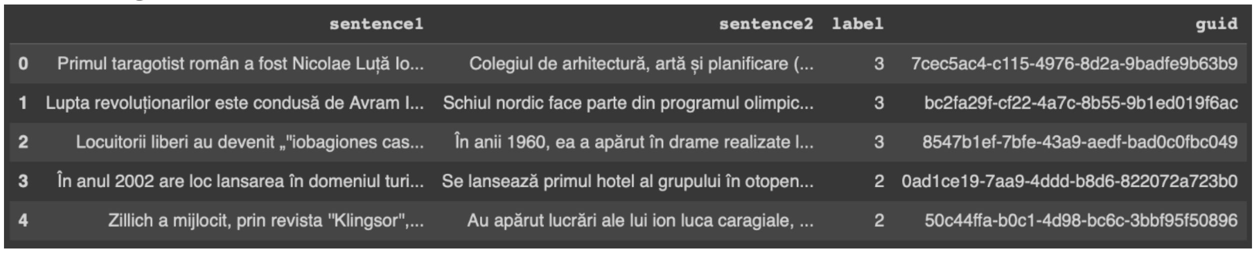
PROJECT 1

Sentence Pair Classification

Dolganiuc Ana

1. Purpose:

   For this project we had to train a model in order to get the best classification of the class for each pair of sentences. The train and validation set have the following structure:



	sentence1	sentence2	label	guid
0	Primul taragotist român a fost Nicolae Luță lo...	Colegiul de arhitectură, artă și planificare (...	3	7cec5ac4-c115-4976-8d2a-9badfe9b63b9
1	Lupta revoluționarilor este condusă de Avram I...	Schiul nordic face parte din programul olimpic...	3	bc2fa29f-cf22-4a7c-8b55-9b1ed019f6ac
2	Locuitorii liberi au devenit „"iobagiones cas...	În anii 1960, ea a apărut în drame realizate l...	3	8547b1ef-7bfe-43a9-aedf-bad0c0fbc049
3	În anul 2002 are loc lansarea în domeniul turi...	Se lansează primul hotel al grupului în otopen...	2	0ad1ce19-7aa9-4ddd-b8d6-822072a723b0
4	Zillich a mijlocit, prin revista "Klingsor",...	Au apărut lucrări ale lui ion luca caragiale, ...	2	50c44ffa-b0c1-4d98-bc6c-3bbf95f50896

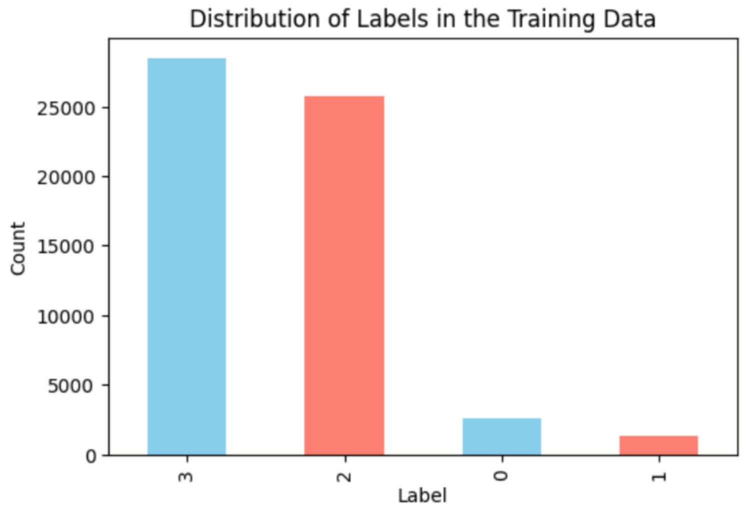
   sentence1, sentence2 – the pair of sentences

   label – the class of each pair of sentences

   guid - the id for each pair of sentences

   For the test set we will have to predict the label.

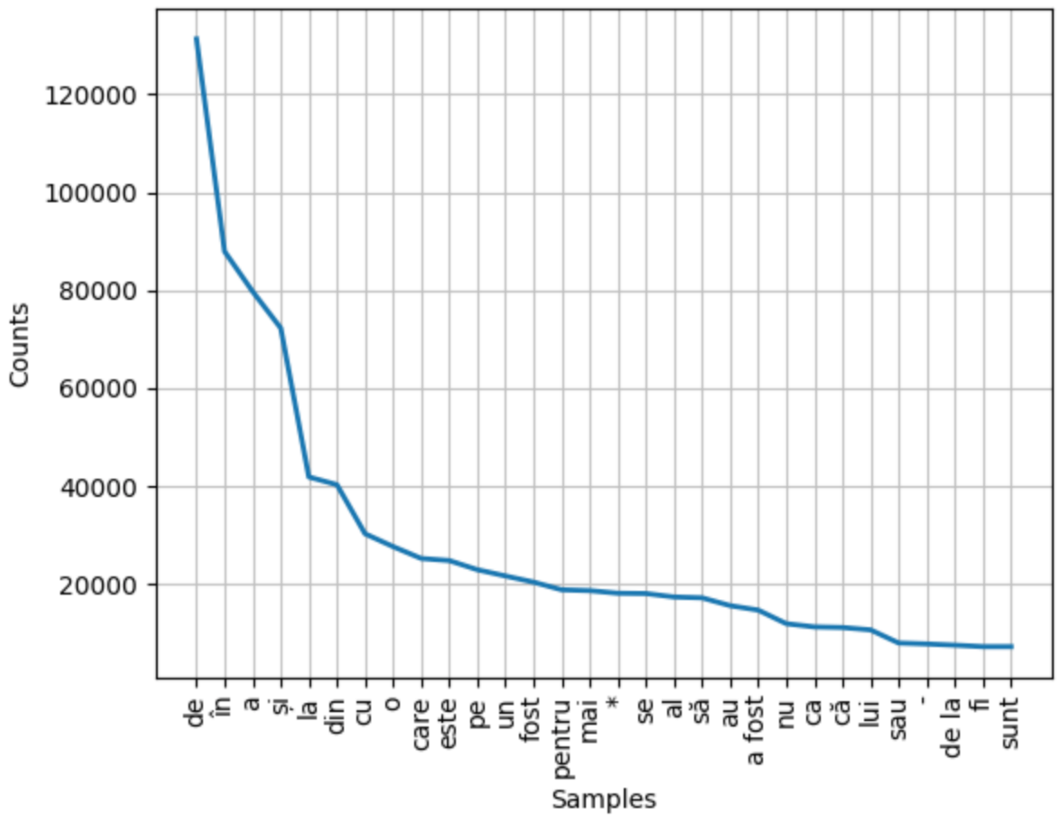
2. Data preprocessing
   a. Size of the data sets: The train set has 58114 entries, the validation set has 3059 entries.
   b. The distribution of the labels in the training data:



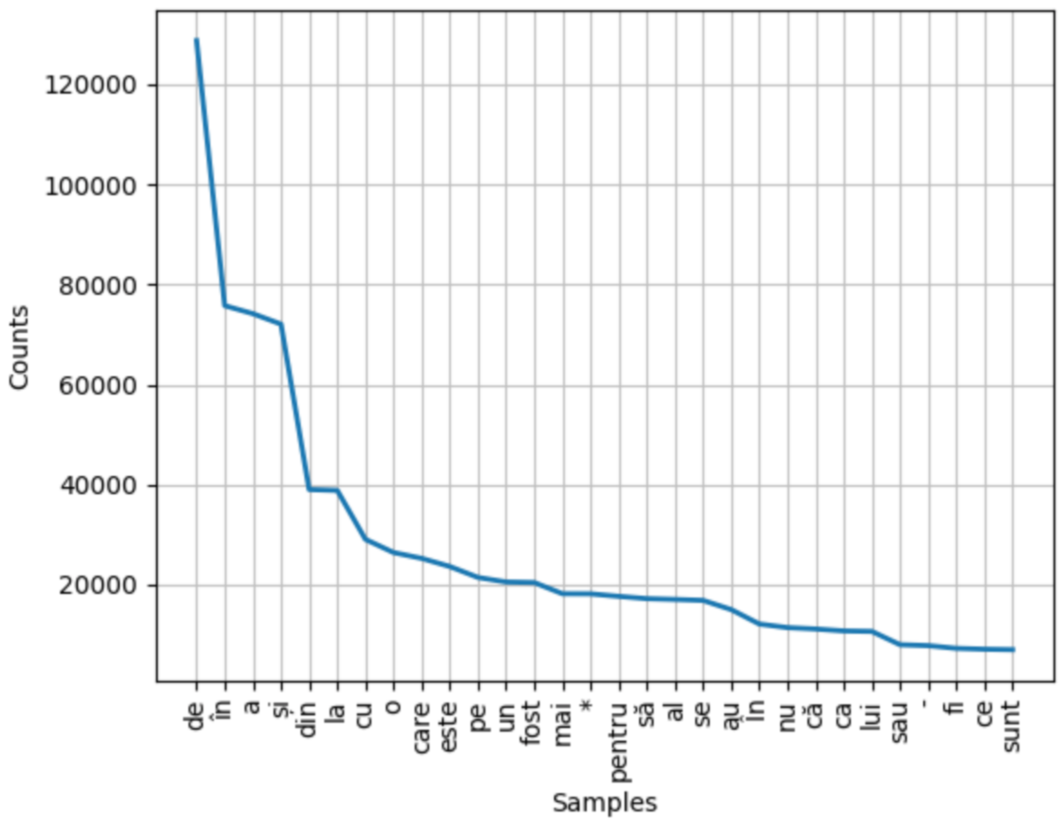
As we can see, there is a class imbalance:
Class 0 =2592 entries
Class 1= 1300 entries
Class 2= 25722 entries
Class 3= 28500 entries

This means we will have to balance the class distribution by oversampling the minority classes (0,1). That will help the model learn the patterns in the minority of classes. I used the oversampling method instead of undersampling, because the dataset is relatively small and applying undersampling could lead to information loss.

c. Tokenization of the data:
   i. Using the BOW tokenizer:
      The resulting vocabulary size: 1.737.998



   ii. Using the TF-IDF Tokenizer:
       The resulting vocabulary size: 368.053



As we can see the listed most common words are slightly different, because the TF-IDF tokenizer assigns weights based on the importance of the words, while the BOW tokenizer counts the words without taking into consideration their importance. For my algorithm, I used the TF-IDF tokenizer because I wanted it to obtain the significance of the words in the database.

d. Custom stop words:

   In order to remove the noise words, reduce dimensionality and improve the interpretability of the model, I used a custom list of stop words. The list was created using the most common words in the dataset, as well as by searching on the internet the most common stop words in Romanian language.

e. Romanian stemmer:

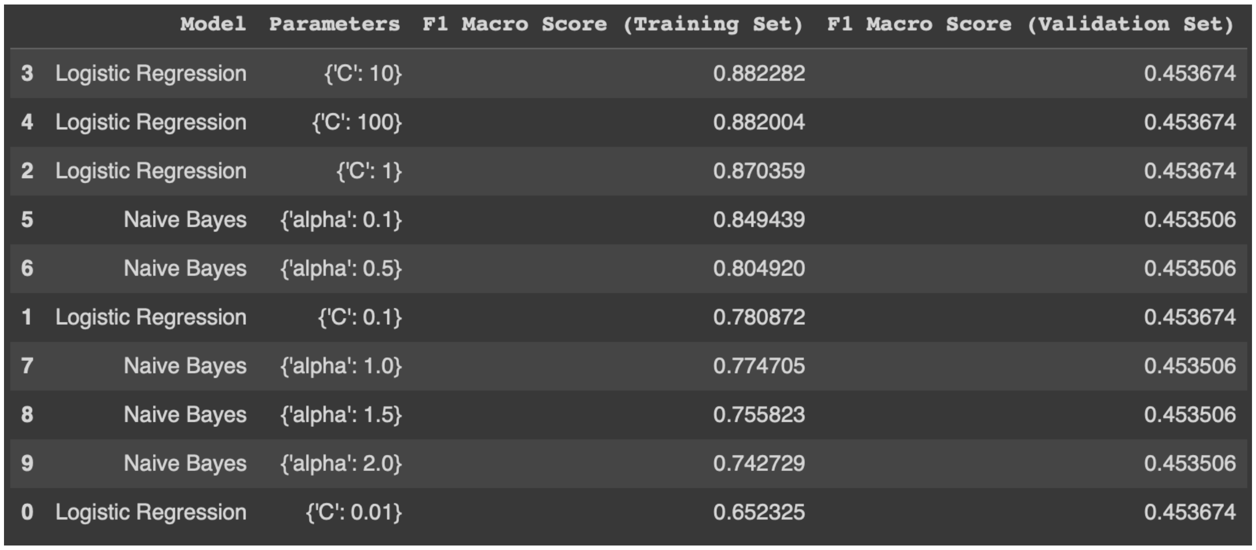
   In order to normalize the text, enhance the text matching and to improve the generalization, I reduced the words to their root form by removing the suffixes and the prefixes from the words. In this way I created my own stemmer for the Romanian language by using some common linguistic rules applicable to the Romanian language.

3. Model Selection and hyperparameter tunning:

   For modelling and data training I used two models:

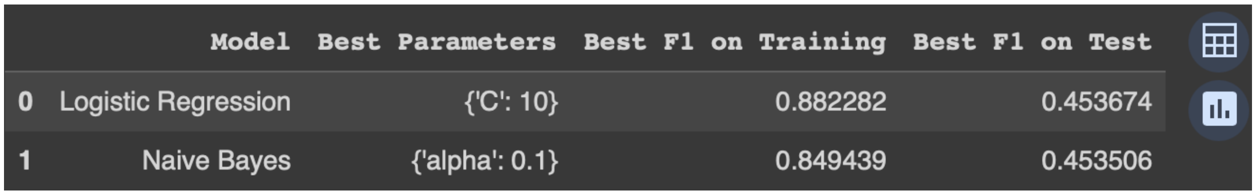
   a. Logistic Regression- it is a good model for text classification because it works well on sparse data, supports multiclass classification, allows regularization techniques and is computationally efficient.
   b. Naive Bayes- handles well small sets of data, handles high-dimensional data and is not sensitive to the specific characteristics of nlp.

After analyzing the performance of both models on the training and validation sets, these were the results:



	Model	Parameters	F1 Macro Score (Training Set)	F1 Macro Score (Validation Set)
3	Logistic Regression	{'C': 10}	0.882282	0.453674
4	Logistic Regression	{'C': 100}	0.882004	0.453674
2	Logistic Regression	{'C': 1}	0.870359	0.453674
5	Naive Bayes	{'alpha': 0.1}	0.849439	0.453506
6	Naive Bayes	{'alpha': 0.5}	0.804920	0.453506
1	Logistic Regression	{'C': 0.1}	0.780872	0.453674
7	Naive Bayes	{'alpha': 1.0}	0.774705	0.453506
8	Naive Bayes	{'alpha': 1.5}	0.755823	0.453506
9	Naive Bayes	{'alpha': 2.0}	0.742729	0.453506
0	Logistic Regression	{'C': 0.01}	0.652325	0.453674

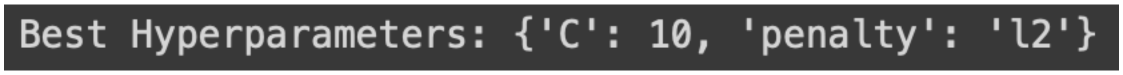
As we can see, the best result was given by the Logistic Regression with the inverse of regularization strength "C" = 10, giving the accuracy of 0.453674 on the F1 Macro Score. For the Naïve Bayes classifier the biggest accuracy was given by the additive smoothing parameter "alpha"= 0,1, with the accuracy of 0.453506.



	Model	Best Parameters	Best F1 on Training	Best F1 on Test
0	Logistic Regression	{'C': 10}	0.882282	0.453674
1	Naive Bayes	{'alpha': 0.1}	0.849439	0.453506

In this way, I chose to continue with Logistic Regression, given the better accuracy on the validation set.

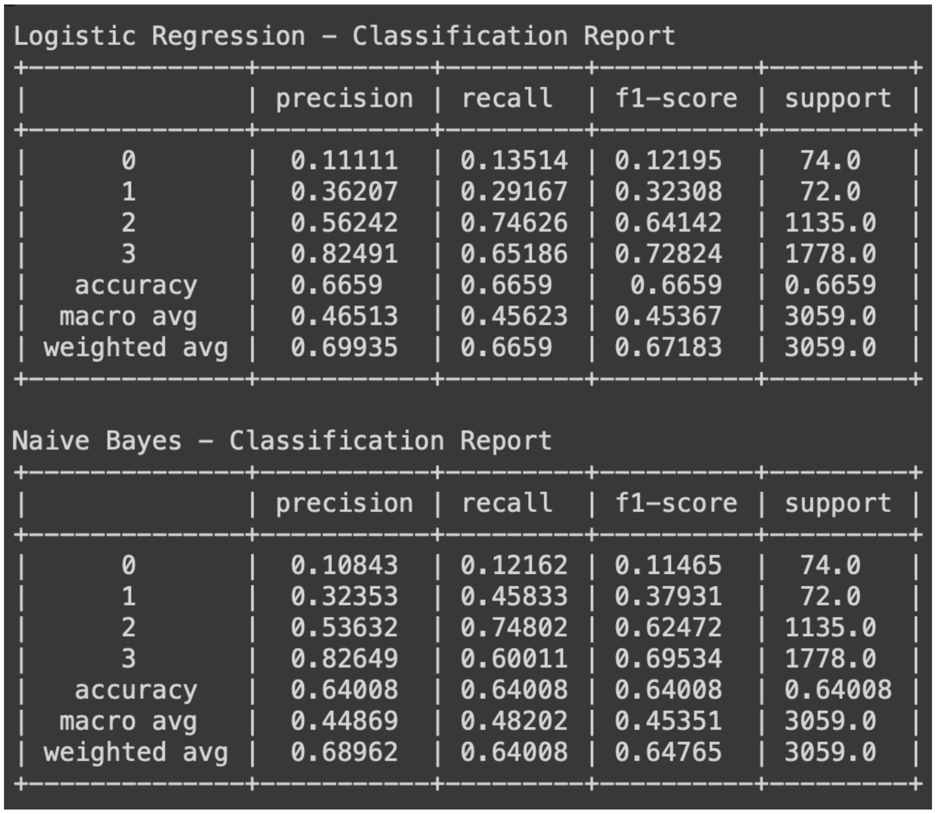
After performing another grid search I found out that the best regularization technique for my model was L2:



```
Best Hyperparameters: {'C': 10, 'penalty': 'l2'}
```

Besides this I used the multiclass LR because the labels where classified in 4 classes. Also, because of the big difference between the training and validation accuracy, I used the Bagging classifier to reduce the variance in the data set and increase the stability of the learning algorithm.

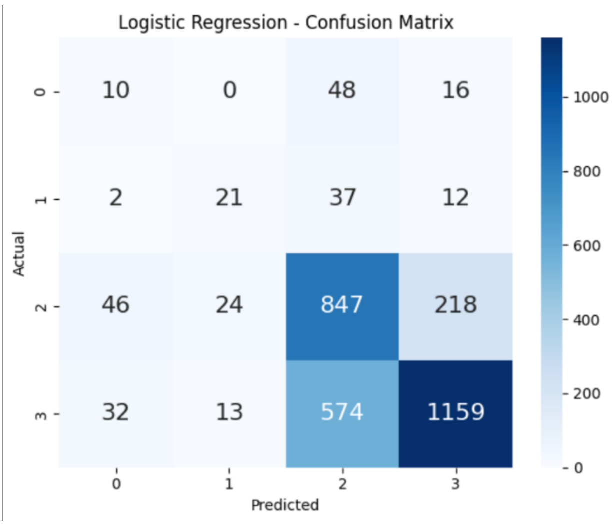
4. Confusion Matrix and classification accuracy rates:

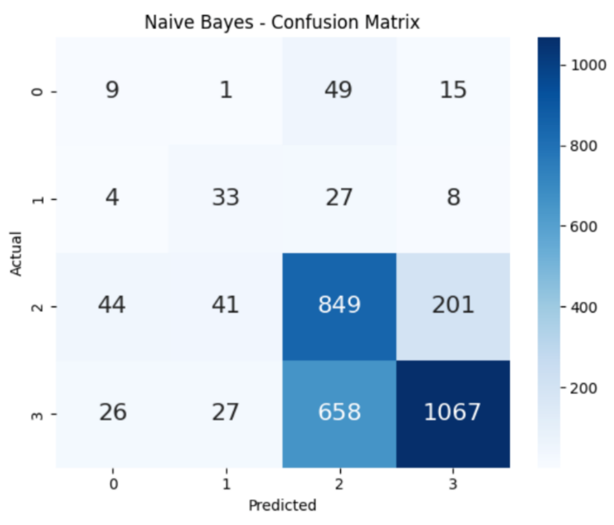


```
Logistic Regression - Classification Report
+--------------+-----------+---------+----------+---------+
|              | precision | recall  | f1-score | support |
+--------------+-----------+---------+----------+---------+
|      0       |  0.11111  | 0.13514 | 0.12195  |  74.0   |
|      1       |  0.36207  | 0.29167 | 0.32308  |  72.0   |
|      2       |  0.56242  | 0.74626 | 0.64142  | 1135.0  |
|      3       |  0.82491  | 0.65186 | 0.72824  | 1778.0  |
|   accuracy   |  0.6659   | 0.6659  |  0.6659  | 0.6659  |
|  macro avg   |  0.46513  | 0.45623 | 0.45367  | 3059.0  |
| weighted avg |  0.69935  | 0.6659  | 0.67183  | 3059.0  |
+--------------+-----------+---------+----------+---------+

Naive Bayes - Classification Report
+--------------+-----------+---------+----------+---------+
|              | precision | recall  | f1-score | support |
+--------------+-----------+---------+----------+---------+
|      0       |  0.10843  | 0.12162 | 0.11465  |  74.0   |
|      1       |  0.32353  | 0.45833 | 0.37931  |  72.0   |
|      2       |  0.53632  | 0.74802 | 0.62472  | 1135.0  |
|      3       |  0.82649  | 0.60011 | 0.69534  | 1778.0  |
|   accuracy   |  0.64008  | 0.64008 | 0.64008  | 0.64008 |
|  macro avg   |  0.44869  | 0.48202 | 0.45351  | 3059.0  |
| weighted avg |  0.68962  | 0.64008 | 0.64765  | 3059.0  |
+--------------+-----------+---------+----------+---------+
```

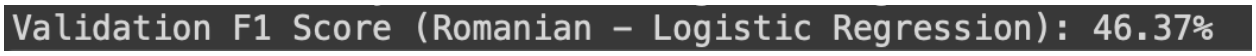
As we can see, the Logistic Regression has a better precision, while Naive Bayes has a better recall. Also, the best accuracy was on the 3rd class due to the bigger weight in the training set.





Looking at the confusion matrix we can conclude that Logistic Regression did better on predicting the 3rd class, while Naïve Bayes did better on predicting the 2nd class.

In conclusion, the best algorithm to use was Multinomial Logistic Regression, using the TF-IDF tokenizer, a custom stop word list and a custom stemmer for the Romanian language. Doing all this steps I managed to get the accuracy of 46,37% on the validation set.



```
Validation F1 Score (Romanian - Logistic Regression): 46.37%
```