UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *Dimokritos Kolitsos*
sdi: sdi1900085

Course: *Artificial Intelligence II (M138, M226, M262, M325)*
Semester: *Fall Semester 2023*

Contents

1. Abstract

The task involves developing a sentiment classifier for tweets related to the Greek general elections. To tackle this task I will be implementing a logistic regression model. The process will be to first apply basic must have steps like data preprocessing, text cleaning and vectorization before finaly training the model but also experimenting and implementing other techniques and methods to improve the model such as tokenization, lemmatization/stemming, hyperparameter tuning etc. The goal is to experiment and improve the model's performance as much as possible.

2. Data processing and analysis

2.1. Pre-processing

At first I checked just in case there was any missing (NaN) or odd(for example not a label in the labels column) data in the dataset which there were none so I removed that codeblock aswell.

Then I started thinking about text cleaning and what should I exclude from the tweets that is not helpful or needed for the training of the model. I ended up removing special characters, URLs and usernames using regular expressions. This will help the model focus on words that can have an indication of sentiment and context and not on irrelevant words and characters.

The processed text was then tokenized using the Natural Language Toolkit library. Tokenization is necessary as it helps the model understand the structure of the text and allows for techniques like lowercasing (and lemmatization etc) to be applied easily.

All tokens were then converted to lowercase characters to maintain consistency, limit extra characters from needing to be included and understood and prevent possible confusion from the model in general.

Afterwards I also tried applying lemmatization to the text to see if it will improve the specifier's training. I contemplated whether to try stemming, lemmatization or both but since spaCy already has a Greek language model it made my decision easier. Lemmatization involves reducing words to their base or root form, contributing to a more refined and standardized representation of the text, meaning that the the model can focus more on the essence of the words and as with the previous steps, it helps leaving less margin for confusion.

2.2. Analysis

For the analysis of the data I implemented a simple word cloud graph to get a rough idea of how the dataset looks and the also of dominant words in the dataset. Otherwise I already had a rough idea already from printing the dataset each time to check if the preprocessing techniques I implemented were working correctly and also see the difference they made each time.

2.3. Data partitioning for train, test and validation

I found out that repartitioning the dataset and trying out different configurations

from the one already given didn't yield any significant differences in the results so I left the dataset split as given from the beginning

2.4. Vectorization

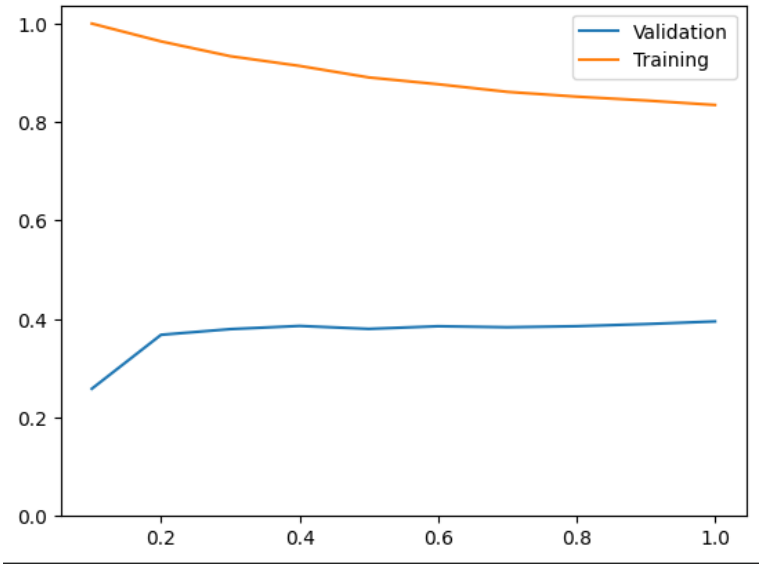
Vectorization is a necessary step in transforming the processed text into a numerical format, ready for the model to use. I chose the TF-IDF (Term Frequency-Inverse Document Frequency) technique for vectorization since besides converting the text to numbers, it takes into account a word's significance and assigns weight to words which can be really helpful for the sentiment classifier and it's training process. The vectorization of the text is allowing the model to learn patterns and relationships within the textual data which is crucial.

3. Algorithms and Experiments

3.1. Experiments

At first, after some basic data exploration I decided to run a training session without any preprocessing except for vectorization which is necessary to see how the model performs with raw text and have a base point to compare my results and see how each new technique and feature I try improve the model. As me and many others though noticed, the results most of the time were not really representative and could not be really used as a measure point as, at least for me, the results with the raw unprocessed text were almost the best I could achieve in terms of numbers. Here are the results of the training of my model only having applied vectorization to the text:

```
F1 Score Train: 1.0
F1 Score Validation: 0.25840293219839866
F1 Score Train: 0.9637664264918252
F1 Score Validation: 0.3679832105413363
F1 Score Train: 0.9337159983448147
F1 Score Validation: 0.37954375160779635
F1 Score Train: 0.9140226570971349
F1 Score Validation: 0.38598852789310145
F1 Score Train: 0.8905950845655513
F1 Score Validation: 0.3800048624434547
F1 Score Train: 0.8768156451934553
F1 Score Validation: 0.3853594374451677
F1 Score Train: 0.8613254669320782
F1 Score Validation: 0.38332330268607595
F1 Score Train: 0.8516335098970931
F1 Score Validation: 0.3854257450311662
F1 Score Train: 0.8437934561180693
F1 Score Validation: 0.38976624381081476
F1 Score Train: 0.8346821867727422
F1 Score Validation: 0.39515109628192335
```

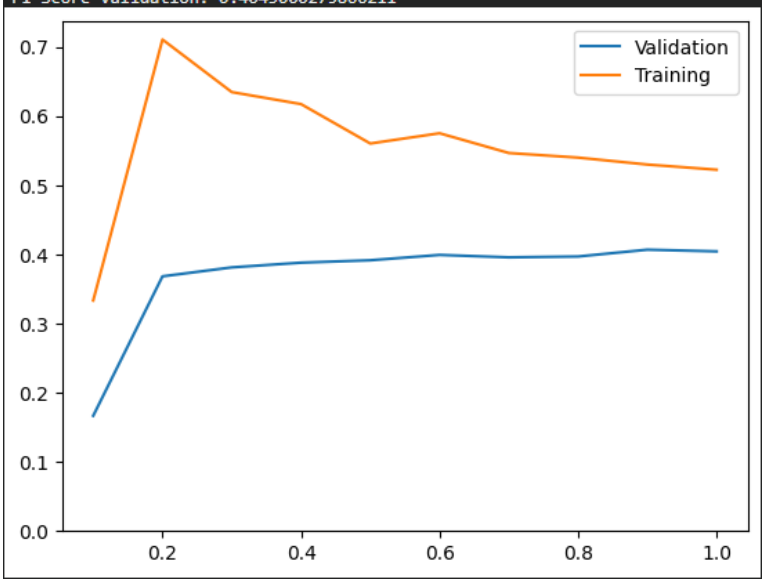



```
Accuracy on the validation set: 0.39

Classification Report:
              precision    recall  f1-score   support

    NEGATIVE       0.38      0.40      0.39      1744
     NEUTRAL       0.40      0.39      0.39      1744
    POSITIVE       0.40      0.39      0.40      1744

    accuracy                           0.39      5232
   macro avg       0.39      0.39      0.39      5232
weighted avg       0.39      0.39      0.39      5232
```

We notice that the results are quite high already if we take into account that (as we'll see later) event after applying various different techniques both to process our data and also after the training (optimization, hyperparameter tuning etc), the numbers in the end won't go much higher than 0.4 accuracy that we already saw here in some cases.

At this point I started experimenting with the various preprocessing methods that most of them were mentioned in the previous section of the report. The problem is I managed to forget to change the text which the vectorization is applied on to the processed text, so I continued on with the post training techniques while the model was still being trained on the unprocessed data. At that time, me being affected by the conversations that for most people too, results would not change much after trying multiple different stuff and the fact that the dataset might have been labeled in almost a random way affecting the results and how the numbers didn't reflect how changes would actually benefit or not the model, made me not give much thought at the fact that I was seeing identical results (and also the fact that each run the results may vary a little bit without anything being changed on the code so there was a slight difference each time I changed something also didn't help) led me to overlook my mistake and move on to try to implement techniques such as hyper-parameter tuning. After experimenting with some parameters (described below), I found some that improved the look of the previous graph that looked like there was overfitting. So with the unprocessed text but better parameters (c=0.1, solver=liblinear) the best the model performed is this:

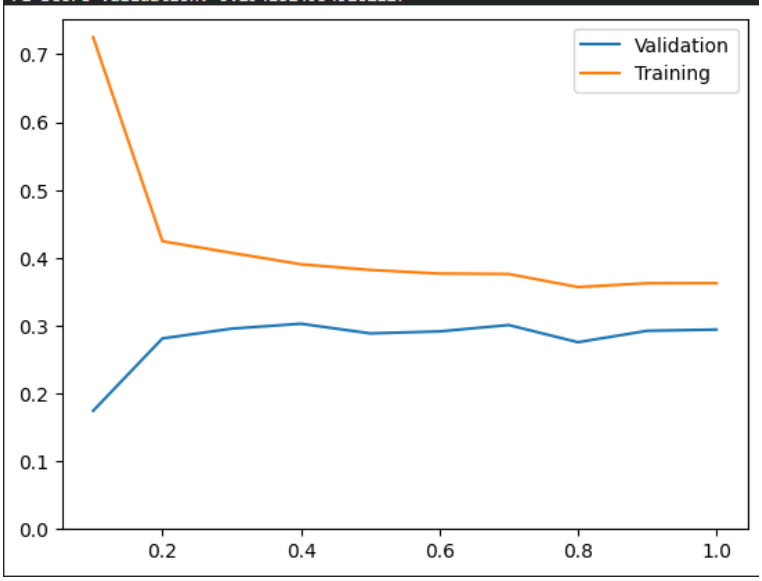
```
F1 Score Train: 0.3333333333333333
F1 Score Validation: 0.16666666666666666
F1 Score Train: 0.7105024638215189
F1 Score Validation: 0.3681956395222702
F1 Score Train: 0.6345575166747595
F1 Score Validation: 0.38112932303996694
F1 Score Train: 0.6171900034963008
F1 Score Validation: 0.3879622183596296
F1 Score Train: 0.5602064958517702
F1 Score Validation: 0.3914355377865426
F1 Score Train: 0.575041025985563
F1 Score Validation: 0.39915933280522636
F1 Score Train: 0.5464895886633201
F1 Score Validation: 0.39566546296891403
F1 Score Train: 0.5398416160348556
F1 Score Validation: 0.3968836224816933
F1 Score Train: 0.529809880145503
F1 Score Validation: 0.4067448780403142
F1 Score Train: 0.5223354653450245
F1 Score Validation: 0.4043000275866211
```




```
Accuracy on the validation set: 0.40

Classification Report:
              precision    recall  f1-score   support

    NEGATIVE       0.39      0.48      0.43      1744
     NEUTRAL       0.41      0.39      0.40      1744
    POSITIVE       0.43      0.34      0.38      1744

    accuracy                           0.40      5232
   macro avg       0.41      0.40      0.40      5232
weighted avg       0.41      0.40      0.40      5232
```

After I realized my mistake and started using the processed text to train my model I discovered that my initial preprocessing techniques were leaving the dataset almost empty. What happened was that I was not accounting for Greek words when I was checking to remove symbols and special characters and all Greek words were being removed aswell. What's interesting though is that even training with next to no text at all the model still managed a considerably close score to the ones I've been getting and the ones others were reporting which further enhanced the argument that the the particular dataset wouldn't allow for much better results whatever techniques we apply. Here are the classification report and learning curves of the model that trained with empty tweets except a small amount that was populated with random english words or numbers:

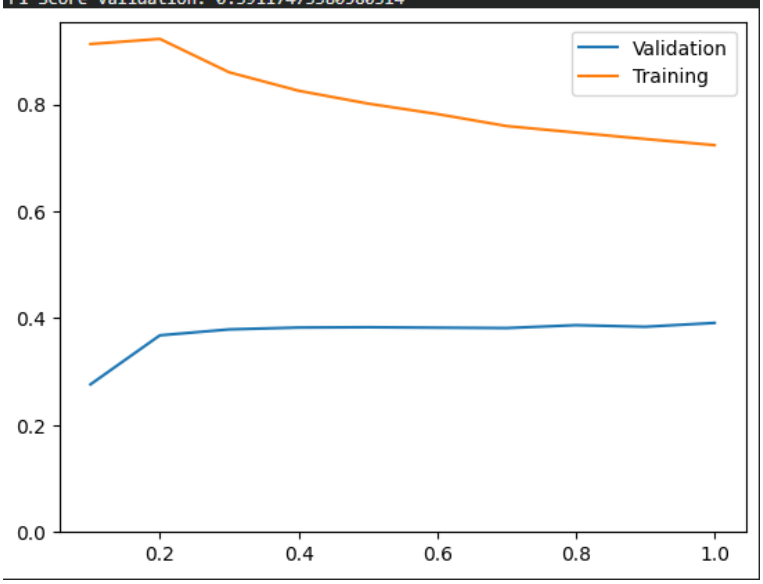
```
F1 Score Train: 0.725
F1 Score Validation: 0.1744336698626402
F1 Score Train: 0.4244329350716952
F1 Score Validation: 0.2810742057647482
F1 Score Train: 0.40726806853122616
F1 Score Validation: 0.2955290061426044
F1 Score Train: 0.3904589407834974
F1 Score Validation: 0.30276612112403406
F1 Score Train: 0.3821094026621124
F1 Score Validation: 0.28852767937688895
F1 Score Train: 0.3767468040160192
F1 Score Validation: 0.29143397683423555
F1 Score Train: 0.3761017813396913
F1 Score Validation: 0.3008425842754071
F1 Score Train: 0.3568581856082697
F1 Score Validation: 0.2755227878786082
F1 Score Train: 0.3624913219674011
F1 Score Validation: 0.2923373397328094
F1 Score Train: 0.36261635371464007
F1 Score Validation: 0.29418249345282127
```




```
Accuracy on the validation set: 0.35

Classification Report:
              precision    recall  f1-score   support

    NEGATIVE       0.34      0.73      0.47      1744
     NEUTRAL       0.35      0.18      0.24      1744
    POSITIVE       0.36      0.12      0.18      1744

    accuracy                           0.35      5232
   macro avg       0.35      0.35      0.30      5232
weighted avg       0.35      0.35      0.30      5232
```

I finally trained the model with the actual preprocessed text which I thoroughly checked to be sure that the preprocessing techniques I applied were applied as intended. Here is the results of the model's performance:

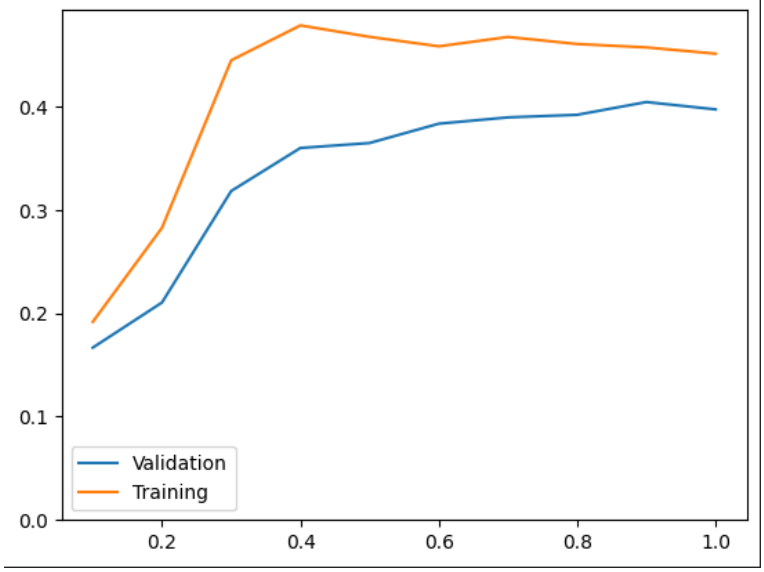
```
F1 Score Train: 0.9128472222222221
F1 Score Validation: 0.27628126771737654
F1 Score Train: 0.9223968860714543
F1 Score Validation: 0.36796186066143083
F1 Score Train: 0.8599233325454037
F1 Score Validation: 0.379056601168708
F1 Score Train: 0.8254724820568063
F1 Score Validation: 0.38253158423044853
F1 Score Train: 0.8014904663152176
F1 Score Validation: 0.3830839087301112
F1 Score Train: 0.7819250817156588
F1 Score Validation: 0.38228033533605804
F1 Score Train: 0.7594253248322667
F1 Score Validation: 0.38156153718791896
F1 Score Train: 0.7472649898710085
F1 Score Validation: 0.38695955732481735
F1 Score Train: 0.7351984062074473
F1 Score Validation: 0.38401401514115435
F1 Score Train: 0.7237121972658908
F1 Score Validation: 0.39117473380580314
```




```
Accuracy on the validation set: 0.40

Classification Report:
              precision    recall  f1-score   support

    NEGATIVE       0.38      0.40      0.39      1744
     NEUTRAL       0.40      0.39      0.39      1744
    POSITIVE       0.41      0.40      0.41      1744

    accuracy                           0.40      5232
   macro avg       0.40      0.40      0.40      5232
weighted avg       0.40      0.40      0.40      5232
```

Although the numbers might not be as high we can clearly see that it's starting to look more consistent and stable overall. The training f1 score though is still alot higher than the validation score which indicates that we're overfitting. I took some time to experiment both with multiple preprocessing techniques ending up with the ones (described in section 2) that led to a better perfoming model (in numbers and on paprer). Also finding and applying better parameters for this particular instance of processed data yielded even better results and a better look on the graph:

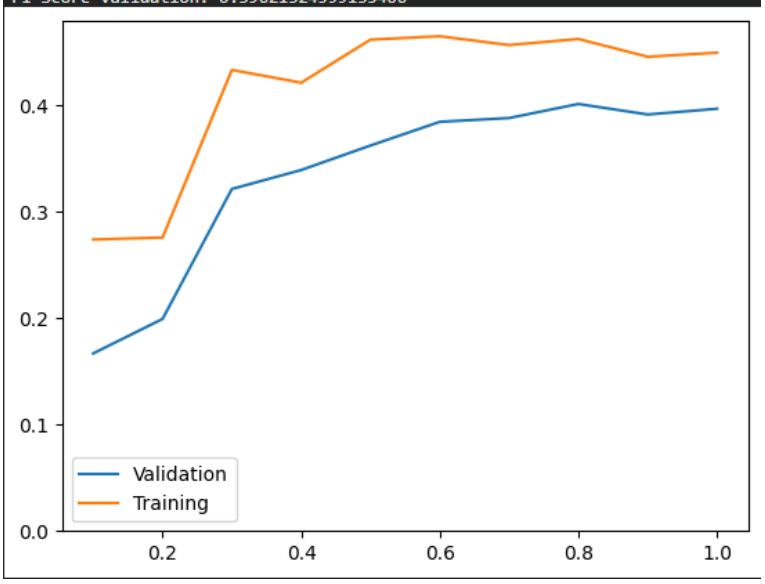
```
F1 Score Train: 0.19160997732426302
F1 Score Validation: 0.16666666666666666
F1 Score Train: 0.2825255641565033
F1 Score Validation: 0.2102916162471517
F1 Score Train: 0.4449270409163233
F1 Score Validation: 0.31811366109027983
F1 Score Train: 0.4782539386256604
F1 Score Validation: 0.3597320533044114
F1 Score Train: 0.4671897440417251
F1 Score Validation: 0.3645473460190032
F1 Score Train: 0.45803965933318275
F1 Score Validation: 0.383234576485484
F1 Score Train: 0.4670753096818218
F1 Score Validation: 0.38932640970256327
F1 Score Train: 0.4602634327894471
F1 Score Validation: 0.39182889763008255
F1 Score Train: 0.4569668812143073
F1 Score Validation: 0.4041511878170926
F1 Score Train: 0.4509300496662985
F1 Score Validation: 0.39704896647592075
```




```
Accuracy on the validation set: 0.40

Classification Report:
              precision    recall  f1-score   support

    NEGATIVE       0.40      0.52      0.45      1744
     NEUTRAL       0.40      0.35      0.37      1744
    POSITIVE       0.42      0.34      0.38      1744

    accuracy                           0.40      5232
   macro avg       0.40      0.40      0.40      5232
weighted avg       0.40      0.40      0.40      5232
```

After experimenting a bit more with preprocessing techniques and staying only with the ones that resulted with a better performing model, I turned my attention to the post-training optimization. I fiddled with hyperparameters more and started using grid search to find the best ones for my model that extracted most of the preprocessed data and these are the best results I ended up with:

```
F1 Score Train: 0.2735042735042735
F1 Score Validation: 0.16666666666666666
F1 Score Train: 0.2754304515029979
F1 Score Validation: 0.19902277366082455
F1 Score Train: 0.4326928274957165
F1 Score Validation: 0.32094811399233936
F1 Score Train: 0.4206775942936499
F1 Score Validation: 0.3387298387532438
F1 Score Train: 0.46108797435250776
F1 Score Validation: 0.36171711877975676
F1 Score Train: 0.4642669527905559
F1 Score Validation: 0.38395867187765637
F1 Score Train: 0.45598886020441937
F1 Score Validation: 0.3875214729961675
F1 Score Train: 0.46169034069026776
F1 Score Validation: 0.40071694011565717
F1 Score Train: 0.4450145248268466
F1 Score Validation: 0.39085617685674073
F1 Score Train: 0.44885176356846496
F1 Score Validation: 0.39621324599135466
```




```
Accuracy on the validation set: 0.40

Classification Report:
              precision    recall  f1-score   support

    NEGATIVE       0.40      0.52      0.45      1744
     NEUTRAL       0.40      0.35      0.37      1744
    POSITIVE       0.42      0.34      0.38      1744

    accuracy                           0.40      5232
   macro avg       0.40      0.40      0.40      5232
weighted avg       0.40      0.40      0.40      5232
```

Trial	Pre-processing	Vectorization	Optimization	Score
Unprocessed Text	-	TF-IDF	-	0.39
Almost Empty Text	Greek words removed	TF-IDF	-	0.35
Preprocessed Text	Text cleaning, tokenization, lemmatization	TF-IDF	Hyperparameter tuning	0.40

Table 1: Trials

3.1.1. Table of trials.

3.2. Hyper-parameter tuning

The hyperparameter tuning process involves playing with various parameter combinations to optimize the logistic regression model's performance. After finding some parameters and possible combinations for each and playing with them I decided to use grid search which tests all possible combinations of the parameters you provide it and gives you the ones that yield the best results in the particular instance of the model depending on the training and the data it trains on. I noticed that there wasn't a standard set of parameters that would always be better and that grid search would result with different parameters each time depending the data.

With my experimenting I came to the conclusion that changing the parameters won't usually change the perdormance of the model in a drastic way or nearly as much as it can with preprocessing the data and trying different techniques. What it really has an impact on is how the training process will be conducted and how much will the model be over/under - fitting. This might have to do with the fact that the results with our particular dataset would not go much higher than 0.4/0.41 accuracy but the way the model trains and how the fitment is applied would make a bigger difference that would be easily visible by looking at the learning curves provided above. Overall playing with the parameters helped reduce the apparent overfitting and enhance the model's generalizability as the results from the training and validation dataset are now much closer indicating that the model is as good at analyzing unknown tweets that it is at analyzing tweets that it has trained on.

3.3. Optimization techniques

Involving post training optimization techniques I only experimented with hyper-parameter tuning described above.

3.4. Evaluation

For the evaluation of the model's performance I plotted learning curves and run the classification report function provided by the Scikit-Learn library. The results are all showcased in section 3.1

<Provide and comment diagrams and curves>

3.4.1. ROC curve.

3.4.2. Learning Curve.

3.4.3. Confusion matrix.

4. Results and Overall Analysis

4.1. Results Analysis

This section was pretty much covered by section 3.1

4.1.1. Best trial.

Accuracy on the validation set: 0.40

Classification Report:

	precision	recall	f1-score	support
NEGATIVE	0.40	0.52	0.45	1744
NEUTRAL	0.40	0.35	0.37	1744
POSITIVE	0.42	0.34	0.38	1744
accuracy			0.40	5232
macro avg	0.40	0.40	0.40	5232
weighted avg	0.40	0.40	0.40	5232

4.2. Comparison with the first project

<Use only for projects 2,3,4>
<Comment the results. Why the results are better/worse/the same?>

4.3. Comparison with the second project

<Use only for projects 3,4>
<Comment the results. Why the results are better/worse/the same?>

4.4. Comparison with the third project

<Use only for project 4>
<Comment the results. Why the results are better/worse/the same?>

5. Bibliography

References

[1] Data Magic (by Sunny Kusawa). How to use kaggle notebook | kaggle tutorial | machine learning | data magic.

[2] Kindson The Genius. Logistic regression in python step by step in 10 minutes.

[3] Kunaal Naik | Data Science Masterminds. sklearn logistic regression hyperparameter optimization.

[4] Data School. Important tuning parameters for logisticregression.

[5] utsav aggarwal. Text preprocessing | tokenization | cleaning | stemming | stopwords | lemmatization.

[1] How to use Kaggle Notebook | Kaggle Tutorial | Machine Learning | Data Magic
[2] Logistic Regression in Python Step by Step in 10 minutes
[5] Text Preprocessing | tokenization | cleaning | stemming | stopwords | lemmatization
[3] sklearn Logistic Regression hyperparameter optimization
[4] Important tuning parameters for LogisticRegression
StackOverflow What is the best way to remove accents (normalize) in a Python unicode string?