IDS 566 Advanced Text Analytics Business Project

Analysis of 20 Newsgroups Dataset Using Various Classification Models

1. INTRODUCTION

One of the widely used natural language processing task in different business problems is "Text Classification". Our goal for this project is to classify the text documents into defined categories. The dataset we are using is "The 20 Newsgroups data". The 20 Newsgroup data is the collection of approximately 20,000 newsgroup documents, divided into 20 different newsgroups. We are using the scikit-learn API to fetch the data for this task.

Project outline:

1. Necessary pre-processing steps before all models
    a. Challenges faced
2. Methodology
    a. Types of models
    b. Text processing taking place for each model
3. Results and Discussion

2. EXPLORATORY DATA ANALYSIS (EDA)

We decided to sort the EDA process into two categories: general pre-processing steps that were common across all vectorizers and models and certain pre-processing steps that we put as options to measure model performance with or without them. Accuracy was chosen as a measure of comparison between models since greater the accuracy, better the model performance on test data.

The following general pre-processing steps were carried out since any document being input to a model would be required to be in a certain format:

1. Converting to lowercase
2. Removal of stop words
3. Removing alphanumeric characters
4. Removal of punctuations
5. Vectorization: TfIdfVectorizer was used. The model accuracy was compared with those that used CountVectorizer. In all cases, when TfIDFVectorizer was used, it gave better results and hence was chosen as the default Vectorizer.

The following steps were added to the pre-processing steps as optional to see how model performance changed with and without these steps:

1. Stemming
2. Lemmatization
3. Using Unigrams/Bigrams

We analyzed the spread of output labels in the training data to check for whether it was skewed towards any of the class. Figure 1 below shows this class distribution. As seen in the figure, the spread of variables is reasonably even thus showing a near equal distribution of each output variable, hence the classification models won't be biased towards any particular class and subsequently no resampling was required. Also, from Figure 2 we see that the class distribution in the test.

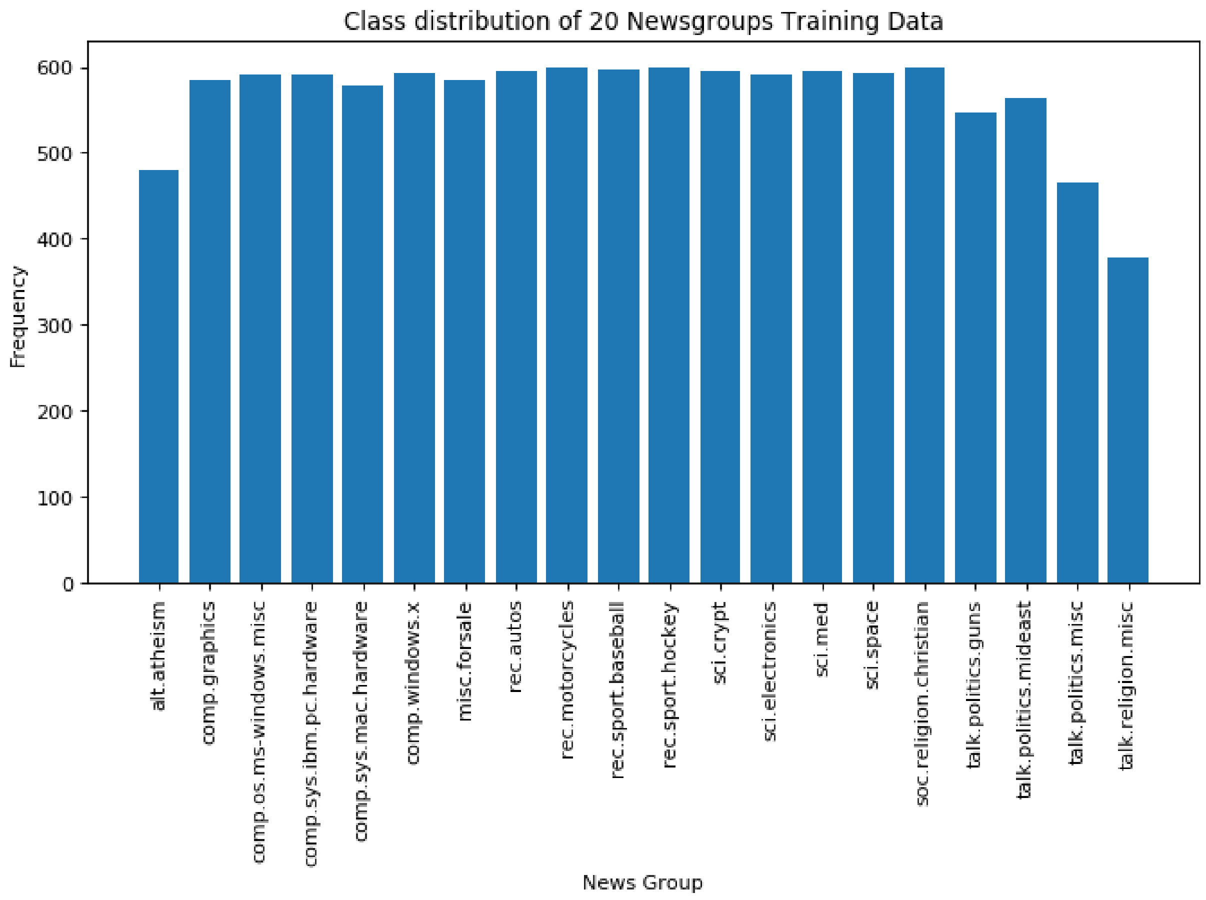


Figure 1. Class Distribution of 20 Newsgroups in Training Dataset

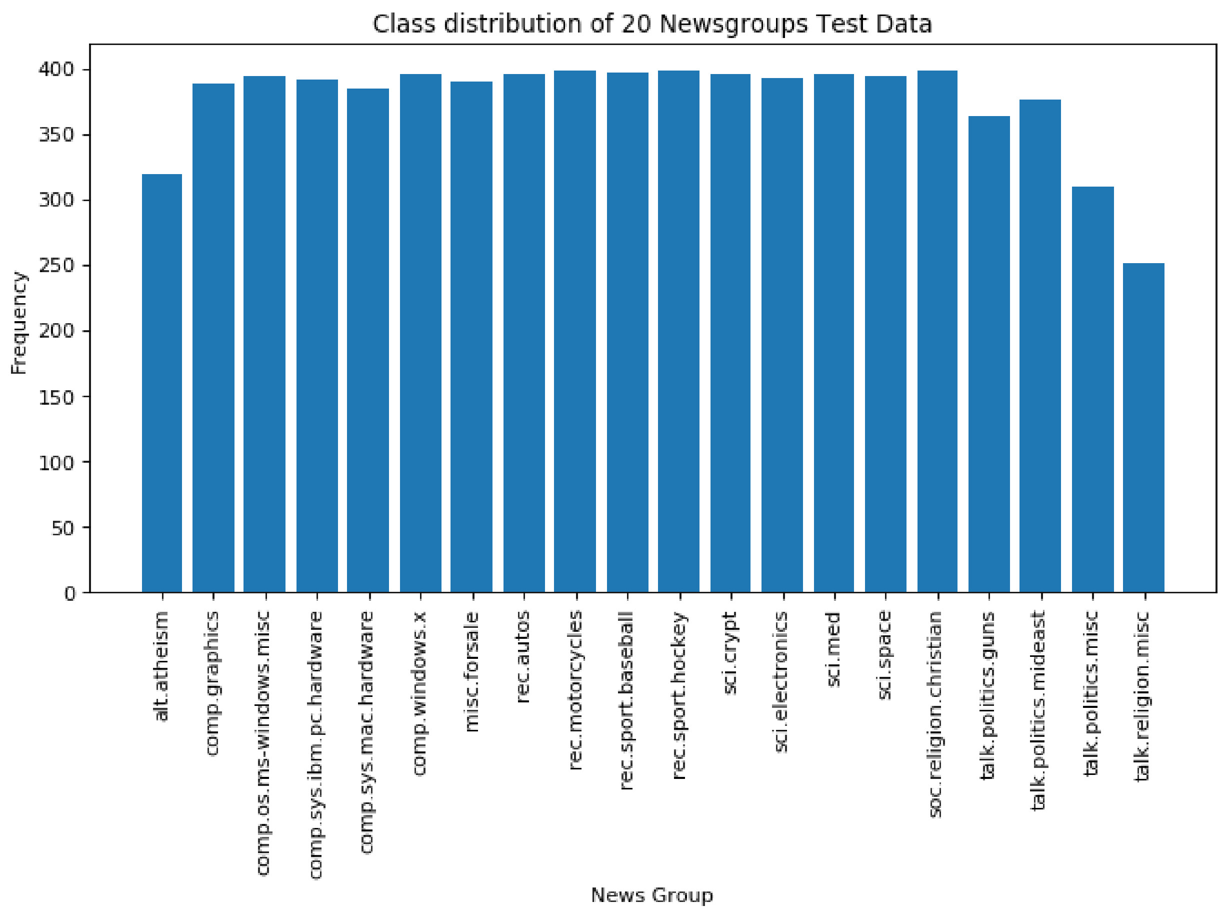


Figure 2. Class Distribution of 20 Newsgroups in Testing Dataset

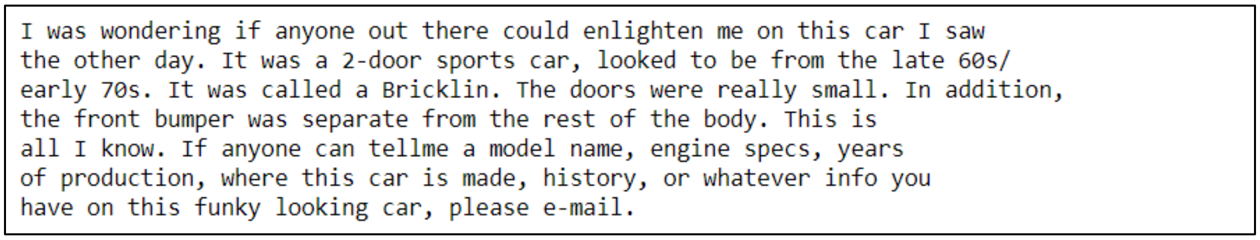
Further, the data (text) that was available had header, footer and quotes included, and looked something like:

```
From: lerxst@wam.umd.edu (where's my thing)
Subject: WHAT car is this!?
Nntp-Posting-Host: rac3.wam.umd.edu
Organization: University of Maryland, College Park
Lines: 15

 I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.

Thanks,
- IL
   ---- brought to you by your neighborhood Lerxst ----
```

We extracted the main body of text by removing header, footer and quotes and eventually the output looked something like:



```
I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.
```

We were getting better accuracies with the initial version (~10% more) but we believe that it was because of the bias introduced into the data. Most writers write on a particular set of topics and our models were using that information also to classify. This meant that the classifier gave less weightage to what exactly the text talked about. This could be a major issue when we see a test data that doesn’t have such information (author, server, organization) included. So, to train a good general model that takes into account only the words used we did the extraction.

3. METHODOLOGY

In this study, we compare various features of TF-IDF Vectorizer to find the set which works the best under different classification models. We start the modeling procedure by first considering no stemming and lemmatization in the vectorizer. Next, we will apply only stemming and lastly, we will apply only lemmatization in the vectorizer. Within each of the three settings we first use only unigrams as the resulting features from the TF-IDF Vectorizer and then compare it with the results of when both unigrams and bigrams are considered in the feature set. Subsequently, for all of these combinations we consider four different classification models to examine how they perform on the test data. These include Naive Bayes, Logistic Regression, Stochastic Gradient Descent Classifier and k Nearest Neighbors. The analysis is performed using sklearn library in Python.

3.1 Not performing Stemming and Lemmatization in the TF-IDF Vectorizer

With TF-IDF Vectorizer we can extract "bag of words" from the text and apply TF-IDF (term frequency - inverse document frequency) weights. By trials, we found that applying sublinear *tf* scaling (1 + log(*tf*)) improves overall results. The TF-IDF Vectorizer created a Document Term Matrix having ~11000 documents in the training set and ~68,000 words (without stemming/lemmatization).

We first extract only unigrams from the text data to form the vectorizer. We then test four classifiers: Multinominal Naive Bayes, Logistic Regression, Stochastic Gradient Descent

Classifier and k Nearest Neighbors. For Logistic Regression and k Nearest Neighbors we used grid search to tune their parameters.

For Logistic Regression, the parameter ‘penalty’ was tuned with values {‘L1’ and ‘L2’} norm. Best performance of Logistic Regression is achieved when ‘L2’ norm is used in the penalization. For k Nearest Neighbors, the parameter ‘n_neighbors’ was tuned with values {5, 10, 100 and 200}, and parameter ‘weights’ was tuned with values {‘uniform’ and ‘distance’}. Using ‘n_neighbors’ as 5 and ‘weights’ as ‘distance’ gave the best performance for k Nearest Neighbors.

Next, we extract unigrams as well as bigrams from the text data to form the TF-IDF Vectorizer. Again, we test the four classifiers with similar grid search approach for Logistic Regression and k Nearest Neighbors. The test data performance results obtained are shown in Table I.

Table I. Performance of classifiers on test data

ngrams	**Classifier Accuracy (%)**			
	MultinomialNB	**LogisticRegression**	**SGDClassifier**	**KNeighborsClassifier**
Unigram	66.93	68.96	69.67	8.83
Unigram+Bigram	65.57	68.34	70.39	7.73

3.2 Applying only Stemming in the TF-IDF Vectorizer

Now we use stemming technique, i.e., cutting the words to their root form. We use Snowball English Stemmer algorithm from the NLTK package in Python and we defined a class to represent the Snowball Stemmer. We again test the four classifiers, first with only unigrams and then with both unigrams and bigrams in the vectorizer.

Again, performing grid search on Logistic Regression gave best performance when ‘L2’ norm was used in the penalization. Also, k Nearest Neighbors gave best performance when ‘n_neighbors’ was tuned to 5 and ‘weights’ to ‘distance’. The test data performance results obtained are shown in Table II.

Table II. Performance of classifiers on test data

ngrams	**Classifier Accuracy (%)**			
	MultinomialNB	**LogisticRegression**	**SGDClassifier**	**KNeighborsClassifier**
Unigram	66.49	69	70.18	8.42
Unigram+Bigram	65.59	68.34	70.66	8.5

3.3 Applying only Lemmatization in the TF-IDF Vectorizer

Finally, we apply lemmatization, i.e. getting grammatically correct normal form of the word with the use of morphology. We use Word Net Lemmatizer from NLTK package and part-of-speech word tagging and we define a class to represent Lemma Tokenizer. We again test the four classifiers, first with only unigrams and then with both unigrams and bigrams in the vectorizer.

Again, performing grid search on Logistic Regression gave best performance when ‘L2’ norm was used in the penalization. Also, k Nearest Neighbors gave best performance when ‘n_neighbors’ was tuned to 5 and ‘weights’ to ‘distance’. The test data performance results obtained are shown in Table III.

Table III. Performance of classifiers on test data

ngrams	**Classifier Accuracy (%)**			
	MultinomialNB	**LogisticRegression**	**SGDClassifier**	**KNeighborsClassifier**
Unigram	66.87	69.01	69.92	8.71
Unigram+Bigram	65.72	68.36	70.58	8.03

3.4 *Results and Discussions*

We now compare the results that we achieved in the previous sub sections. In Figures 3 to 6 we visualize the performance of each classifier. From Figures 3 and 6 we see that classifiers Multinominal Naïve Bayes and k Nearest Neighbors perform best when neither stemming nor lemmatization is done and only unigrams are extracted in the TF-IDF Vectorizer. From Figure 4 we see that Logistic Regression has the best performance when Lemmatization is used with only unigrams are extracted in the TF-IDF Vectorizer. Finally, from Figure 5 we see that Stochastic Gradient Descent Classifier has the best performance when Stemming is used with both unigrams and bigrams are extracted in the TF-IDF Vectorizer.

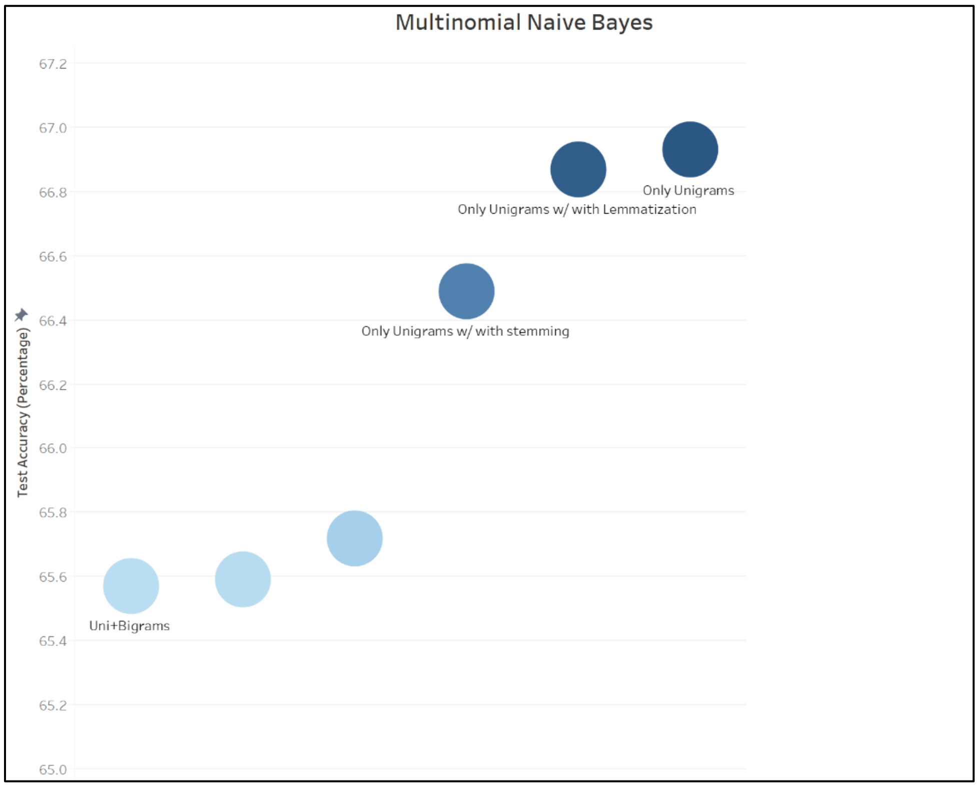


Figure 3. Multinominal Naïve Bayes Performance

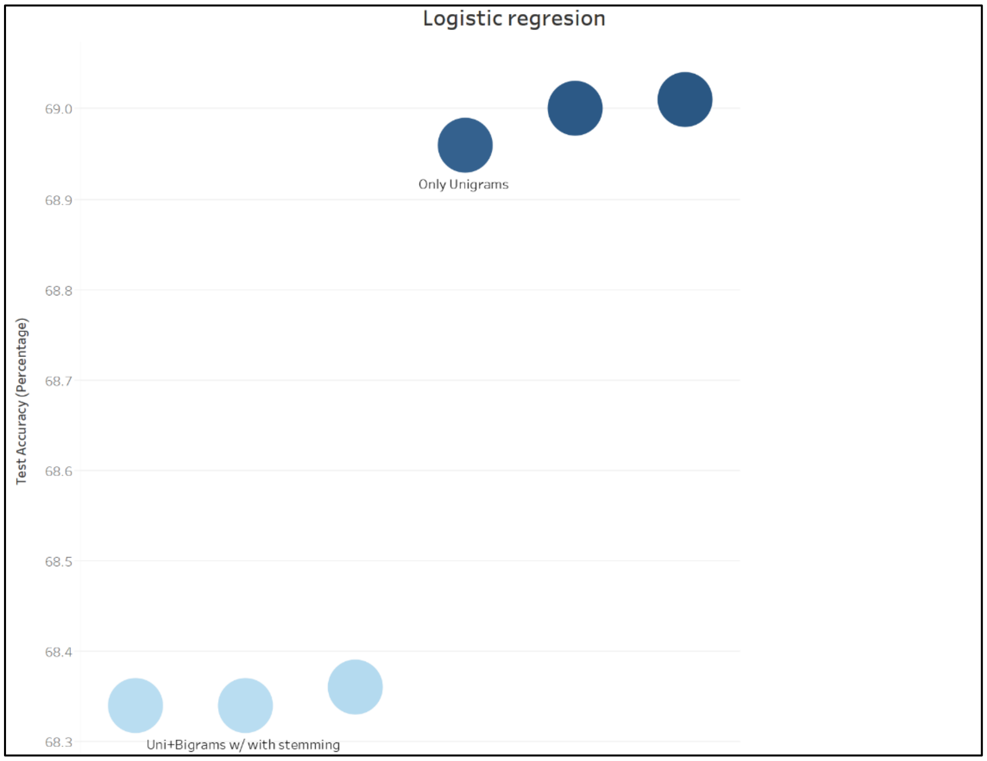


Figure 4. Logistic Regression Performance

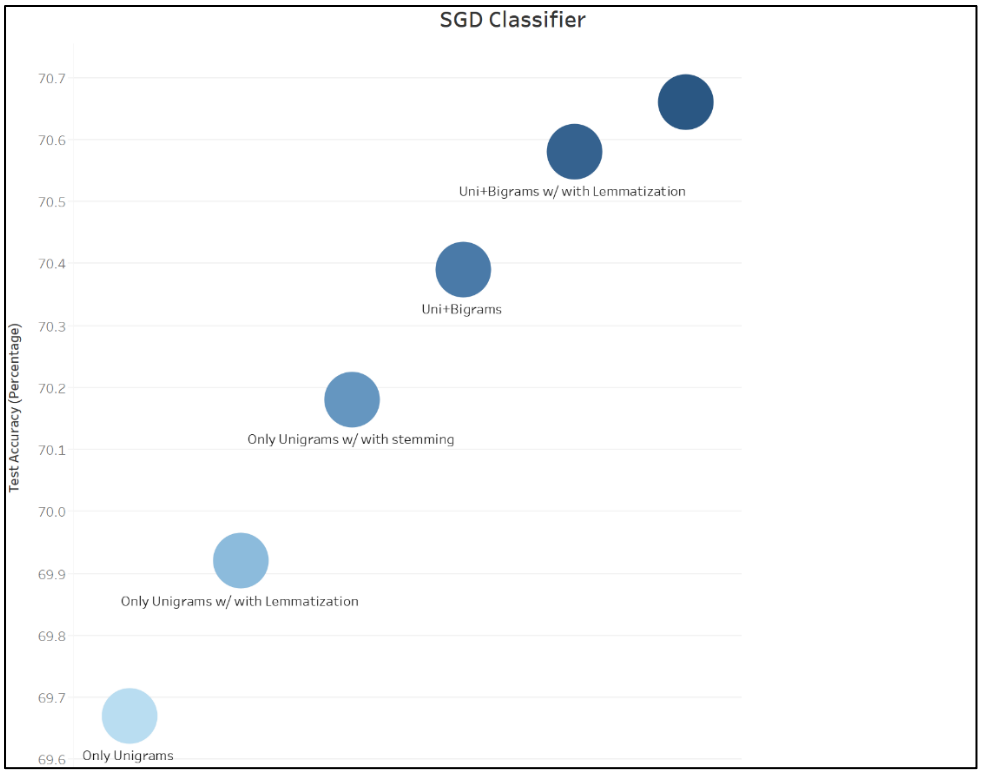


Figure 5. SGD Classifier Performance

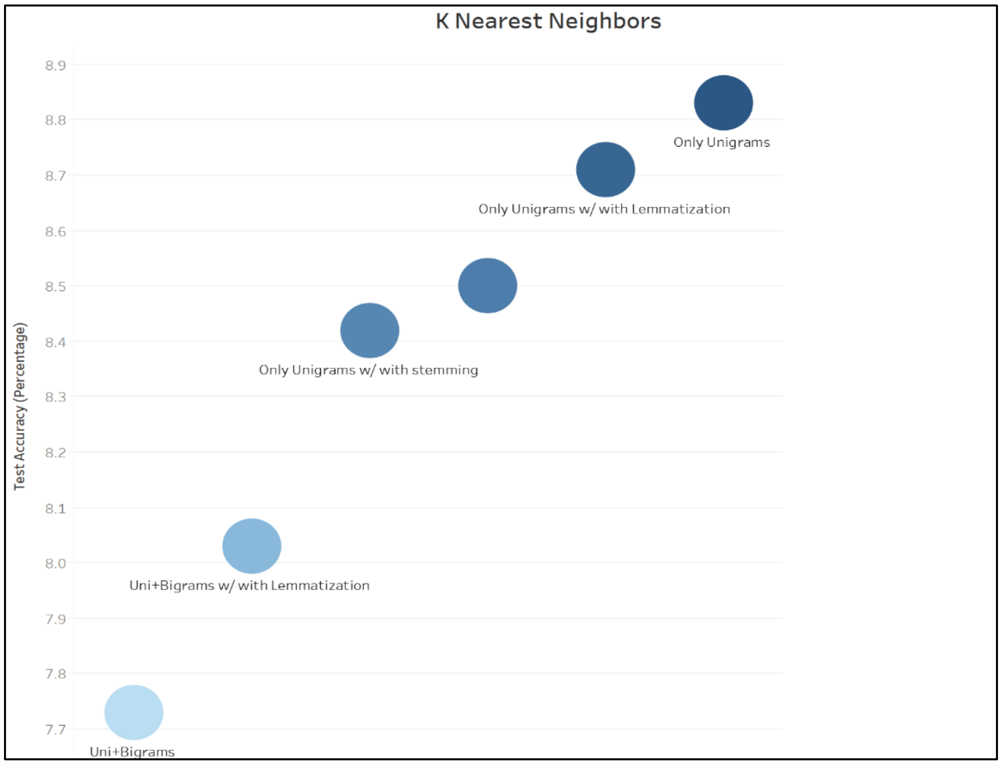


Figure 6. k Nearest Neighbors Performance

The best performance obtained for each classifier is shown in Table III and also visualized in Figure 7. We see that the highest accuracy of 70.66% is achieved by Stochastic Gradient Descent Classifier, followed by Logistic Regression and Multinominal Naïve Bayes. We can see that k Nearest Neighbors performs extremely poorly. This is due to the fact that k Nearest Neighbors is known to suffer from curse of dimensionality when number of features is large. It's distance measure becomes meaningless when the dimension of the data increases significantly. The table also shows performance of neural networks (has 2nd best performance), which is described in more detail in Section 3.5. Finally, in Figure 8 we display the performance metrics such as precision, recall and f1-score for each of the 20 classes for the best performing Stochastic Gradient Descent Classifier.

Table III. Summarizing the best performance from each classifier

Classifier	**Model**	**Test Accuracy (%)**
MultinomialNB	No Stemming and Lemmatization + Unigrams	66.93
LogisticRegression	With Lemmatization + Unigrams	69.01
SGDClassifier	With Stemming + Unigrams and Bigrams	70.66
KNeighborsClassifier	No Stemming and Lemmatization + Unigrams	8.83
Neural Networks	With Lemmatization + Unigrams	69.30

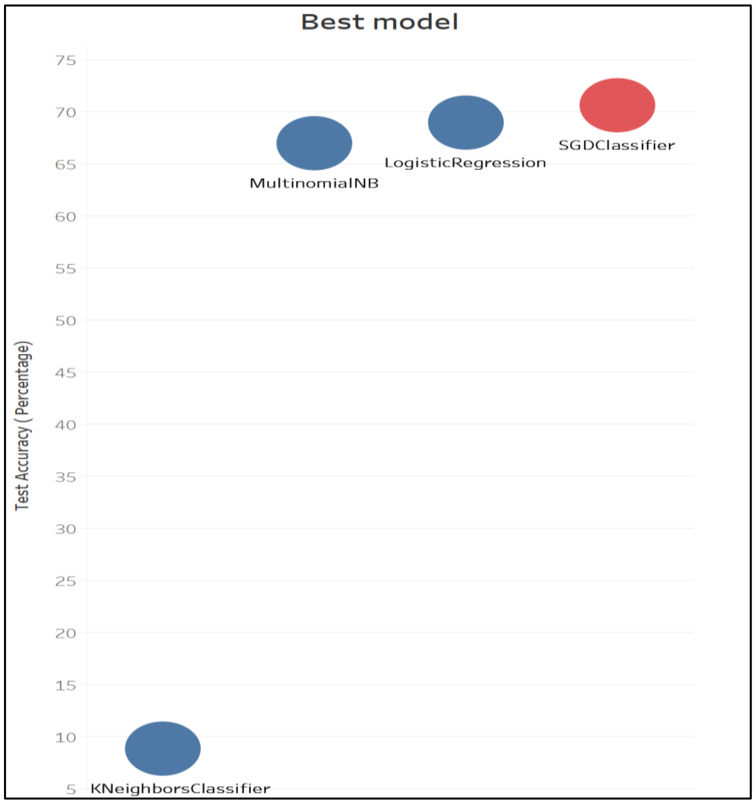
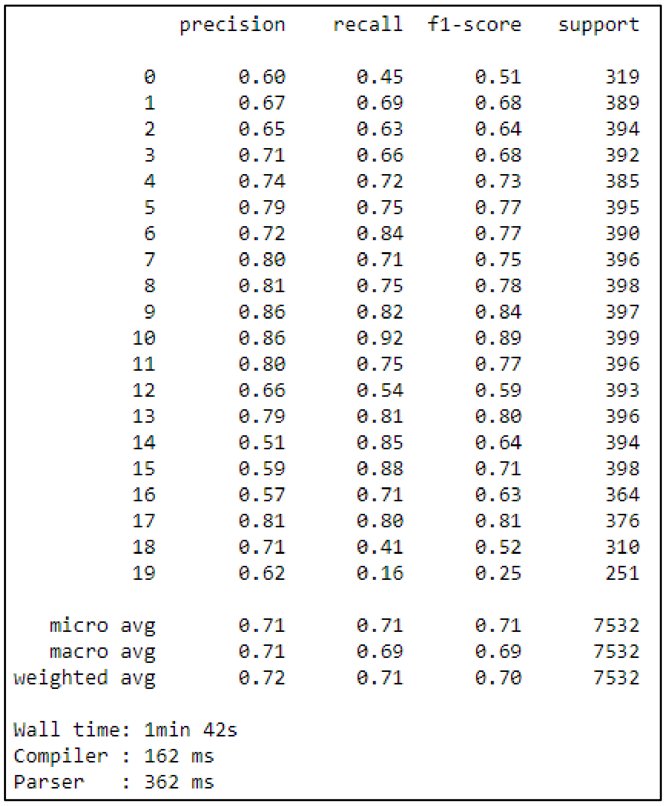


Figure 7. Comparing the best performance of each classifier



```
              precision    recall  f1-score   support

           0       0.60      0.45      0.51       319
           1       0.67      0.69      0.68       389
           2       0.65      0.63      0.64       394
           3       0.71      0.66      0.68       392
           4       0.74      0.72      0.73       385
           5       0.79      0.75      0.77       395
           6       0.72      0.84      0.77       390
           7       0.80      0.71      0.75       396
           8       0.81      0.75      0.78       398
           9       0.86      0.82      0.84       397
          10       0.86      0.92      0.89       399
          11       0.80      0.75      0.77       396
          12       0.66      0.54      0.59       393
          13       0.79      0.81      0.80       396
          14       0.51      0.85      0.64       394
          15       0.59      0.88      0.71       398
          16       0.57      0.71      0.63       364
          17       0.81      0.80      0.81       376
          18       0.71      0.41      0.52       310
          19       0.62      0.16      0.25       251

   micro avg       0.71      0.71      0.71      7532
   macro avg       0.71      0.69      0.69      7532
weighted avg       0.72      0.71      0.70      7532

Wall time: 1min 42s
Compiler : 162 ms
Parser   : 362 ms
```

Figure 8. Performance metrics for the best SGD Classifier

3.5 Neural Networks

We also studied how the Neural Networks perform on this data. We used the same pre-processing for the neural network that was used for the other models in previous sub sections, but restricted the min_df threshold in the TF-IDF Vectorizer to 0.0005. This meant that anything that is in less than 0.05% of documents will be removed when vectorizing. The main reason to do this was to speed the training of neural networks because it was taking a significant time with 67,822 features. Setting the min_df threshold reduced the features to about 11,445. We also used one hot encoding on the labels.

We tried multiple neural networks with different number of nodes, hidden layers, min_df, and optimizers. For hidden layers, if we added too many hidden layers, the model was overfitting on the training data, giving us accuracies of more than 90% but poor accuracies on the validation data. For min_df we tried multiple values between 0.0001 and 0.05, but 0.0005 gave us the best result and good training speed.

The final model that we used had 1 hidden layer with 1500 nodes. The classifier was 'adam', the loss was 'categorical cross entropy'. We also did 2 callbacks, Model check point (that saves the best model) and Early stopping (that stopped the training if the loss did not improve for for 3 epochs). The training/validation accuracy and loss across epochs looked like as shown in Figure 8. And finally using the best saved weights our model was able to achieve an accuracy of 69.30%.

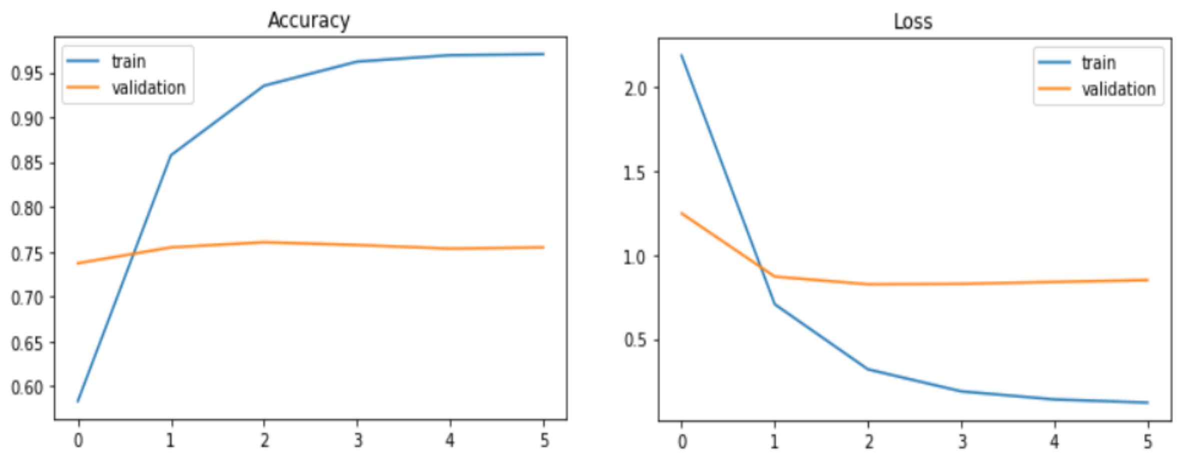


Figure 9. Training/validation accuracy and loss across epochs

4. CONCLUSION

Overall, we saw that the best performing classifiers were, Stochastic Gradient Descent classifier, Neural Networks and Logistic Regression. We also noticed that using TF-IDF Vectorizer gave better results than Count Vectorizer. The best accuracy for Stochastic Gradient Descent was achieved by using Stemming along with both unigrams and bigrams extracted in TF-IDF Vectorizer. For neural networks better results were achieved using Lemmatization and limiting the min_df to 0.0005 significantly increased the speed of training. Also, unigrams helped us achieve high accuracy most of the time, so we don't really need to extend the features and increase the time taken by the classifiers. Finally, we were able to achieve an accuracy of 70% for 20 classes using the techniques learned in class and training different models and optimizing their parameters using grid search.