Artificial Intelligence

COSC 6368 Spring 2020

Final Project

Text Analytics on News group articles using Tokenization Tools

Team:

Sravanthi Pasumarthi (1868536)

Sai Sri Varsha Gadde (1885164)

SURVEY:

One of the widely used task in different business problems is “Text Analytics”. 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.

Text analytics is the process of extracting information and uncovering actionable insights from unstructured text. Text analytics allows data scientists and analysts to evaluate content to determine its relevancy to a specific topic. Researchers mine and analyze text by leveraging sophisticated software developed by computer scientists. Text Analytics also involves applying of statistical and machine learning techniques to be able to predict /prescribe or infer any information from the text data.

Our aim for this project is to use Machine Learning and Neural Network and Natural Language Processing to build a model which extracts information from text, such as tweets, emails, support tickets, product reviews, and survey responses and classifies text documents into defined categories. We are trying to classify the text collected from various sources compiled into the dataset “The 20 Newsgroups data”. Some of the class labels are graphics, hardware, sports, autos, motorcycles, guns, medicine and crypto.

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.

Literature Review:

Artificial intelligence (AI) is wide-ranging branch of computer science concerned with building smart machines capable of performing tasks that typically require human intelligence. It makes it possible for machines to learn from experience, adjust to new inputs and perform human-like tasks. The ideal characteristic of artificial intelligence is its ability to rationalize and take actions that have the best chance of achieving a specific goal.

AI is used in decision making by considering the machines the real-time scenario. An Artificially Intelligent machine reads the real-time data, understands the business scenario and reacts accordingly. The process of transforming a computer to a computer-controlled robot or designing software that thinks and reacts exactly the way a human being thinks.

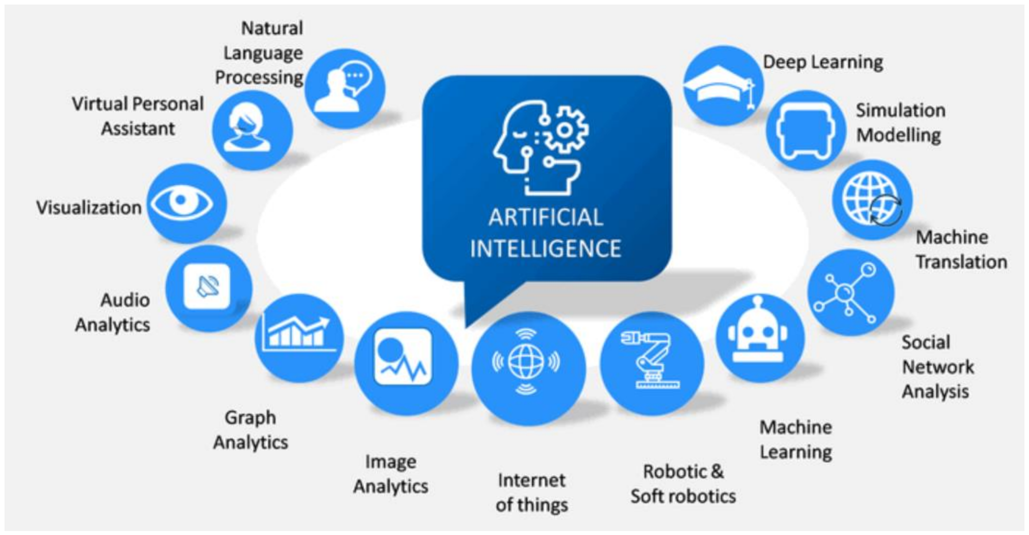


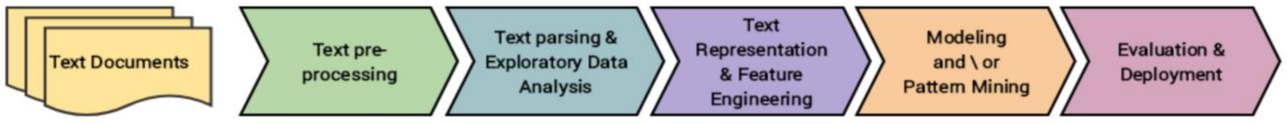
Figure: Artificial Intelligence and its applications

Machine learning is an application of AI that provide system the ability to automatically learn and improve from experience. It focuses on the development of computer programs that can access data and use it learn for themselves. It is used to induce regularities and patterns, providing easier implementation with low computation cost, as well as fast training, validation, testing, and evaluation, with high performance compared to physical models, and relatively less complexity. The continuous advancement of these methods over the last two decades demonstrated their suitability for sales forecasting and product predictions.

Machine learning offers a number of tools that provide rich snippets of data which can be applied to both unstructured and structured data. The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide.

Natural Language Processing is a branch of artificial intelligence that deals with the interaction between computers and humans using the natural language**.** Natural language processing systems can analyze unlimited amounts of text-based data without fatigue and in a consistent, unbiased manner. They can understand concepts within complex contexts to extract key facts and relationships or provide summaries. Given the huge quantity of unstructured data that is produced every day, from electronic health records (EHRs) to social media posts, this form of automation has become critical to analyzing text-based data efficiently.
Text mining is an artificial intelligence (AI) technology that uses natural language processing (NLP) to transform the free (unstructured) text in documents and databases into normalized, structured data suitable for analysis or to drive machine learning (ML) algorithms.

We have used some feature engineering techniques in NLP like TF-IDF Vectorizer and Count Vectorizer for tokenization in order to make the data ready for fitting the models. These are most computationally intensive steps in text analytics.


Text Documents
Text pre-processing
Text parsing & Exploratory Data Analysis
Text Representation & Feature Engineering
Modeling and \ or Pattern Mining
Evaluation & Deployment

Machine Learning algorithms have important characteristics that need to be carefully taken into consideration. The first is that they are as good as their training, whereby the system learns the target task based on past data. If the data is scarce or does not cover varieties of the task, their learning falls short, and hence, they cannot perform well when they are put into work. ML Algorithms that are using as part of this project for text analytics are as follows.

Logistic Regression:

Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. It is basically a supervised classification algorithm. In a classification problem, we analyze a dataset in which there are one or more independent variables that determine an outcome. It is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables. It is used to estimate the probability of an event occurring having been given some previous data.

Logistic regression analysis studies the association between a categorical dependent variable and a set of independent (explanatory) variables. Logistic regression is used when the dependent variable has only two values, such as 0 and 1 or Yes and No.

Multinomial Naïve Bayes:

Naive Bayes classifier is a simple probabilistic classifier which is based on Bayes theorem with strong and naïve independence assumptions. It is one of the most basic text classification techniques with various applications in email spam detection, personal email sorting, document categorization.

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable, whereas Multi nominal estimates the conditional probability of a particular word/term/token given a class. Multinomial distribution describes the probability of observing counts among a number of categories.

Multinomial estimates the conditional probability of a particular word given a class as the relative frequency of term t in documents belonging to class. The variation takes into account the number of occurrences of term t in training documents from class, including multiple occurrences. Multinomial Naive Bayes classifier considers feature vectors which are representation of the frequencies with which few events have been generated by a multinomial distribution.

Stochastic Gradient Descent:

The word 'stochastic' means a system or a process that is linked with a random probability. In Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. Gradient descent is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient. In machine learning, we use gradient descent to update the parameters of our model. Algorithm iteratively makes small adjustments to a machine learning network configuration to decrease the error of the network. In this order of the training dataset is randomized, one iteration of the gradient descent algorithm requires a prediction for each instance in the training dataset.

Neural Networks:

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates.

Neural networks are multi-layer networks of neurons that we use to classify things, make predictions, etc.

Neural networks can adapt to changing input, so the network generates the best possible result without needing to redesign the output criteria. These networks can learn and model the relationships between inputs and outputs that are complex and nonlinear.

Neural networks consist of input and output layers, as well as (in most cases) a hidden layer consisting of units that transform the input into something that the output layer can use. They are excellent tools for finding patterns which are far too complex or numerous for a human programmer to extract and teach the machine to recognize. Neural networks are a class of machine learning algorithms used to model complex patterns in datasets using multiple hidden layers and non-linear activation functions. Each neuron's coefficients (weights) are then adjusted relative to how much they contributed to the total error.

Tokenization

Tokenization is a technique in NLP (Natural Language Processing) which splits the document into tokens which are words or phrases. These tokens are used for further processing. Tokenization is useful for identifying meaningful keywords. Tokens are separated by white spaces, line breaks or punctuation marks. White spaces and punctuations do not include in the resulting tokens.

System Requirements:

- Python (3.6 version).
- Anaconda.
- Jupyter Notebook.

We also used python modules like Sci-kits learn module, Sklearn library, Keras library and Tensor Flow Library for our project.

IMPLEMENTATION:

Dataset:

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.

Dataset contains two columns a reference to the document_id number and the newsgroup associated with. There are also 20 files that contain all of the documents, one document per newsgroup.

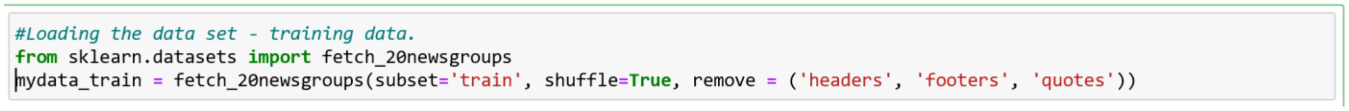
In this dataset, duplicate messages have been removed and the original messages only contain "From" and "Subject" headers (18828 messages total).

Each new message in the bundled file begins with these four headers:

Newsgroup, Document_id, From, Subject:

Each newsgroup file in the bundle represents a single newsgroup

Each message in a file is the text of some newsgroup document that was posted to that newsgroup.



```
#Loading the data set - training data.
from sklearn.datasets import fetch_20newsgroups
mydata_train = fetch_20newsgroups(subset='train', shuffle=True, remove = ('headers', 'footers', 'quotes'))
```

Pre-Processing of Dataset:

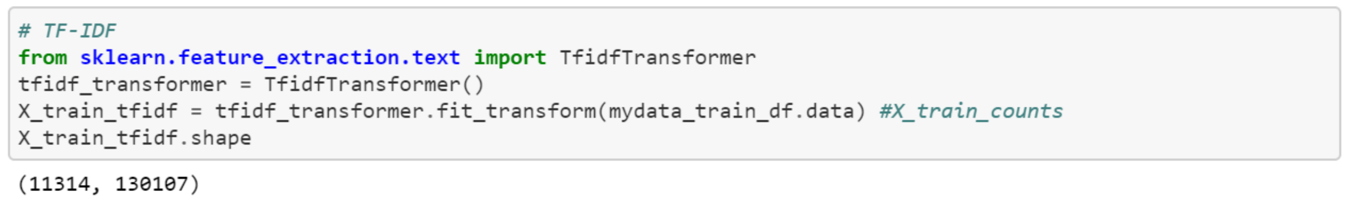
The following general pre-processing steps were carried out since any document being input to a model would be required to be in a certain format:

1. Converting to lowercase
2. Removal of stop words
3. Removing alphanumeric characters
4. Removal of punctuations

5.Vectorization:TF-IDF Vectorizer was used. The model accuracy was compared with those that used Count Vectorizer. In all cases, when TF-IDF Vectorizer was used, it gave better results and hence was chosen as the default Vectorizer.

TF-IDF: TF-IDF is the inverse document frequency which tells how important a word is in a document. It is successfully used for stop-words filtering in various subject fields including text summarization and classification. We have used TF-IDF weighting where words that are unique to a particular document would have higher weights compared to words that are used commonly across documents.

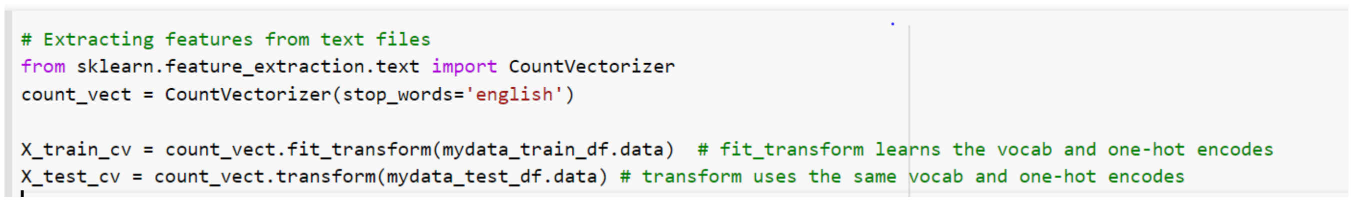
After extracting "bag of words" from the text and apply TF-IDF (term frequency - inverse document frequency) weights. Experimentally we found that applying sublinear TF scaling (1 + log(tf)) improves overall results. We will first extract only unigrams from the text data to form the vectorizer. We repeat the same process for bigrams as well.



```
# TF-IDF
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(mydata_train_df.data) #X_train_counts
X_train_tfidf.shape
```

```
(11314, 130107)
```

Count Vectorizer: Count Vectorizer provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary.



```
# Extracting features from text files
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer(stop_words='english')

X_train_cv = count_vect.fit_transform(mydata_train_df.data)  # fit_transform learns the vocab and one-hot encodes
X_test_cv = count_vect.transform(mydata_test_df.data) # transform uses the same vocab and one-hot encodes
```

Stemming: Stemming is the process of reducing a word to its word stem that affixes to suffixes and prefixes or to the roots of words known as a lemma. Stemming is important in natural language understanding (NLU) and natural language processing (NLP).

Lemmatization: Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma.

Unigrams and Bigrams: N-grams of texts are extensively used in text mining and natural language processing tasks. An n-gram is a contiguous sequence of n items from a given sample of text or speech. an n-gram of size 1 is referred to as a "unigram"; size 2 is a "bigram".

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 models won't be biased towards any particular class and subsequently no resampling was required

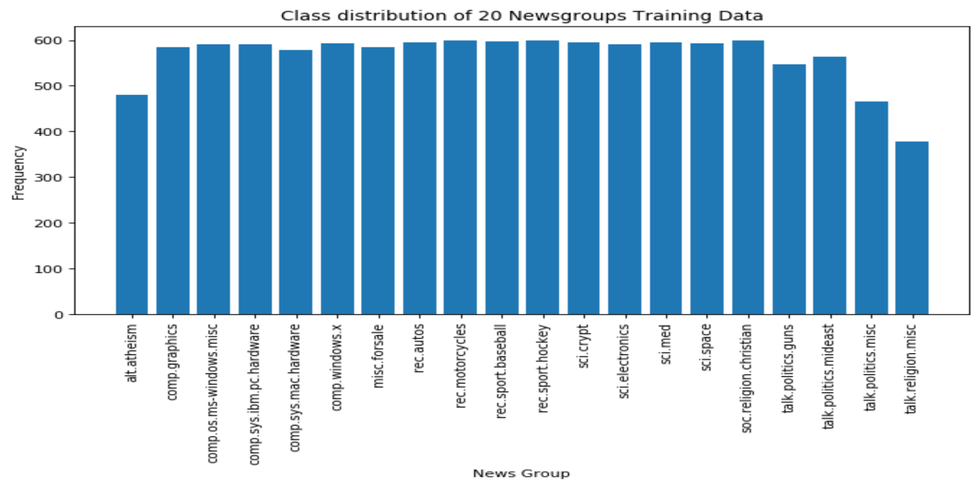


Figure 1. Class Distribution of 20 Newsgroups in Training Dataset

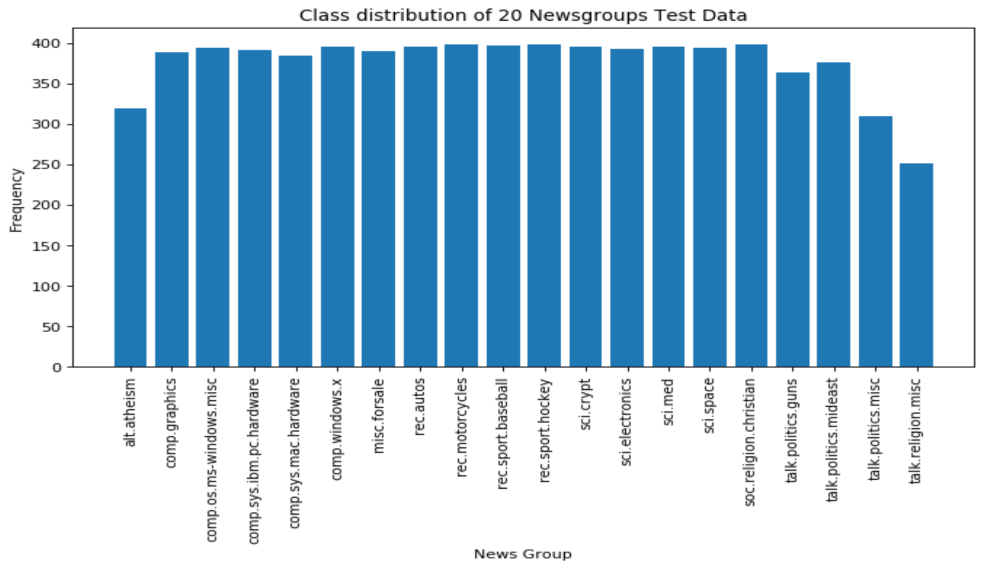


Figure 2. Class Distribution of 20 Newsgroups in Testing Dataset

METHODOLOGY:

In this study, we compared 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. The analysis is performed using sklearn library in Python.

A. 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 extracted only unigrams from the text data to form the vectorizer. We then tested with four classifiers:
- For Logistic Regression 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.
- Next, we extracted unigrams as well as bigrams from the text data to form the TF-IDF Vectorizer. Again, we tested the four classifiers with similar grid search approach for Logistic Regression. The test data performance results obtained are shown in Table I.

N-GRAMS	**Classifier Accuracy (%)**		
	Multinomial NB	**Logistic Regression**	**SGD Classifier**
Unigram	66.93	68.96	69.97
Unigram + Bigram	65.57	68.34	70.39

Table I. Performance of classifiers on test data w/o stemming and lemmatization

B. Applying only Stemming in the TF-IDF Vectorizer:

- We have used stemming technique, i.e., cutting the words to their root form. We used Snowball English Stemmer algorithm from the NLTK package in Python and we defined a class to represent the Snowball

Stemmer. We again test the there classifiers, first with only unigrams and then with both unigrams and bigrams in the vectorizer. Performing grid search on Logistic Regression gave best performance when 'L2' norm was used in the penalization. The test data performance results obtained are shown in Table II.

N-GRAMS	Classifier Accuracy (%)		
	Multinomial NB	Logistic Regression	SGD Classifier
Unigram	66.49	69	70.18
Unigram + Bigram	65.59	68.34	70.66

Table II. Performance of classifiers on test data with stemming

C. Applying only Lemmatization in the TF-IDF Vectorizer:

- Finally, we applied 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 defined a class to represent Lemma Tokenizer. We again tested with 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. The test data performance results obtained are shown in Table III.

N-GRAMS	Classifier Accuracy (%)		
	Multinomial NB	Logistic Regression	SGD Classifier
Unigram	66.49	69	70.18
Unigram + Bigram	65.59	68.34	70.66

Table III. Performance of classifiers on test data with Lemmatization

Classifiers Multinomial Naïve Bayes performed best when neither stemming nor lemmatization was done and only unigrams were extracted in the TF-IDF Vectorizer.

Logistic Regression had the best performance when Lemmatization was used with only unigrams extracted in the TF-IDF Vectorizer. Stochastic Gradient Descent Classifier had the best performance when Stemming was used with both unigrams and bigrams extracted in the TF-IDF Vectorizer. Overall, this was the best performing model achieving an accuracy of 70.66%.

RESULTS AND ANALYSIS:

We compared the results that we achieved in the previous sub sections. In Figures 3 to 5 we visualize the performance of each classifier.

- From Figures 3 and 5 we see that classifiers Multinomial Naïve Bayes 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.
- 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.

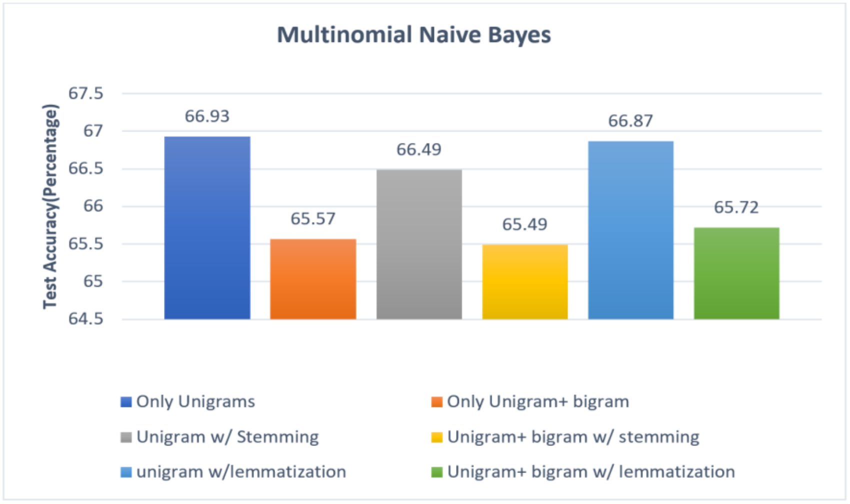


Figure 3. Multinomial Naïve Bayes Performance

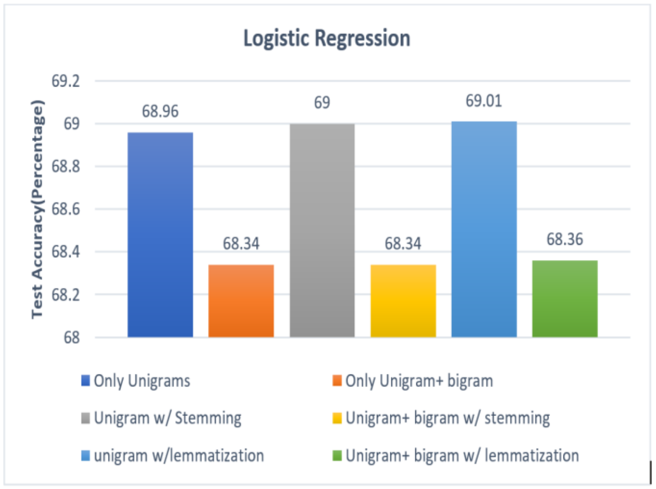


Figure 4. Logistic Regression Performance

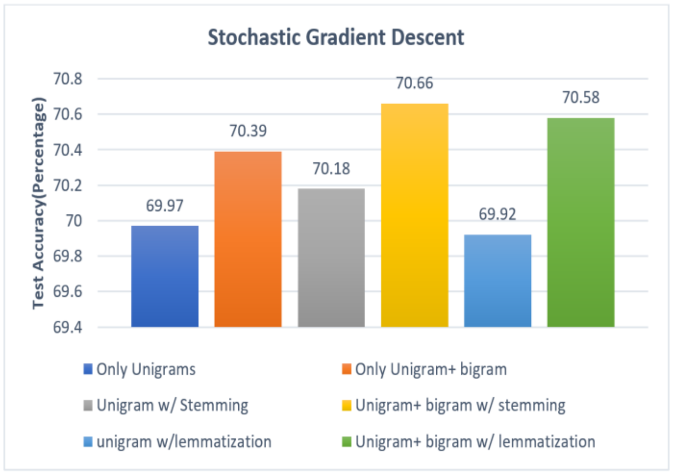


Figure 5. Stochastic Gradient Descent Performance

We see that the highest accuracy of 70.66% is achieved by Stochastic Gradient Descent Classifier, followed by Logistic Regression and Multinomial Naïve Bayes.

Classifier	Model	Test Accuracy(%)
Multinomial NB	No Stemming and Lemmatization + Unigrams	66.93
Logistic Regression	With Lemmatization + Unigrams	69.01
Stochastic Gradient Descent	With Stemming + Unigrams and Bigrams	70.66

Table IV Summarizing the best performance from each classifier

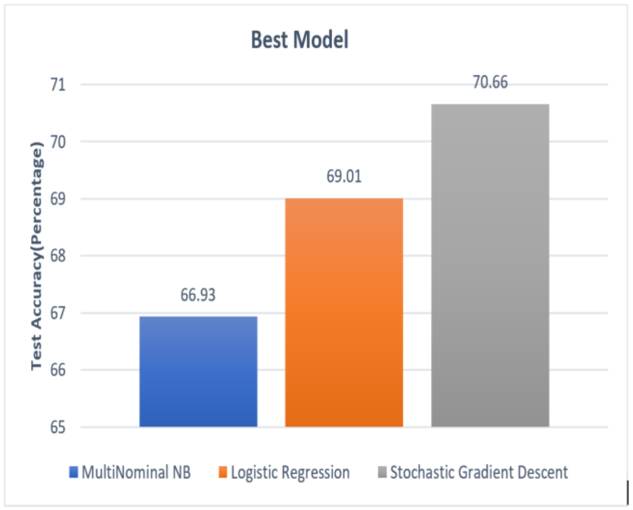
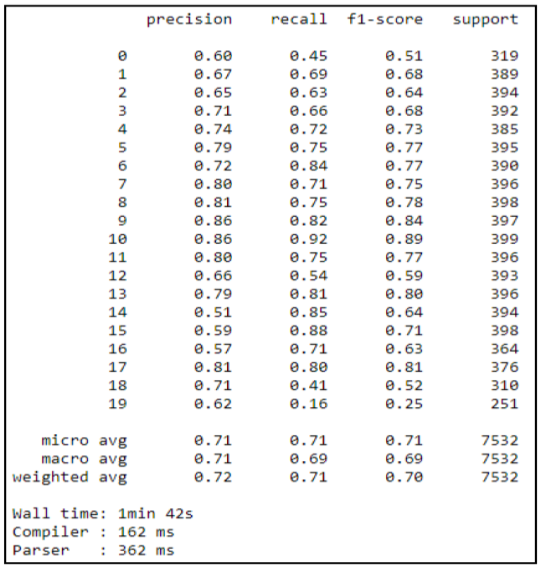


Figure 6: Comparing the best performance of each classifier



```
              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 7: Performance metrics for the best SGD classifier

In Figure 8 we displayed the performance metrics such as precision, recall and f1-score for each of the 20 classes for the best performing Stochastic Gradient Descent Classifier.

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 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) and finally using the best saved weights our model was able to achieve an accuracy of 69.30%.

Neural Networks with Lemmatization+ Unigrams Test Accuracy Percentage is 69.30(2nd best Performance).

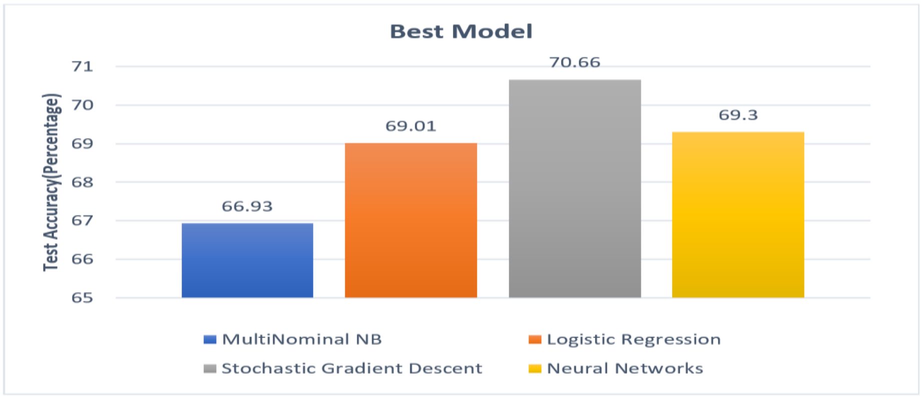


Figure 8. Comparing the performance all the models

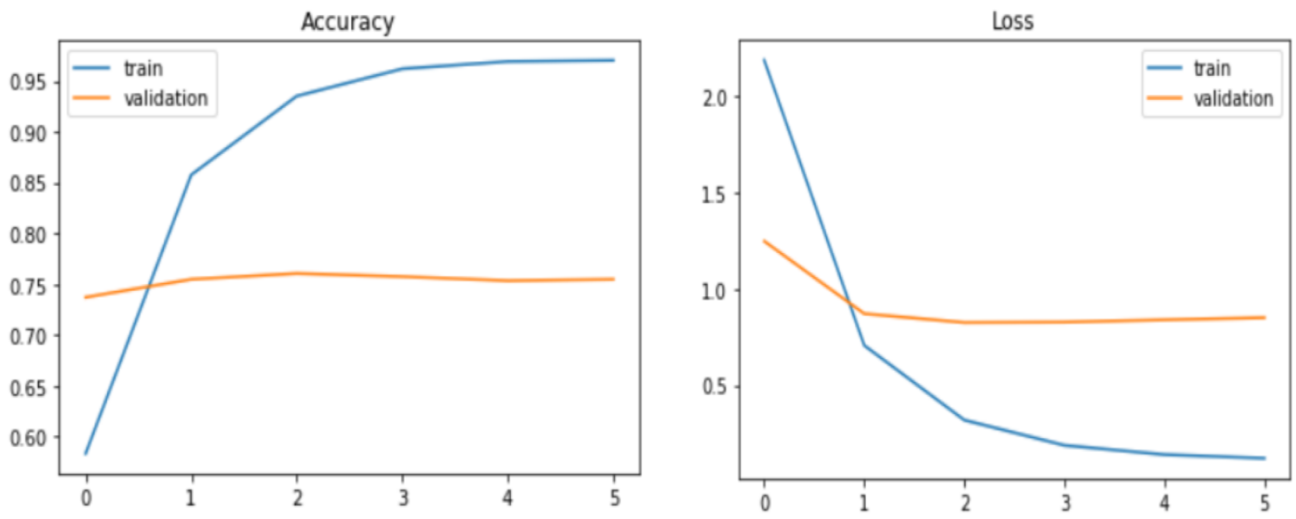


Figure 9. Training/validation accuracy and loss across epochs

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 and training different models and optimizing their parameters using grid search.

REFERENCES:

[1] Dataset: https://www.kaggle.com/crawford/20-newsgroups

[2] R. Sagayam, A survey of text mining: Retrieval, extraction and indexing techniques, International Journal of Computational Engineering Research, vol. 2, no. 5, 2012.

[2] S. M. Weiss, N. Indurkhya, T. Zhang, and F. Damerau, Text mining: predictive methods for analyzing unstructured information. Springer Science and Business Media, 2010.

[3] K. Sumathy and M. Chidambaram, “Text mining: Concepts, applications, tools and issues-an overview,” International Journal of Computer Applications, vol. 80, no. 4, 2013.

[4] C.Ramasubramanian , R.Ramya, Effective Pre-Processing Activities in Text Mining using Improved Porter’s Stemming Algorithm, International Journal of Advanced Research in Computer and Communication Engineering Vol. 2, Issue 12, December 2013.

[5] Vairaprakash Gurusamy, SubbuKannan, Preprocessing Techniques for Text Mining, Conference paper- October 2014

[6] V. Balakrishnan and E. Lloyd-Yemoh, "Stemming and lemmatization: a comparison of retrieval performances", Lect. Notes Softw. Eng., vol. 2, no. 3, pp. 262, 2014.

[7] S. M. Xi, "Application of Natural Language Processing for Information Retrieval", Applied Mechanics and Materials, vol. 380, pp. 2614-2618, 2013.

[8] E. T. Al-Shammari, "Lemmatizing stemming and query expansion method and system", Google Patents, 2013.