** In the 'text' column there are the text of the tweet. This column will represent vector X in the training and classification process.

**Clean Data**

In order to train the model in the best way, we would like to handle the data we put into the model as well as possible, all the "background noise" that does not contribute to the process will be removed.

For this we will have to refer to five things in the text:

1. Change all uppercase letters to lowercase.

For example, in English every first letter in a sentence is an uppercase letter and all the rest are lowercase letters, so it is better to change only the uppercase letters for a lower calculation cost.

1. There are sentences that contain all kinds of "background noise" such as: parentheses, dots, exclamation marks, numbers and more.

The solution to these "background noises" is to replace them all with an empty string " ".

1. root word (stemming): For example: loves, loving.

If we leave these two words it will harm the model, in that the number of features will increase and the result will be a decrease in the function of our model.

The solution in this case is to take the basis of these two words which is: LOV, and call it the "Root Word".

After that, every word with the same base will be replaced by its "Root Word".

1. Removing AB Deviation: We will remove all abbreviations and leave only the complete words.

For example: wasn't = was not.

1. Remove "Stop Word": remove all stop words which are a set of words such as body words and connecting words.

for example: and, are, an, but …

**Vectorizing**

The vectorization step is a very important step in NLP, this step is responsible for converting the text into a number.

We need to convert the text into numbers because the NLP and Machine Learning algorithms do not understand "alpha - beta", but only numbers, and it is best to have 0 or 1 for classification.

The function looks like this: Text Vectorizing (input: 'text', output: number)

The input is a list of words (words = tokens), and the output will be featuring vector.

There are three types of vectorizations methods:

1. "Count-Vectorization".
2. "Ngrams".
3. "TF – IDF".

I will explain a little bit on a "Count-Vectorization".

**"Count-Vectorization"**

Explain with an example.

Our data frame is three sentences.

Sentence 1: I like to play football.

Sentence 2: did you go outside to play tennis.

Sentence 3: john and I play tennis.

From these three sentences we will create a list of tokens. In this list each word will be unique, if a word is found in several sentences in DF, the word will be found in the list only once.

The list of tokens looks like this: ["I", "like", "to", "play", "football", "did", "you", "go", "outside", "tennis", "john", "and"].

After that we will create a new DF to write numbers instead of words, either "0" or "1".

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | "I" | "like" | "to" | "play" | "football" | "did" | "you" | "go" | "Outside" | "tennis" | "john" | "and" |
| Sen1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Sen2 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| Sen3 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

The new DF is called a Document term matrix, and each row of the matrix represents a feature vector of each sentence.

**Modeling + Confession matrix**

I will explain about the confession matrix with the help of an example.  
We have a data table like this:

|  |  |  |  |
| --- | --- | --- | --- |
| Chest pain | Good blood Circ | Blocked Arteries | Weight |
| no | no | no | 125 |
| yes | yes | yes | 180 |
| yes | yes | no | 210 |
| … | … | … | … |

This is a table with patient data, and we want to develop a model that will know how to predict which patient will have a chance of getting heart disease.

|  |
| --- |
| Heart Disease |
| no |
| yes |
| no |
| …. |

Model predict is this:

To do this we have several possible methods to use for the prediction operation:

* Logistic Regression
* K – Nearest Neighbors
* Random Forest

And more…

How do we decide which method is right for us?

First, we divide our data into training data and test data. After that, any method we want to test will learn the data of the training, And finally we will examine all the methods with the test data.

We will then want to test the performance of each method on the test data. One way to examine the performance is with the help of a confession matrix.

A confession matrix looks like this:

Actual

Predicted

|  |  |  |
| --- | --- | --- |
|  | Has Heart disease | Dose Not Have Heart disease |
| Has Heart disease | True Positive | False Positive |
| Dose Not Have Heart disease | False Negatives | True Negatives |

**True Positive:** The number of times the patient had heart disease, and the algorithm also predicted this.

**False Positive:** The number of times the patient did not have heart disease, and the algorithm predicted that he did have heart disease.

**True Negatives:** The number of times the patient did not have heart disease, and the algorithm predicted that he would not have heart disease.

**False Negatives:** The number of times the patient had heart disease, and the algorithm predicted that he would not have heart disease.

The size of the matrix will always be according to the number of things the model has to choose from. In our model, the model has to choose between two things, so the matrix is 2\*2, but if the model has to choose between 3 things, the matrix will be 3\*3, and so on up to n\*n.

At the end of building the matrix of each method, we will compare the number of successes in each method. We will compare only the green squares in the drawing between all methods. The method that has the most success we will use. We will always refer to the diagonal of the matrix in the test, even with the matrix it is in order n\*n.

**TEST**

This is the last part of training the model, in order to implement it we will take data with the same characteristics, but which was not part of the data that the model learned or was tested on, and we will enter it as input to the model and see if the classification was successful or not.

If so, the model works well and nothing needs to be changed.

If not, we will have to return to the learning process and change the commands there and see if the model succeeds in classifying or not, if still not, return to the code again and change until the model succeeds in classifying.