Çağrı Çiftçi 2016400243
Öykü Yılmaz 2019400294
M. Erdinç Oğuz 2017400267

MACHINE LEARNING PROJECT REPORT
SENTIMENT ANALYSIS ON IMDB USER REVIEWS

STEP 1

DATA COLLECTION

Collection Method

The data necessary is collected with using Python. IMDbPY library is used with some modifications. Since the library was not enough to satisfy our needs. The changes made are listed below:

- In movieParser.py -> in “DOMHTMLReviewsParser” class -> in Rule key “title”:

The tag that contains the title of the review was taken as if it was in a <div> tag. But currently, the site puts the title in an <a> tag.

```
        extractor=Path('.//div[@class="title"]//text()')
```

was changed to:

```
        extractor=Path('.//a[@class="title"]//text()')
```

- In movieParser.py -> in “postprocess_data” function:

When retrieving the rating, library function was only returning the ratings only if they had length two. This corresponds to ratings with value 10. Besides, the function returned only the first digit of that rating, i.e. ‘1’. If the rating had only one digit, the output was “null”. The function is altered so that any rating is properly returned.

```
            if review.get('rating') and len(review['rating']) == 2:
                review['rating'] = int(review['rating'][0])
```

was changed to

```
            if review.get('rating') and len(review['rating']) <= 2:
                review['rating'] = int(review['rating'])
```

This method is chosen because in the future, if it is needed to increase the number of reviews it can be done by changing only one or two lines of code.

Details of Data Collection

The task was to retrieve reviews of the movies that start with the letter F. To do that, the top 250 IMDb movies and top 250 Indian movies are collected. Because these had high rankings, to avoid bias in data two words with initial ‘f’ (fail and fast) were searched among the movies. Maximum of twenty-five reviews were taken from a specific movie, also to prevent bias. Reviews taken are shuffled before being sorted by their ratings. Finally, they are written to files according to the instructions given. It has been realized that some (four) reviews are not encoded in python write function’s default, hence we added a try catch check to ‘write to file’ part of code to validate the encoding of the review.

Summary of Work Done

Since the first part did not include heavy work, we met online for coding. First, Mehmet Erdinç Oğuz made a research about the library and started the initial project. Afterwards, someone shared their screen and coding was done simultaneously. Also, the report was written in the same way.

STEP 2

Important Note:

After installing the requirements.txt, run:

```
python -c "import nltk; nltk.download('punkt'); nltk.download('stopwords'); nltk.download('averaged_perceptron_tagger'); nltk.download('wordnet')"
```

In this step the main purpose of the assignment was to select a model for our data. To decide on that, it was needed to do a literature search on how to clean the data, feature extraction and selection, and models can be used. The dataset was given, so the content and class information is retrieved from this dataset.

DATA PREPROCESSING

The data consist of sentences written by people all over the world in English. Thus, data was not clean at first to extract features. To get a more indicative data set, the first step was to remove the special character, double spaces, single characters, and links. Then all characters were turned into lower case.

After getting a cleaner charset, looking at the dataset, it has been realized that there are too many versions of the same word in the grammar of English. For instance, words “love”, “loved”, “loves”, “loving” have all the same root: “love”. These words being different features could make a negative effect on the results by decreasing the overall number of occurrences. To deal with this issue, in literature there were two popular techniques: stemming and lemmatization. They both generate the root of the words. The difference between them is stemming may create words that do not exist where lemmatization depends on the word’s part-of-speech and produces dictionary words. Even the lemmatization is more complex and takes more time to execute, it is used to have better results.[1]

In lemmatization according to NLTK library the words are tagged according to part-of-speech tagging. Then using WORDNET library each word is lemmatized.

[1] https://www.baeldung.com/cs/stemming-vs-lemmatization

This process is applied both on the test and the training data.

To remove the meaningless common words, stopword list from NLTK library is used. In addition to these words, some missing words are added to that list (e.g. “us”, “la”, “br”). Those newly added words are taken from a file name words.txt which is in the submission in Moodle.

FEATURE EXTRACTION AND SELECTION

To create features, two methods were tried on the dataset: TFIDF and BAG OF WORDS (Count). TFIDF measures the relevance of the word to a document. [2] Bag of words simply counts the frequencies of words.[3] To get the features “scikit learn”s built-in functions were used. Then two methods of feature selection were tried on the data: mutual information and giving different parameters to TFIDF and COUNT vectorizers (maximum number of features, maximum percentage of the number of documents that the feature exists – max_df (maximum document frequency), minimum number of documents that the feature exists-min_df (minimum document frequency)). In bag of words, both one and two-gram words are used.

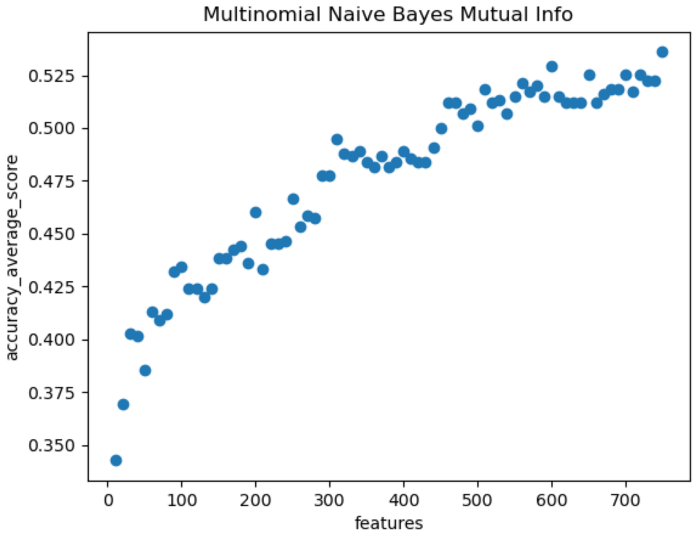
Mutual information filters the features according to the amount of information the presence and absence of a term contributes to the correctness of classification on each class.[4]

For mutual information, first the features are extracted with TFIDF and COUNT vectorizers, giving the default parameters so that they do not do extra feature selection. Then the selected features are tested on the test data with several candidate models. The built-in function returns the scores of all features. Then the features are sorted by their scores. From this sorted list, best scored features are taken. Another parameter (threshold) is used to determine the number of features. In another words in each run, we took the best <threshold> number of features and tried only on Multinomial Naïve Bayes since it took huge amount of time. The accuracy scores were much lower than TFIDF values, so it is not tried on other models due to time limits.

[2] monkeylearn.com/blog/what-is-tf-idf/

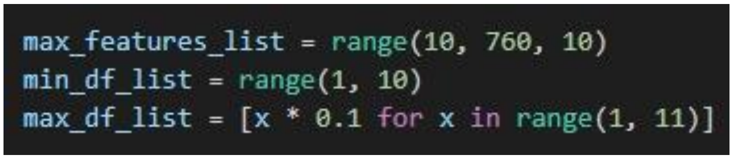
[3] https://aiaspirant.com/bag-of-words/

[4] https://nlp.stanford.edu/IR-book/html/htmledition/mutual-information-1.html#:~:text=A%20common%20feature%20selection%20method,the%20correct%20classification%20decision%20on%20



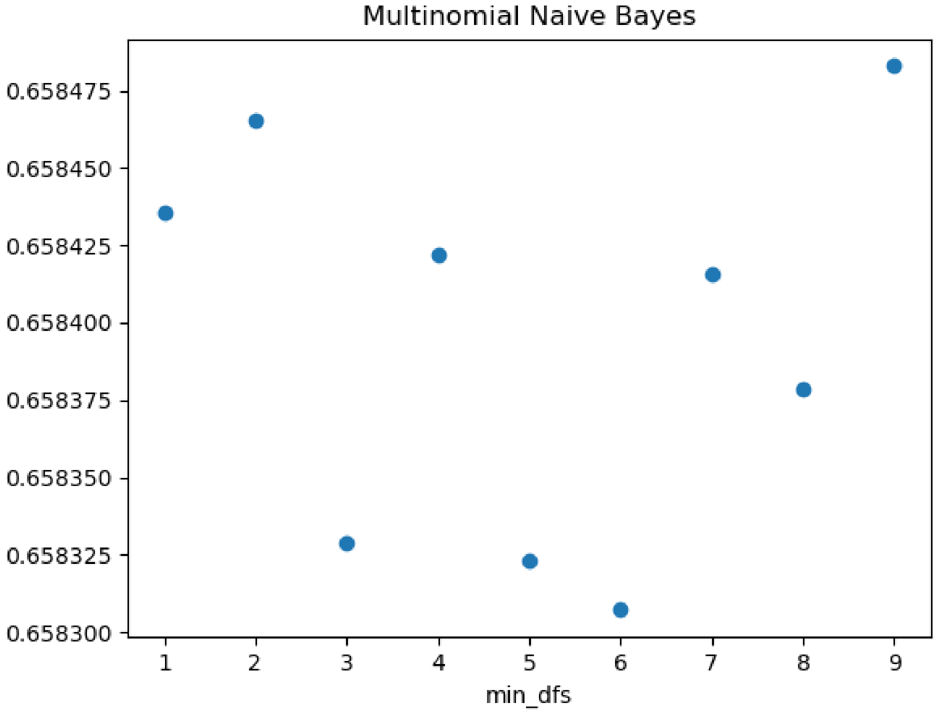
The graph below shows the effect of the threshold value on mutual information selection.

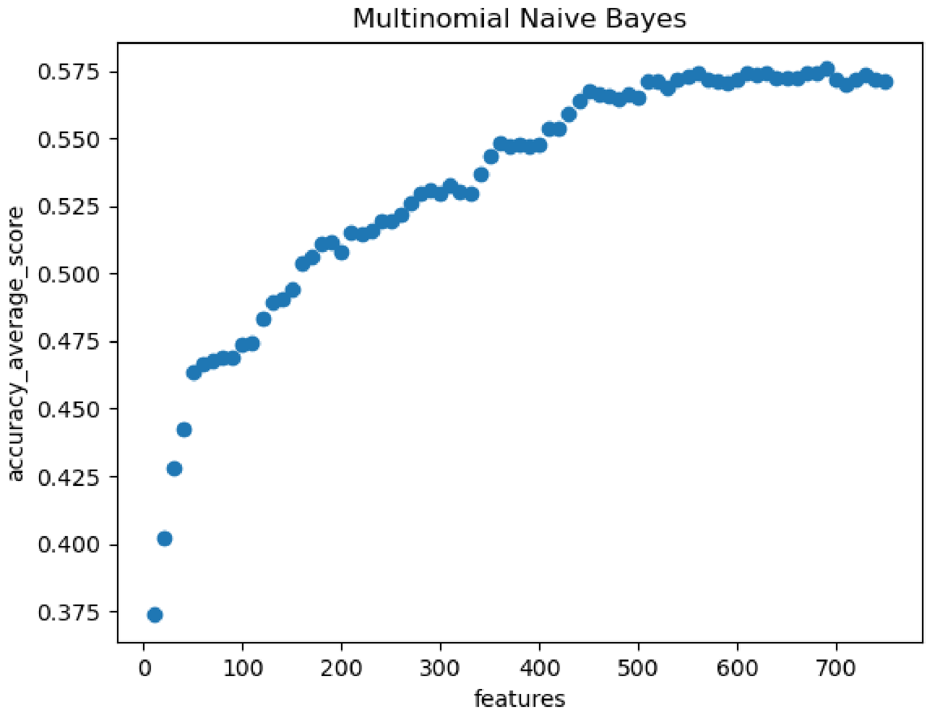
For the parameter selection, ranges of parameters are as follows:

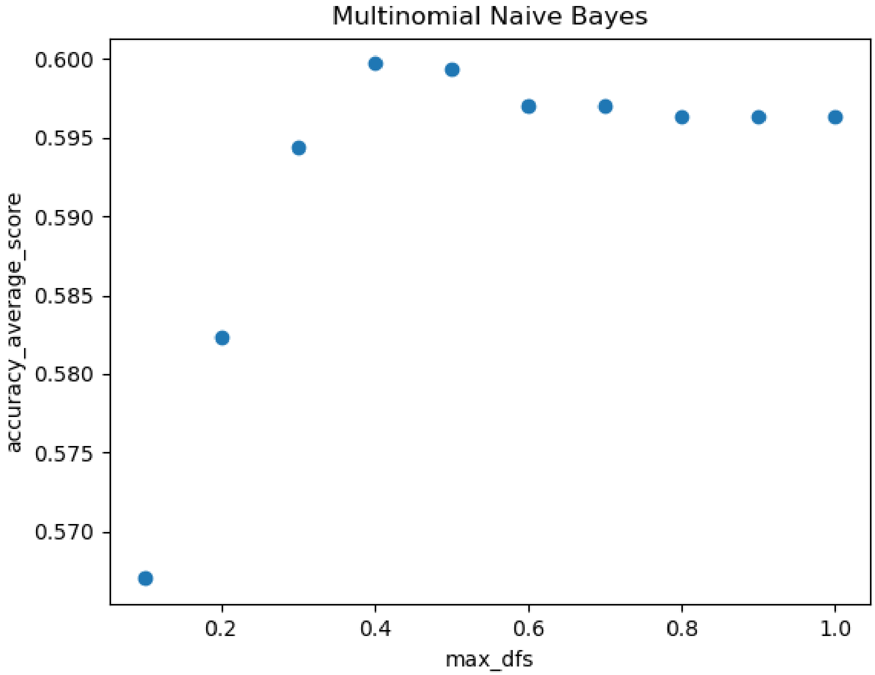


```
max_features_list = range(10, 760, 10)
min_df_list = range(1, 10)
max_df_list = [x * 0.1 for x in range(1, 11)]
```

After selection of features, those features are tested on the VAL data with several candidate models. After selecting the best model (see Model Selection) with the best parameters, the average of the accuracies for that model is taken and plotted. These graphs give the idea of the effect of the parameters on the accuracy. Yet, these graphs do not indicate specific accuracies since the accuracies are the average values of different combinations of max_features, max_df, and min_df values. These graphs are only for Multinomial Naïve Bayes, which gives the best accuracy result.


Multinomial Naive Bayes
0.658475
0.658450
0.658425
0.658400
0.658375
0.658350
0.658325
0.658300
1
2
3
4
5
6
7
8
9
min_dfs





As can be seen from the Mutual Information and the features graph, in the overall, performance increases as the number of features increase.

MODEL SELECTION

Candidates

To get the highest accuracy, different combinations of feature selection method (i.e bag of words and TFIDF) parameters are tried on Multinomial Naïve Bayes, Gaussian Naïve Bayes, Logistic Regression with 5-fold cross validation for determining the learning rate, and Random Forest Regression.

Since training takes a lot of time, the maximum feature numbers are multiples of 50. After getting the best model, interval is decreased to 1 to get more precise and the best model combination.

According to the results below, best performance is from Multinomial Naïve Bayes.

Multinomial Naïve Bayes - TFIDF

Min_df	Max_df	Feature Number	Accuracy
0.4	2	1780	0.676
0.4	2	1750	0.674
0.4	2	1760	0.674

Multinomial Naïve Bayes – Bag of Words

Min_df	Max_df	Feature Number	Accuracy
0.4	2	1650	0.657
0.4	2	1600	0.655
0.4	2	1750	0.653

Gaussian Naïve Bayes - TFIDF

Min_df	Max_df	Feature Number	Accuracy
0.4	1	600	0.6
0.4	2	600	0.6
0.4	3	600	0.6

Gaussian Naïve Bayes – Bag of Words

Min_df	Max_df	Feature Number	Accuracy
0.3	2	650	0.512
0.3	10	650	0.512
0.3	3	650	0.511

Logistic Regression - TFIDF

Min_df	Max_df	Feature Number	Accuracy
0.5	3	450	0.633
0.5	4	450	0.633
0.5	5	450	0.633

Logistic Regression – Bag of Words

Min_df	Max_df	Feature Number	Accuracy
0.6	8	550	0.599
0.7	8	550	0.599
0.5	4	550	0.596

Random Forest - TFIDF

Min_df	Max_df	Feature Number	Accuracy
0.4	2	600	0.370
0.4	2	650	0.358
0.4	2	700	0.358

Random Foster – Bag of Words

Min_df	Max_df	Feature Number	Accuracy
0.4	2	700	0.339
0.4	2	650	0.339
0.4	2	600	0.338

Selected Model

The model used is Multinomial Naïve Bayes. Feature selection is done with TFIDF method with parameters min_df=2, max_df= 0.4, max_features = 1780. The performance metrics are as follows:

Accuracy: 0.676

Precision: {'P': 0.76953125, 'N': 0.667953667953668, 'Z': 0.5829787234042553}

Recall: {'P': 0.788, 'N': 0.692, 'Z': 0.548}

Macro Average Precision: 0.6734878804526411

Macro Average Recall: 0.676

Summary of Work Done

We coded all the parts together on Discord.

STEP 3

Important Note:

After installing the requirements.txt, run:

```
python -c "import nltk; nltk.download('punkt'); nltk.download('stopwords');
nltk.download('averaged_perceptron_tagger'); nltk.download('wordnet')"
```

FEATURE EXTRACTION AND SELECTION

This time, in addition to TFIDF and count vectors, pre-trained word embedding models were used. The models used are Glove from Stanford University, FastText from Facebook, and Google's News Vectors. From Stanford, Common Crawl and Twitter data was used. Each of them had different dimensions, vocabulary token count. For more information, check the links in the footnote.[5]

For each word embedding model, feature vectors are calculated as follows: First, for each token in that document, vector of that token is taken from the model. Then three different methods were tried in order to get the best feature vector. First one was to sum each token's vector, second one was to take average of those vectors, and the third one to take the weighted average of tokens according to their TFIDF values in that document.

Also, min-max normalization is tried on document feature matrix to get higher scores. However, since the results were around 35%, they are not included in the graphs.

In addition, in the above approaches, vocabulary of the documents was all tokens coming from all documents. In this trial, the vocabulary created from TFIDF vectorizer with parameters min_df=2, max_df=0.4 and max_features = 1780 were used to limit the vocabulary of the documents. Those parameters are selected since the best result in step 2 was taken by those. Yet, the results were around 34%, so they are not included in the graphs.

MODEL SELECTION

Candidates

To get the highest accuracy, different combinations of pre-trained word embedding models and TFIDF vectors are tried on Gaussian Naïve Bayes, Logistic Regression, Random Forest Regression and different from step 2, Support Vector Machine and Linear Regression.

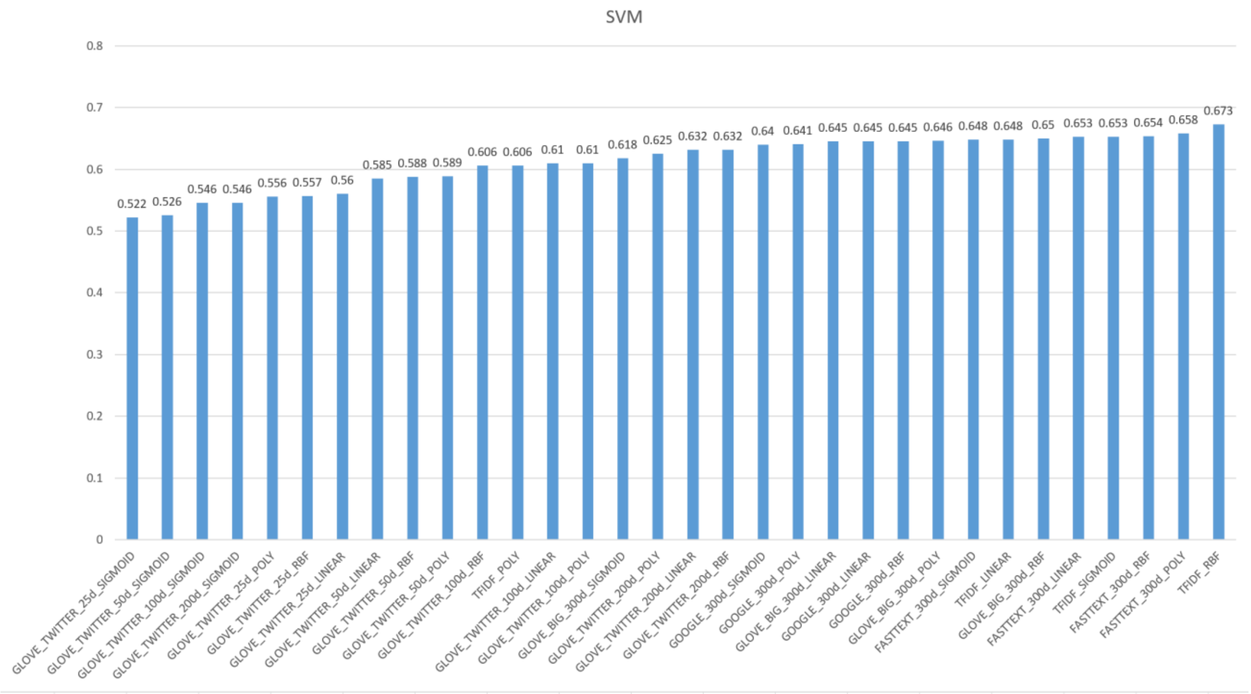
Graphs of the results are as follows:

SVM

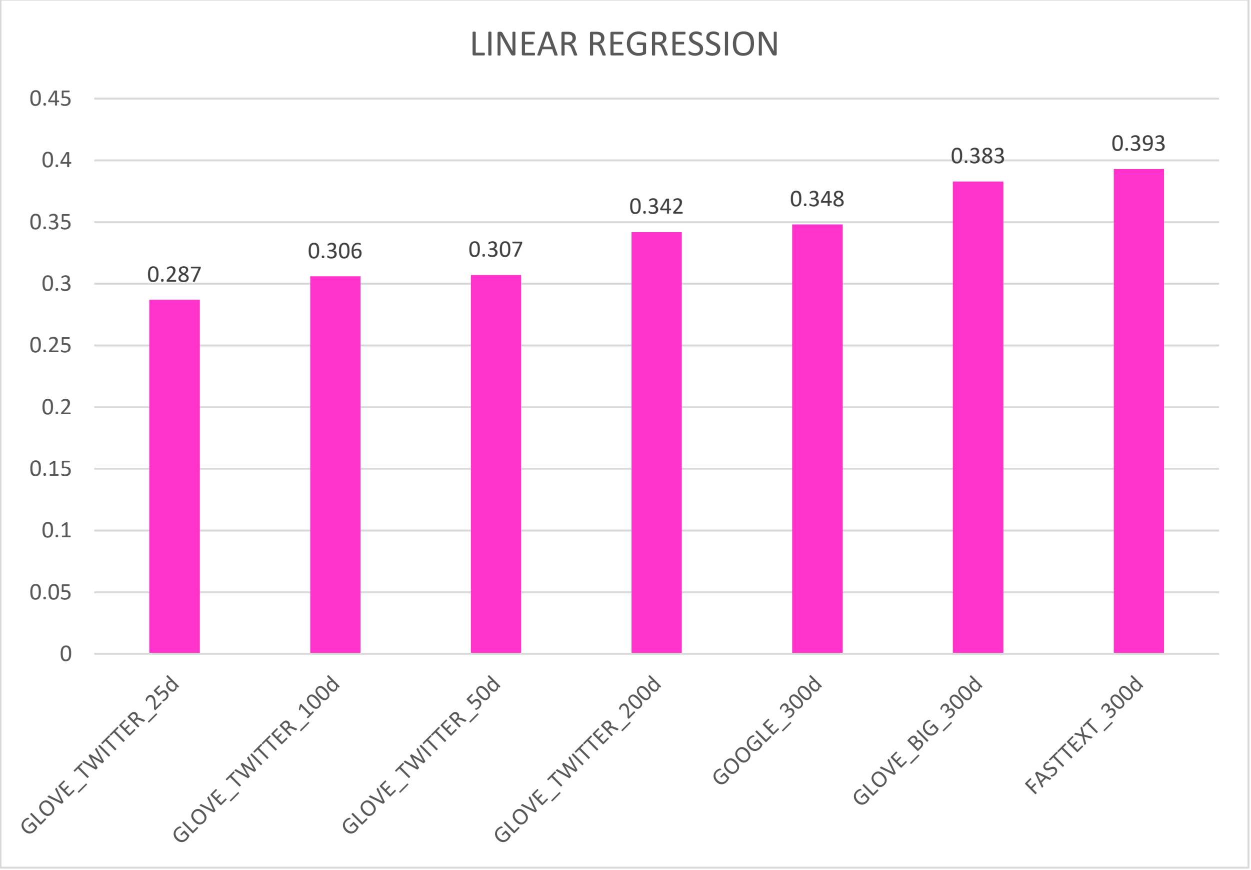
For each feature matrix, SVM is tried with different C values ranging from 0.001 to 100, and different kernels which are linear, polynomial, sigmoid and radial basis function (RBF). For the gamma parameter, "auto" (1 / n_features) and "scale" (1 / (n_features * X.var()) are used.

The best results are taken with c values around 4-5, when kernel is RBF, and with gamma parameter scale.

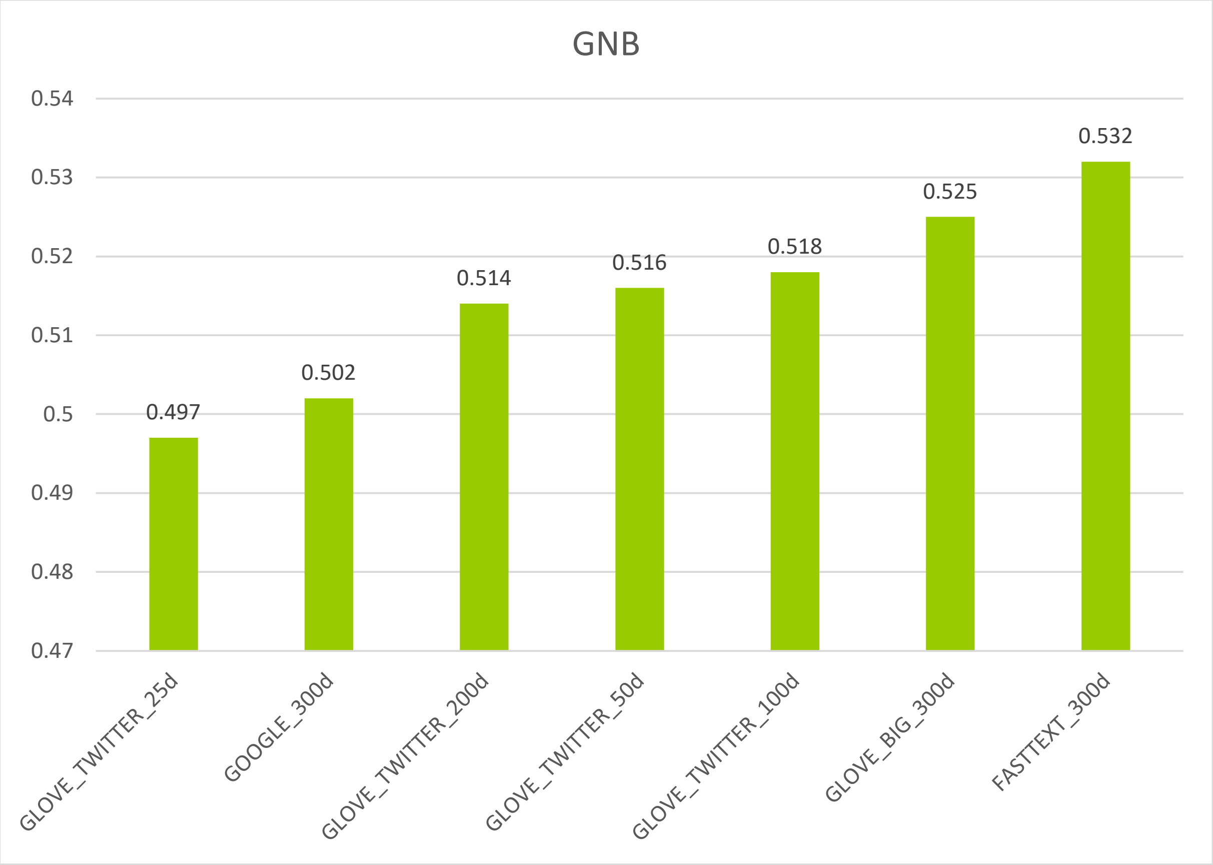
[5] https://github.com/stanfordnlp/GloVe, https://fasttext.cc/docs/en/english-vectors.html, https://code.google.com/archive/p/word2vec



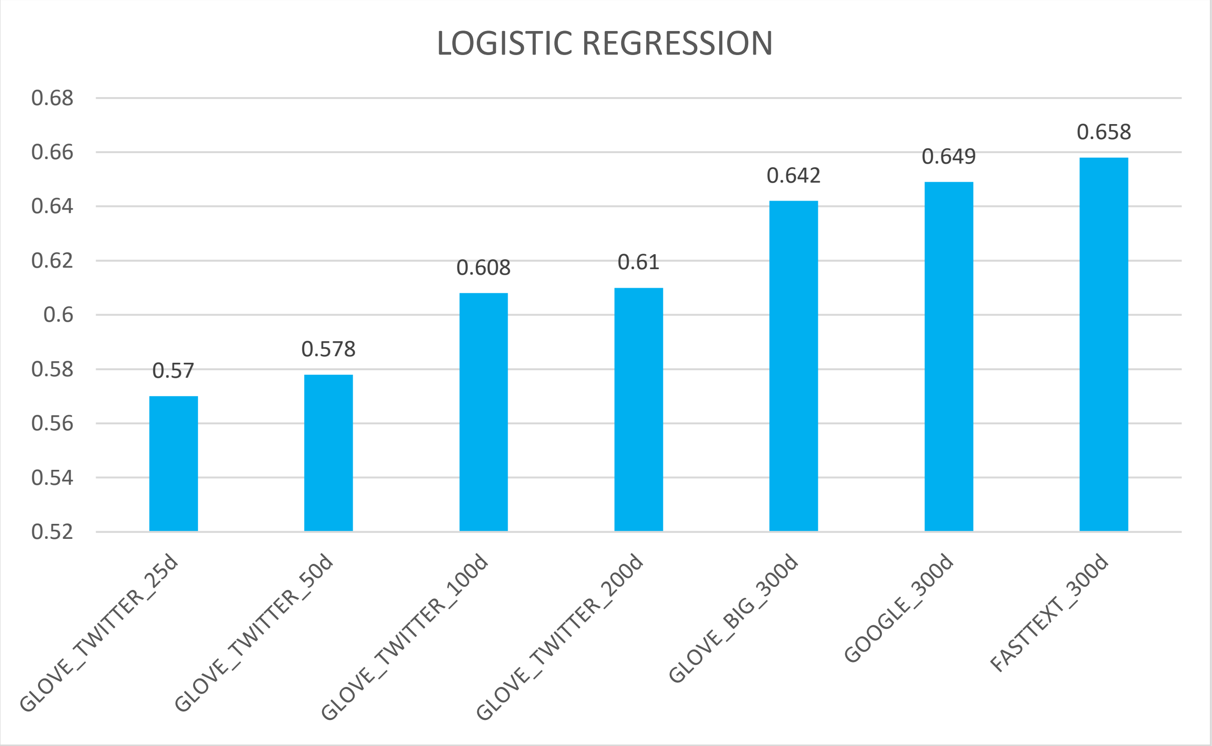
LINEAR REGRESSION



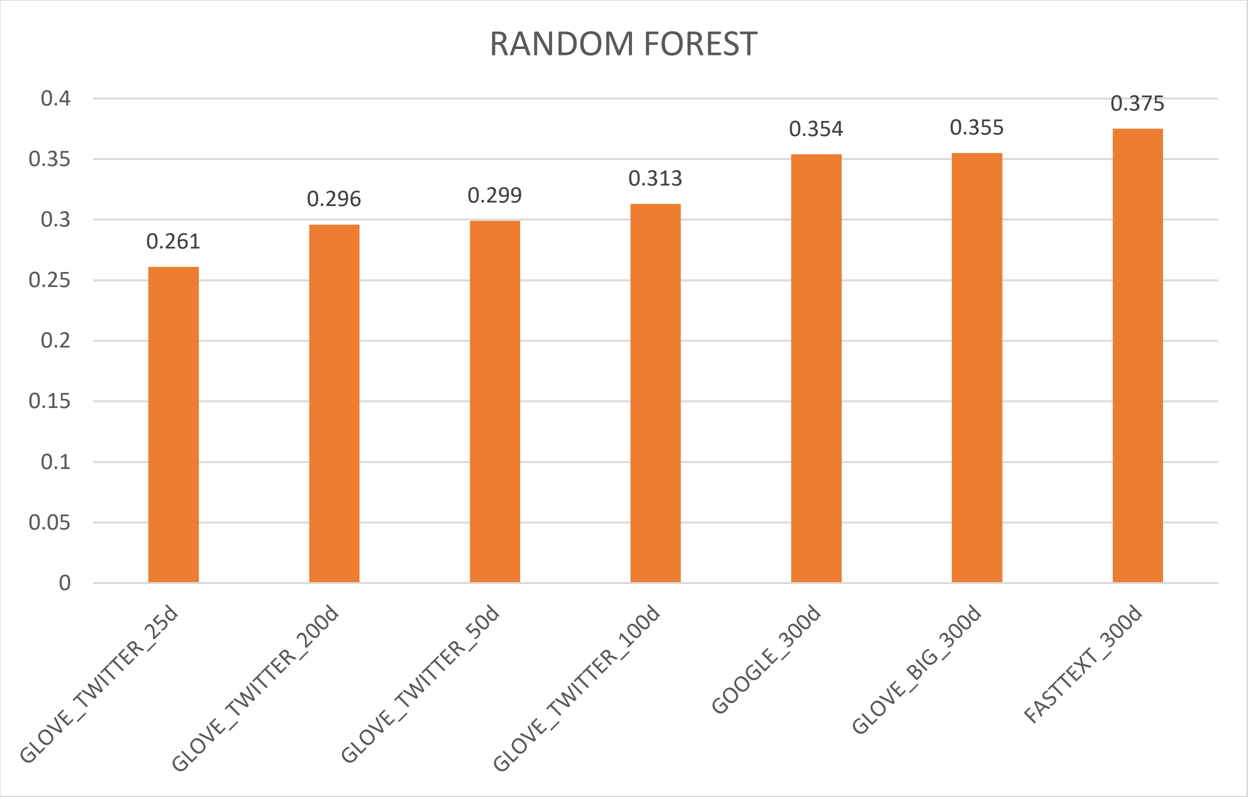
GAUSSIAN NAÏVE BAYES



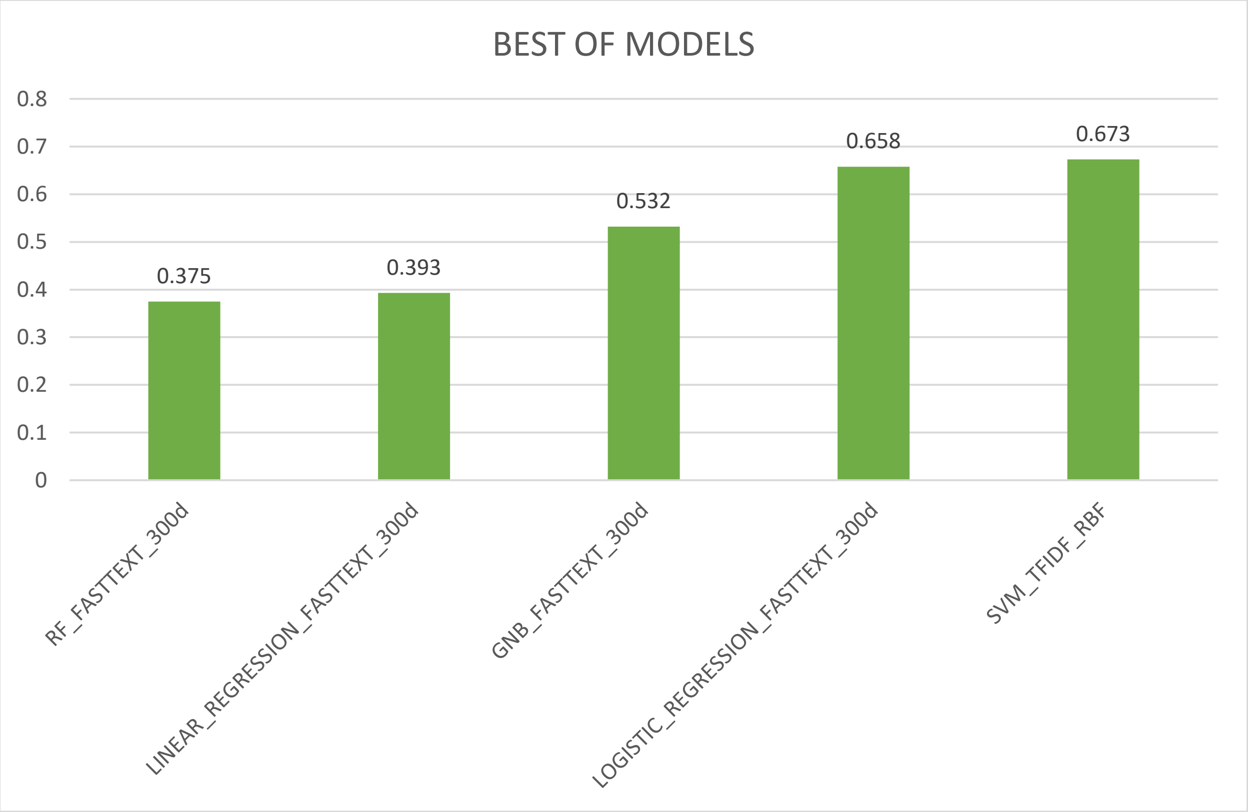
LOGISTIC REGRESSION



RANDOM FOREST



OVERALL RESULTS



Selected Model

The model used is SVM with kernel RBF, c value 5 and gamma scale. Feature selection is done with TFIDF method with parameters min_df=9, max_df= 0.8, max_features = 2359. Although the accuracy on validation data is lower than step 2, it is decided to test SVM on test data. The performance metrics are as follows:

Accuracy: 0.6733333333333333

Precision: {'P': 0.7509578544061303, 'N': 0.7008196721311475, 'Z': 0.563265306122449}

Recall: {'P': 0.784, 'N': 0.684, 'Z': 0.552}

Macro Average Precision: 0.6716809442199089

Macro Average Recall: 0.6733333333333333

Summary of Work Done

Work done by everyone

- Research on word embedding models and their methods
- Research on SVM and its parameters
- Creating plots and writing the report
- Writing of main script
- Deciding on transformer functions for embedding vectors.

Öykü Yılmaz

- Applying SVM with TFIDF and Glove. Trying different parameters for SVM and TFIDF
- Applying one of the transformer functions.
- Running old models with glove.

Mehmet Erdinç Oğuz

- Applying SVM and Linear Regression with FastText. Trying different parameters for SVM.
- Applying one of the transformer functions.
- Running old models with FastText.

Çağrı Çiftçi

- Applying SVM and Linear Regression with Google word embedding model. Trying different parameters for SVM.
- Applying one of the transformer functions.
- Running old models with Google model.

GITHUB REPOSITORY LINK

https://github.com/eridincu/cmpe462_project