Univ. of Athens, Dept of Informatics & Telecoms
YS 19: Deep Learning for NLP
Fall 2022 - Homework 1

Kyriakopoulos Dimitrios - sdi1900093

Notes to the graders: I provide a fully documented ipynb notebook with a link to Google Colaboratory. In order to run the models on the test dataset

1. Go to the "Load the dataset" section (Second code cell of the Notebook).
2. Change `DATA_PATH` value to the path, where the file with the data for training is located.
3. Change `TESTING_DATA_PATH` value to the path, where the file with the data for testing is located.

1 Introduction

In this project, I perform sentiment analysis on IMDb movies' reviews. Firstly, I do data analysis, while I experiment with 2 data preprocessing techniques, which are Bag of Words, TF-IDF and plot my results on them. In addition, I classify the reviews into two classes (Positive and Negative) using logistic regression and I provide a detailed comparison for each method I experimented with.

2 Data Analysis

In this section, I take a deep look at the data providing some useful results. Examining the data I see that there is a balanced distribution as far as the feeling is concerned, as half of them correspond to positive emotion, while the other half correspond to negative emotion (Figure 1).

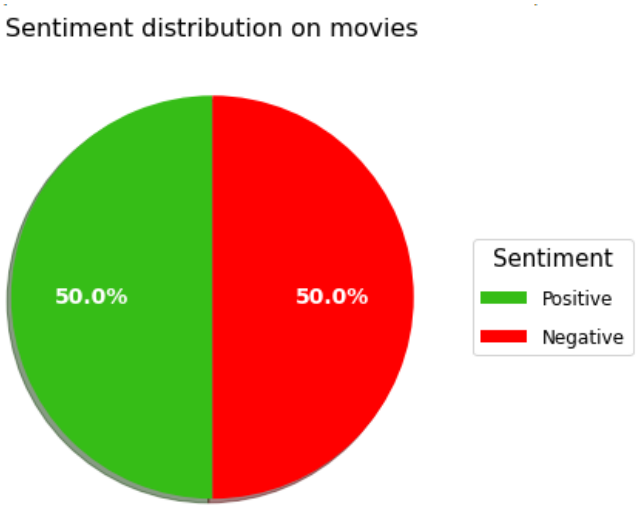

Figure 1: Sentiment Distribution.

In Figure 2 we can see the ten most frequent words and their frequencies among positive and among negative tweets correspondingly.

[11]

	Pos_Common_Word	Frequency
1	film	46521
2	movi	41820
3	one	25685
4	like	18775
5	time	15070
6	good	13841
7	see	13804
8	stori	12924
9	charact	12827
10	make	12625

(a) Most Frequent Words (Positive)

	Neg_Common_Word	Frequency
1	movi	53943
2	film	40824
3	one	24559
4	like	22443
5	make	14812
6	bad	14610
7	even	14112
8	time	13867
9	get	13859
10	watch	13648

(b) Most Frequent Words (Negative)

Figure 2: Most Frequent Words

In addition, I measure the sentiment distribution among the ten movies with the highest number of reviews(Figure 3).

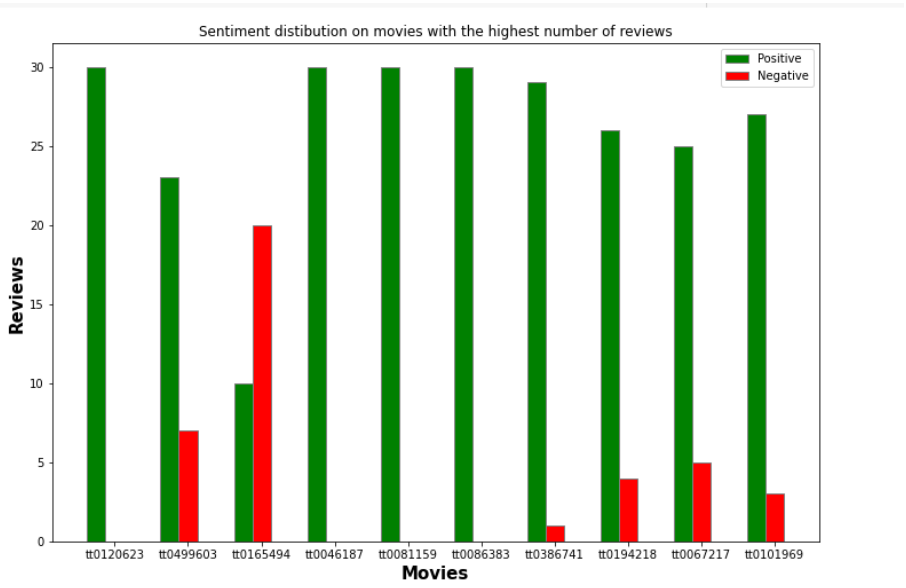

Figure 3: Sentiment Distribution (Movies With Highest Reviews).

3 Text Encoding

In order to convert text into numbers I used two methods of text encoding bag of words and TF-IDF (Term Frequency–Inverse Document Frequency). The bag-of-words model uses the frequency of occurrence of each word as a feature. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. I experiment on the parameter max features in logistic regression parameters. The purpose of max features is to limit the number of features (words) from the dataset for which we want to calculate the scores. This is done by choosing the features based on term frequency across the corpus. I use the model with the default parametres and compare the recall, f1 and precision scores for each number of feature. Below are the results (Figure4, Figure 5) for bag of words and TF-IDF methods correspondingly, which show that 700 and 800 are the ideal max features values for each method.

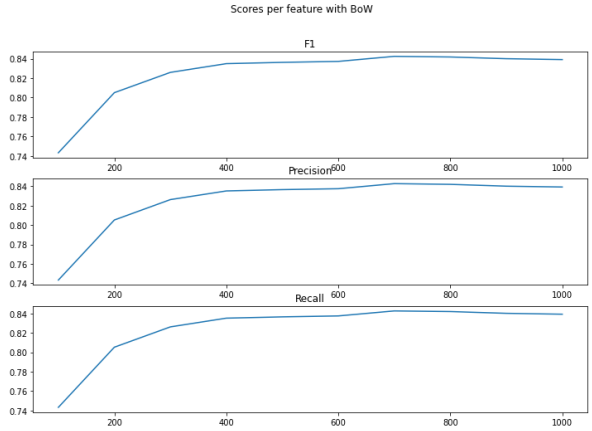

Figure 4: Scores Per Features (BoW).

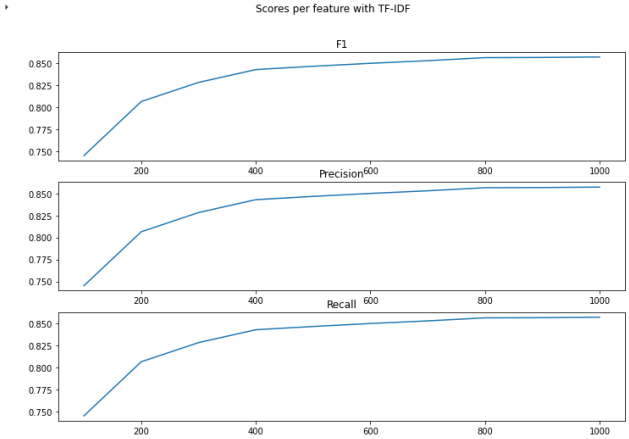

Figure 5: Scores Per Features (TF-IDF).

4 Dataset Split

In order to train and test our logistic regression model we need to split the data into train data and test data. For finding the optimal rate of the data given I experiment with data rate ploting the learning curves, which help us define if our model is underfitting or overfitting. I meassure the misclassification error for each rate which is defined as $1 - Accuracy$. The plots (Figure 5, Figure 6) show that 80% of data is the ideal rate to use. In my model training loss and validation loss are close to each other with validation loss being slightly greater

than the training loss and after some point till the end we can see a pretty flat training and validation loss.

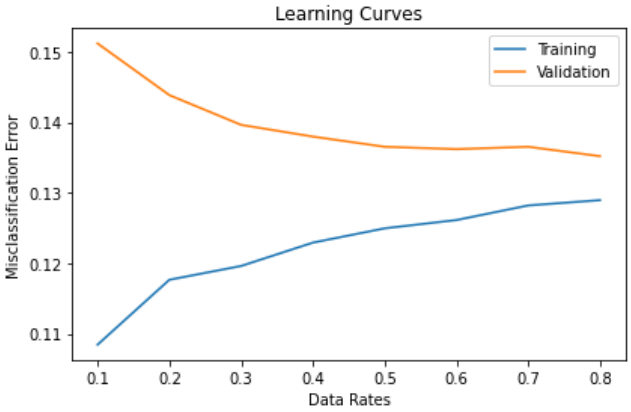

Figure 6: Scores Per Features (TF-IDF).

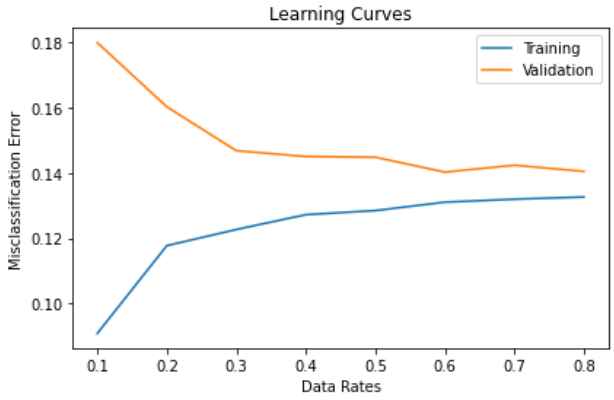

Figure 7: Scores Per Features (TF-IDF).

5 Classification

For the classification I use logistic regression experimenting with the parameters solver, penalty, C.

- **Solver**: The solver I experiment with are **newton-cg, lbfgs, liblinear**. I don't use sag and saga, because the only benefit they have is faster convergence. The data aren't too many and so there is no need for greater speed, while scores are more important to maximize.

- **Penalty**: As penalties I use the values **l1, l2, none**. Penalized logistic regression imposes a penalty to the logistic model for having too many variables. This results in shrinking the coefficients of the less contributive variables toward zero.

- **C**: The C values i used are **0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100**. A high value of C tells the model to give more weight to the training data. A lower value of C will indicate the model to give complexity more weight at the cost of fitting the data

I use GridSearchCV for finding the optimal parameters of Logistic Regression. As scoring metric I use the F1 score. Using grid search with the several combinations of parameters I mentioned I received the warning that lbgfs didn't converge in many cases, which is one of the solver's drawback (it is unsafeguarded, it may not converge to anything). Another factor to take into consideration is scaling, but the most accepted idea is that bag-of-words, Tf-Idf should be left as is mentioned in https://datascience.stackexchange.com/questions/33730/should-i-rescale-tfidf-features. So as to overcome this problem, I increased the parameter max iter, which is the maximum number of iterations taken for the solvers to converge to 500. Grid search result for **bag of words** showed that the best hyperparameters to use for logistic regression are **C: 0.1, penalty: l2, solver: lbfgs**. Although lbgfs uses estimation to the inverse Hessian matrix, which lead to relatively performing the best compared to other methods especially because it saves a lot of memory, it seem that this solver was better that the other solvers in maximizing the F1 score. Grid search result for **TF-IDF** showed that the best hyperparameters to use for logistic regression are **C: 1, penalty: l1, solver: liblinear**. Liblinear is recommended when you have high dimension dataset (recommended for solving large-scale classification problems) as mentioned in https://stackoverflow.com/questions/38640109/logistic-regression-python-solvers-definitions. Furthermore, I use cross validation, to improve model prediction. Above we see one confusion matrix for each encoding method (Figure 8, Figure 9) computed by the prediction on the 20% of the data and the results from cross validation mean scores (Figure 10). We observe that TF-IDF method with the best hyperparameters is slightly better that Bag Of Words.

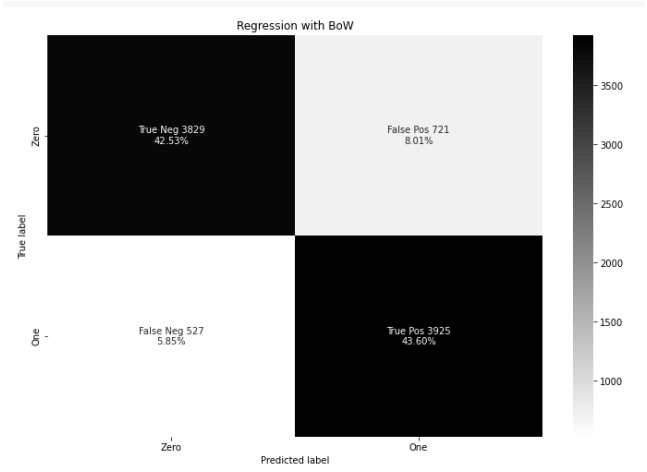

Figure 8: Confusion Matrix (BoW).

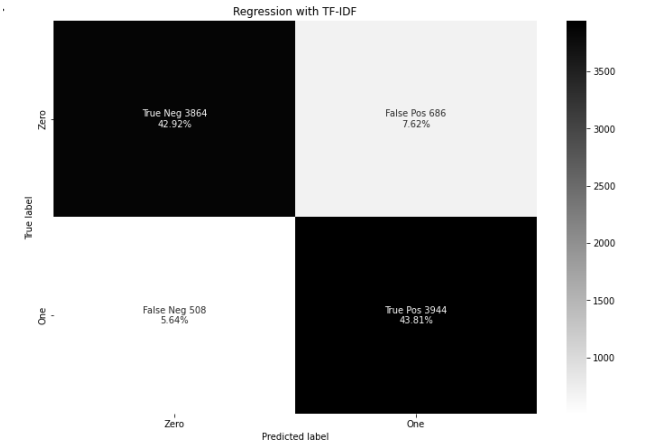

Figure 9: confusion Matrix (TF-IDF).

	Precision	Recall	F1-Score
BoW	84.481274	88.162624	0.849904
TF-IDF	85.183585	88.589398	0.854581

Figure 10: Final Scores