Hate Speech Detection - Pilot Study

Kareem Ehab Kassab 900182771
Computer Science and Engineering
The American University in Cairo
kareemikassab@aucegypt.edu

Mahmoud Elshinawy 900183926
Computer Science and Engineering
The American University in Cairo
mahmoudelshenawy@aucegypt.edu

I. Introduction

During the last stage of the preprocessing phase, we were presented with the challenge of putting the corpus of our data into a suitable representation, and we chose the TF-IDF to avoid the sparsity of the Bag of Words representation. Our corpus contains 31011 unique words, which would require the creation of a 31011*1 vector with each word's frequency; considering we have 135556 entries (sentences), this would take an infeasible memory space and would be definitely inefficient. A convenient, more efficient, and more suitable way to our case would be the Word2vec text embedding that was discussed in class. Word2Vec text embedding has a significantly smaller size for each word compared to the BoW and has the much needed extra characteristic of considering the semantics of the word. In Word2Vec embedding we represented our corpus as numerical embedded vercors, which was then used to train the model on our previously-cleaned dataset using different models. Such models included Logistic Regression, Naive Baye's, Decision Trees, Perceptron, a Neural Network, and SVM. The results of each model were to be reported and analyzed in relationship to the needs of the project's dataset requirements (the pros and cons relative to what is needed), and the top three were relatively compared to each other in order to be able to conclude the best model for the project's case.

II. The Dataset

The data is obtained from Kaggle and it is the most recent as it updated on January 21st, 2022. It contains 135557 instances which is large enough. This dataset can be found here.

The data is very recent, and it has large enough size to train our model on. We will split our data into two sets Training, and Test sets. Training Set will be (80%) and Test Set will be (20%), and we will be using K-fold cross validation in our validation process as it is the best method for validation.

III. Feature Label Correlation

In the process of data cleaning, we need to do feature label correlation to identify how probable each word will infer the class label, and thus better understanding of the relation between the features and the label. In the following graphs, we will be using bunch of frequency dictionaries and bar plots to investigate the correlation in the training data.

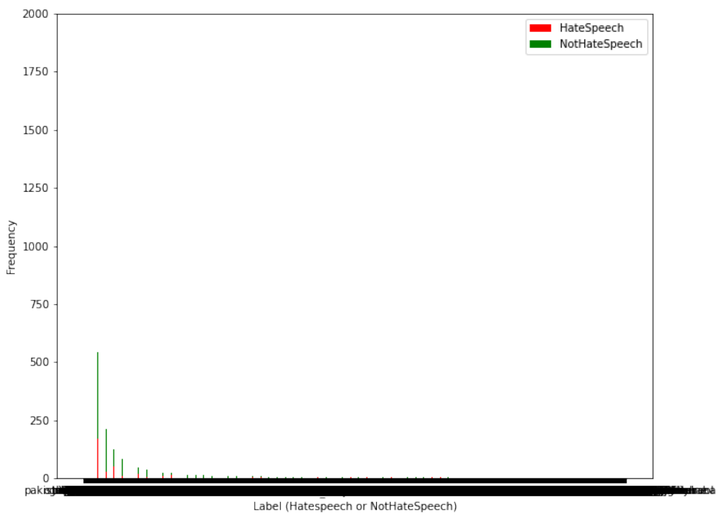


Fig. 1. All words Correlation.

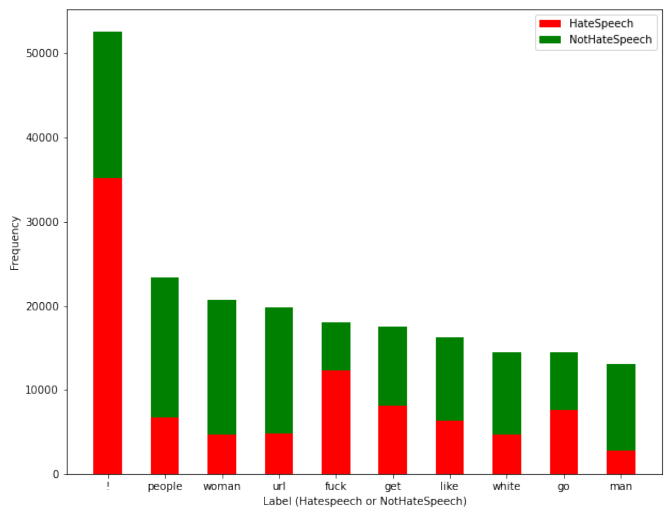


Fig. 2. Most 10 frequent words.

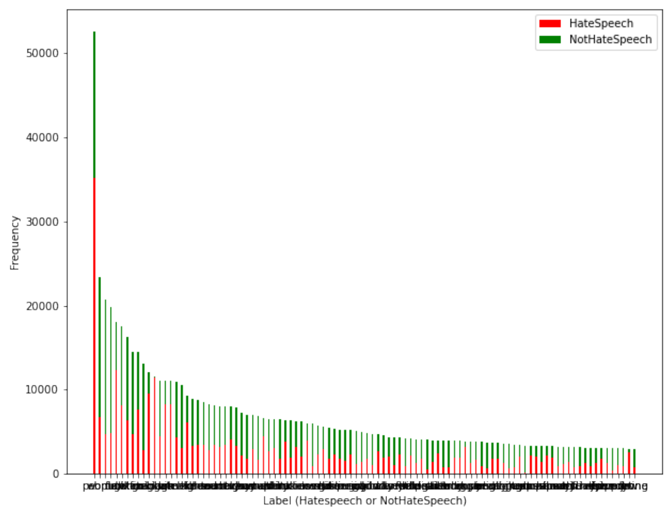


Fig. 3. Most 100 frequent words.

As we see from the graphs the data starts to correlate when we investigate 15000 words which is half of the data. Keeping in mind that it will not be good to drop half of the words appeared in the data, we decided to keep these words to give better context for the word embedding model to operate.

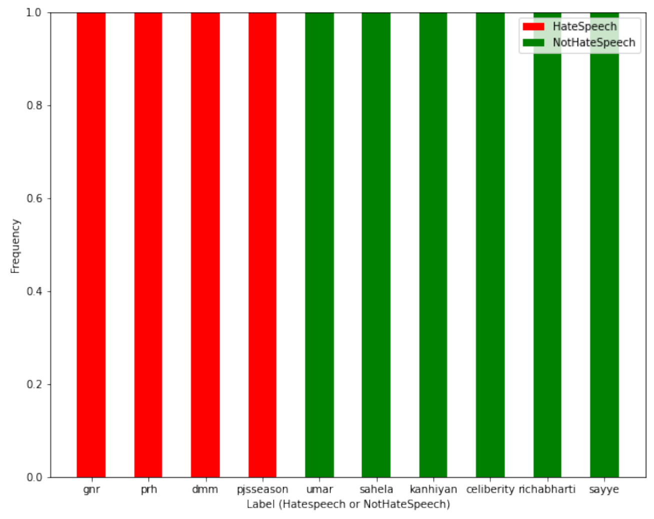


Fig. 4. Least 10 frequent words.

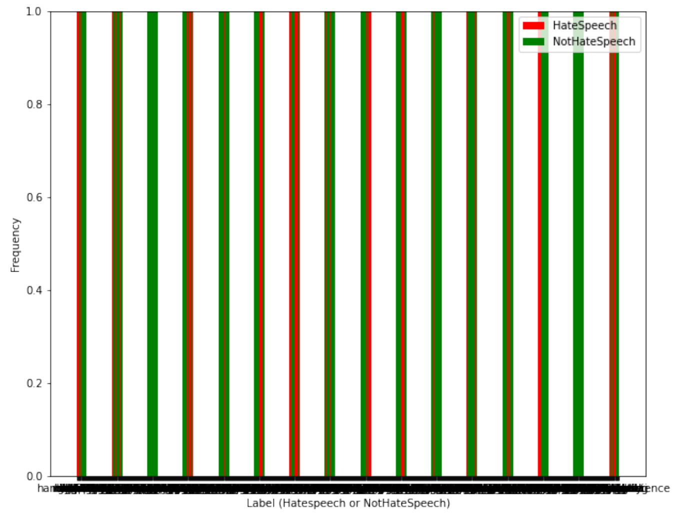


Fig. 5. Least 1000 frequent words.

IV. Data Representation (Word2Vec)

In the preprocessing phase, the data had a class defining column called “hate speech”, and a “text” column with the main text; we created two new columns: one column with the final results of text preprocessing that included removing stop words, numbers, punctuation, and stemming, and the other column that had the TF-IDF. In this phase,

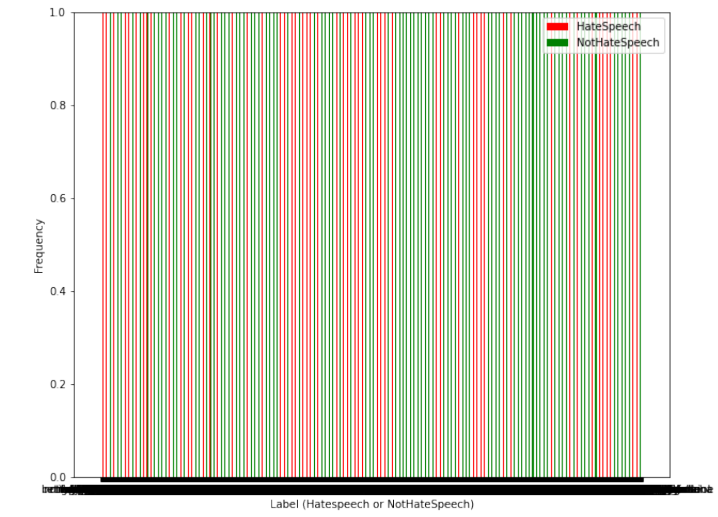


Fig. 6. Least 10000 frequent words.

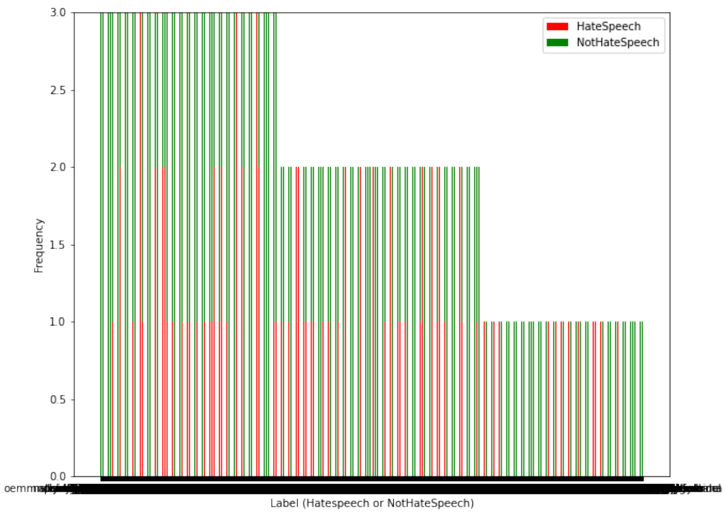


Fig. 7. Least 15000 frequent words.

we have investigated the feature level correlation which we missed last time. Furthermore, since the TF-IDF has memory challenges due to the size of data we are addressing in this project, we decided to use a Word2Vec Numerical representation. This has two main edges over the TF-IDF; first, Word2Vec Text Embedding is more memory efficient, making it more suitable for the case we are considering, and second, it considers the semantic context of the word in relation to the other words around it. Word2Vec embedding has two techniques: the CBOW and the skip-gram. The CBOW model considers a context window and tries to predict a missing word in the middle of the window, and iterates over the corpus with a certain window size.

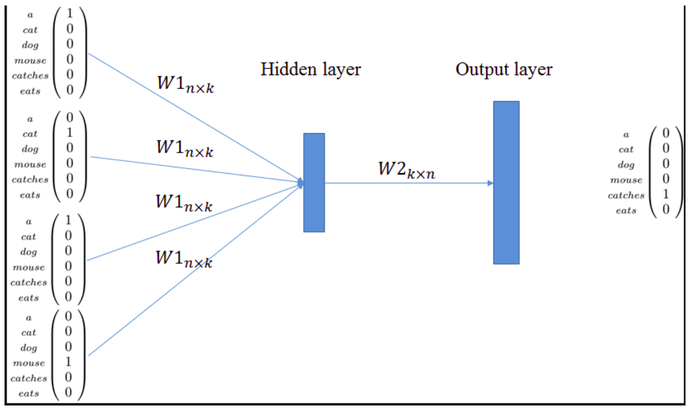
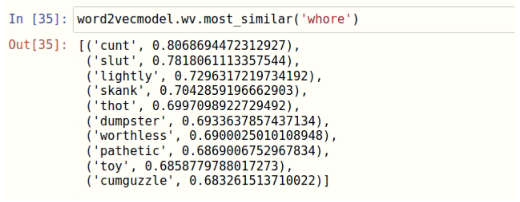


Fig. 8. Word2Vec.

On the other hand, the skip-gram takes the word in the middle of the window and tries to predict the rest of the window (predict the context). Both consider the context, the skip-gram considers it better, but is significantly slower. We used the CBOW model as it was more suitable for our case: it is faster to compile, it predicts the word in its window context, and since we want to identify certain offensive words and hate speech in whichever context, we saw no need to invest the extra time in the skip-gram variation. For our CBOW, we used a vector size of 500 and a window size of 4. The window looped over the whole corpus generating a training set of (108,444*500) and testing set of (27,112*500) an 80/20 split. We can see the words with the highest similarity to the word “whore”.



```
In [35]: word2vecmodel.wv.most_similar('whore')

Out[35]: [('cunt', 0.8068694472312927),
          ('slut', 0.7818061113357544),
          ('lightly', 0.7296317219734192),
          ('skank', 0.7042859196662903),
          ('thot', 0.6997098922729492),
          ('dumpster', 0.6933637857437134),
          ('worthless', 0.6900025010108948),
          ('pathetic', 0.6869006752967834),
          ('toy', 0.6858779788017273),
          ('cumguzzle', 0.683261513710022)]
```

Fig. 9. Similar Words to the word Whore.

We used the output dictionary of vectors to train the different models we considered for the project: logistic regression, Naive Baye's, Decision Trees, Perceptron, a Neural Network, and SVM.

V. Experimental Analysis For Models

After Correlation and text embedding, we used the output numerical representation dictionary of vectors to train the five considered models: logistic regression, Naive Baye's, Decision Trees, Perceptron, and a Neural Network. For the models, we used the SKLearn implementation. We calculated the accuracy, precision, recall, F1score, and provided comments on the results with respect to the convenience to our case.

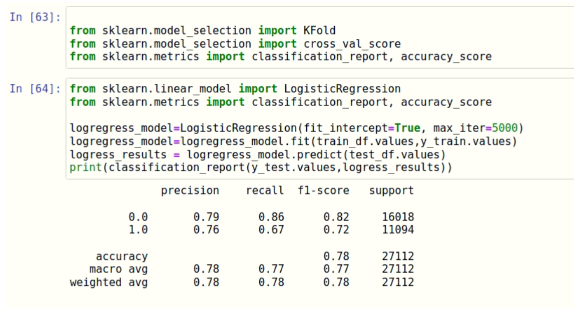
In the project, it is important to stress over the most important performance metrics; in our case, the recall is the most important, even over precision; this is because decreasing the FN (saying that actual offensive/hate speech is not hate speech) is a priority over minimizing FP (saying that normal speech is hate speech). Eventually, we are targeting to maximize both performance metrics, yet, it's still important to set priorities.

A. *Logistic Regression*

Logistic Regression was the first model that came to mind when binary classification was needed. Generally, logistic regression aims to classify different classes using the sigmoid function, which computes the probability of each class and decides on the output class using thresholds.SKLearn offers a logistic regression model (which we used). We ran the logistic regression model on our word2vec embedding with all parameters set to default except for the C, which we set to 1e-5 to control the generalization of our regression model. Like every other algorithm to be used, we calculated the accuracy, precision, recall, and F1 Score.

Logistic regression turned out to be one of the best classifiers to our case, which was expected. This is because of different characteristics of the logistic regression model: First, the logistic regression is very simple in terms of complexity, and so it takes minimal execution time relative to other complex counterparts. Also, it is one of the highest accurate models for detecting Hate/Offensive speech.

Fortunately, the logistic regression model gives a solution since it could minimize the cost function with gradient descent. It can be seen that accuracy does not hinder neither precision nor recall; in other words, it does not diverge from them in the sense that it distorts indication of performance.



```
In [63]:
          from sklearn.model_selection import KFold
          from sklearn.model_selection import cross_val_score
          from sklearn.metrics import classification_report, accuracy_score

In [64]:  from sklearn.linear_model import LogisticRegression
          from sklearn.metrics import classification_report, accuracy_score

          logregress_model=LogisticRegression(fit_intercept=True, max_iter=5000)
          logregress_model=logregress_model.fit(train_df.values,y_train.values)
          logress_results = logregress_model.predict(test_df.values)
          print(classification_report(y_test.values,logress_results))
                        precision    recall  f1-score   support

                 0.0       0.79      0.86      0.82     16018
                 1.0       0.76      0.67      0.72     11094

            accuracy                           0.78     27112
           macro avg       0.78      0.77      0.77     27112
        weighted avg       0.78      0.78      0.78     27112
```

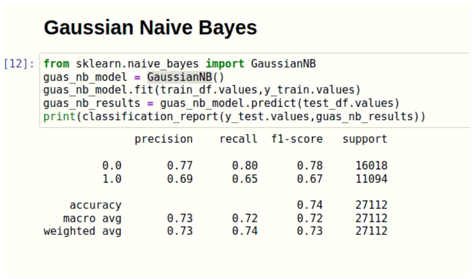
Fig. 10. Logistic Regression Performance Measures.

B. *Naive Bayes*

Naive Bayes can be used as a text classifier based on probabilistic estimates. Since we are using Word2Vec which puts our word corpus in a continuous vector space of 500 dimensions, we are experimenting with the gaussian Naive Bayes. The Gaussian NB follows a normal distribution and uses continuous data, which fits our case.

We did not think that Naive Bayes would be in the top choices for the classification problem of this project. This is mainly because One of the disadvantages of the NB model is it considers the features to be completely independent of each other, however, in our problem they are related by context. Eitherway, we decided to experiment with its most viable options. Unfortunately, the multinomial variation cannot be used because the values are not discrete in our set. We neglected the Bernoulli variation too because our data holds continuous values, and the Bernoulli model deals with binary values only.

Some performance metrics are relatively good, but are not the best. Our most important metric, the recall is overall worse than the other models, also, the class1 recall is lower than that of class 0. The overall metrics are above 70 but still the lowest in all our models.



Gaussian Naive Bayes

```
[12]: from sklearn.naive_bayes import GaussianNB
      guas_nb_model = GaussianNB()
      guas_nb_model.fit(train_df.values,y_train.values)
      guas_nb_results = guas_nb_model.predict(test_df.values)
      print(classification_report(y_test.values,guas_nb_results))

                    precision    recall  f1-score   support

               0.0       0.77      0.80      0.78     16018
               1.0       0.69      0.65      0.67     11094

          accuracy                           0.74     27112
         macro avg       0.73      0.72      0.72     27112
      weighted avg       0.73      0.74      0.73     27112
```

Fig. 11. Naive Bayes Performance Measures.

C. Decision Trees

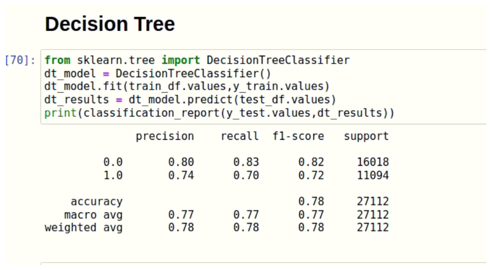
Decision Trees are a choice of considerable significance in classification problems with large datasets; they are one of the easiest to interpret, and popular to implement. It is of considerable performance in classifying complicated patterns; thus, we thought it can be a wise choice to try as it seems to fit our case's problem. We utilized the SKLearn Decision Tree model.

We tried running the model with default parameters. Looking at the performance measures, our most important, which is the recall, had relatively good results, and same was with the precision- see fig. The weighted precision, recall, and F1 score are comparable to the lowest two of the top three models: logistic regression and SVM. It is closely performing to the logistic regression in the top three but has a slightly lower class 1 recall, and this is why we did not put it in the top three, but chose to analyze it like them because the performance was close to them. It is like the same average performance, but the logistic regression is slightly leaning to the side we care about more.

The accuracy was relatively low; thus, it can be deduced that the tree could not be constructed in a way that interpreted the class accurately. One of the cons of such a model is the training time, which is high, as the construction of the tree takes a lot of computing time.

D. Perceptron

A Perceptron is the atomic unit of a Neural Network, and is a model by itself; thus, it is the simplest form of a neural network. It applies an activation function (here is a logistic regression one) to a number of inputs and pushes its output to the next layer of neurons in a neural network; however, as a model itself, we take its output as the final output, and treat it as a binary classifier (outputs class 1 or 0). Generally, it is not a complex model, is weak, and underfits large data sets with a lot of features. We thought of testing it alone because we are considering the neural networks option, and thought of experimenting with it as a classifier on its own.

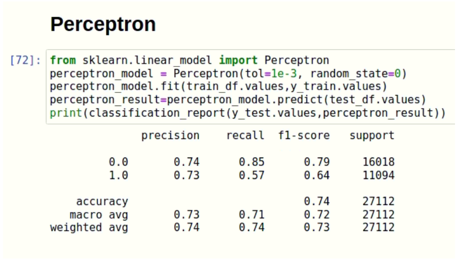


Decision Tree

```
[70]: from sklearn.tree import DecisionTreeClassifier
      dt_model = DecisionTreeClassifier()
      dt_model.fit(train_df.values,y_train.values)
      dt_results = dt_model.predict(test_df.values)
      print(classification_report(y_test.values,dt_results))

                    precision    recall  f1-score   support

               0.0       0.80      0.83      0.82     16018
               1.0       0.74      0.70      0.72     11094

          accuracy                           0.78     27112
         macro avg       0.77      0.77      0.77     27112
      weighted avg       0.78      0.78      0.78     27112
```

Fig. 12. Decision Tree Performance Measures.

As seen in the above table, the performance measures of the perceptron alone are bad, which is expected as it does not fit the problem's criteria well. It has low accuracy, recall bias, low recall, and low precision. see fig13.



Perceptron

```
[72]: from sklearn.linear_model import Perceptron
      perceptron_model = Perceptron(tol=1e-3, random_state=0)
      perceptron_model.fit(train_df.values,y_train.values)
      perceptron_result=perceptron_model.predict(test_df.values)
      print(classification_report(y_test.values,perceptron_result))

                    precision    recall  f1-score   support

               0.0       0.74      0.85      0.79     16018
               1.0       0.73      0.57      0.64     11094

          accuracy                           0.74     27112
         macro avg       0.73      0.71      0.72     27112
      weighted avg       0.74      0.74      0.73     27112
```

Fig. 13. Perceptron Performance Measures.

E. Support Vector Machine (SVM)

Support Vector Machine (SVM) is one of the simplest linear models for classifying linear and non-linear problems. The idea behind SVM is very simple. The algorithm tries to find a line or a hyperplane with the maximum margin, that is the distance that the line can move before it hits a point of one of the two classes. For instance, in figure 14, svm chooses the yellow line over the green one as it has larger margin (Pupale, 2019).

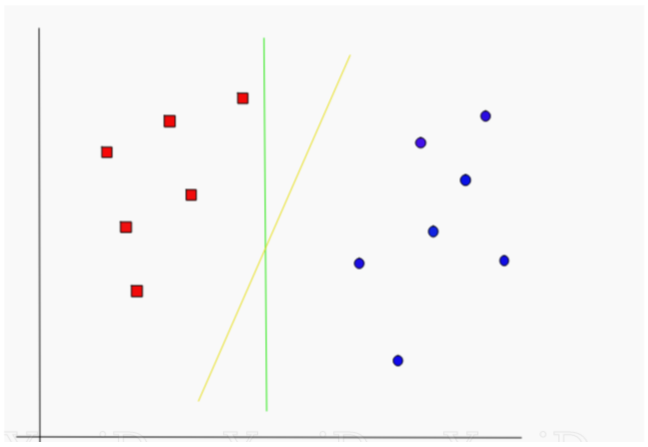
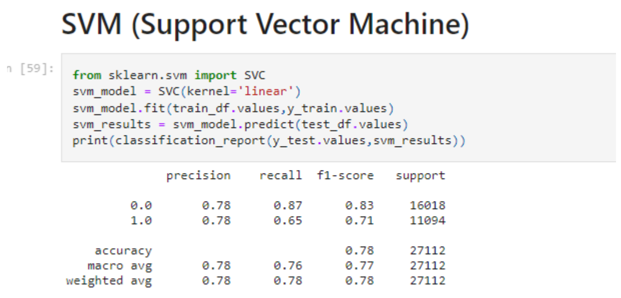


Fig. 14. Lines Drawn by SVM.

When it comes the performance of SVM, it is better in class 0 recall but the Neural Network's recall is way better than this of the SVM as SVM provides very low recall regarding class 1 as shown in Fig.15.



SVM (Support Vector Machine)

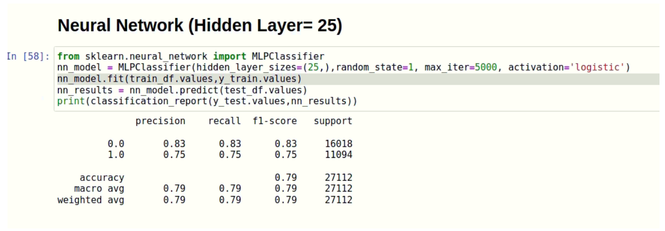
```
In [59]: from sklearn.svm import SVC
         svm_model = SVC(kernel='linear')
         svm_model.fit(train_df.values,y_train.values)
         svm_results = svm_model.predict(test_df.values)
         print(classification_report(y_test.values,svm_results))

              precision    recall  f1-score   support

         0.0       0.78      0.87      0.83     16018
         1.0       0.78      0.65      0.71     11094

    accuracy                           0.78     27112
   macro avg       0.78      0.76      0.77     27112
weighted avg       0.78      0.78      0.78     27112
```

Fig. 15. SVM Performance Measures.

F. Neural Network

In this experiment we tried a neural network implementation, which consists of multiple layers of the perceptron model that feed their outputs to each other. Since this is a binary classification problem, our activation function is a logistic regression function. It had a maximum activation limit of 5000. The neural network is the most complex, and most powerful model that we tried. It highly fits the problem considering its size, complexity, and need to consider context before classifying. It puts weights in the hidden layers to handle the large number of features. We used the MLP implementation of SKLearn with default parameters.

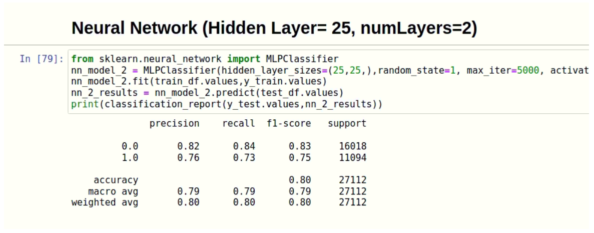
It can be easily noticed that the performance metrics are actually good, they are the highest compared to previous models. There is much less recall bias. The recall, our most important metric, is the highest among the other metrics (fits our criteria). The only disadvantage of the neural network in our case is the time needed to train, which is natural given the complexity of the model. We tried diffent values for hidden_layer_size, that is number of neurons in each hidden layer (500,200,100,50,25), and the best was size of 25 neurons. We also experimented number of hidden layers for this 25-neuron hidden layer network.



Neural Network (Hidden Layer= 25)

```
In [58]: from sklearn.neural_network import MLPClassifier
         nn_model = MLPClassifier(hidden_layer_sizes=(25,),random_state=1, max_iter=5000, activation='logistic')
         nn_model.fit(train_df.values,y_train.values)
         nn_results = nn_model.predict(test_df.values)
         print(classification_report(y_test.values,nn_results))
              precision    recall  f1-score   support

         0.0       0.83      0.83      0.83     16018
         1.0       0.75      0.75      0.75     11094

    accuracy                           0.79     27112
   macro avg       0.79      0.79      0.79     27112
weighted avg       0.79      0.79      0.79     27112
```

Fig. 16. Neural Network One Hidden Layer Performance Measures.



Neural Network (Hidden Layer= 25, numLayers=2)

```
In [79]: from sklearn.neural_network import MLPClassifier
         nn_model_2 = MLPClassifier(hidden_layer_sizes=(25,25,),random_state=1, max_iter=5000, activat
         nn_model_2.fit(train_df.values,y_train.values)
         nn_2_results = nn_model_2.predict(test_df.values)
         print(classification_report(y_test.values,nn_2_results))
              precision    recall  f1-score   support

         0.0       0.82      0.84      0.83     16018
         1.0       0.76      0.73      0.75     11094

    accuracy                           0.80     27112
   macro avg       0.79      0.79      0.79     27112
weighted avg       0.80      0.80      0.80     27112
```

Fig. 17. Neural Network Two Hidden Layers Performance Measures.

VI. Comparative Analysis For The Top Three Models

In the experimental analysis, we did tried several models and different variations for some of the models; since we decided to neglect the models that clearly don't fit our problem, we didn't see a single model perform terribly bad and some models that are not in the top three are close to them, but the top three still differentiate from others. In fact, they most importantly differentiate in the performance metric we care about the most, the recall. The top performing models were:

1) Neural Network.
2) Support Vector Machine (SVM)
3) Logistic Regression

Comparing their performance metrics we can see that for out most important performance metric, the recall, for class 0 (not hate speech) and 1 (hate speech) respectively for each model appear as follows Logistic Regression: 0.86, 0.67 (fig 10) SVM: 0.87, 0.65 (fig 15) Neural Network: 0.84, 0.73 (fig 17)

The class 0 recall differences are not much between the top three, with the SVM leading the models. If we look at the class 1 recall we can see that the neural network is better than SVM and logistic regression. We are more interested (put more significance logically) over the class 1 recall, as it is more dangerous to classify hate speech as not hate speech. So, even though SVM and Logistic regression are a bit higher in class 0, the Neural network model is better for our case as it is better than the other 2 (with a higher difference) in class 1 recall.

Looking at weighted avg precision for all of them, we can see that the neural network is ahead of the SVM and logistic regression (0.8 vs 0.78). The exact same goes for the F1 Score: neural networks have a 0.8 score vs 0.78 for its two other counterparts. For Accuracy, the neural network is also slightly better with an 80% vs a 78% accuracy of its counterparts.

VII. Conclusion

After preprocessing, doing feature correlation. We did Word2Vec text embedding to optimize memory use and put our corpus in a vector form that can be fed into the five classifiers. After running experiments for five considered classifier models and variated some parameters to maximize performance metrics, we observed the results and made comparisons between the different models.

We decided to proceed with the neural network model as it fits all our criteria: it maximizes the recall, and has overall high performance metrics, puts weights and can handle the large number of features coming out of the text embedding that we did. It is to be implemented with a sigmoid activation function to act as a binary classifier, we are still to experiment more with it to optimize the parameters and weights in order to reach better performance metrics.

References

[1] Pupale, R. (2019, February 11). Support vector machines(svm) - an overview. Medium. Retrieved March 20, 2022, from https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989: :text=SVM %20or%20Support%20Vector%20Machine,separates%20the %20data%20into%20classes.