Capstone Project

Husnuye Yasar

Machine Learning Engineer Nanodegree

February 22th, 2019

A comparative study of Sentiment Analysis using various ML algorithms

Project Overview

With the democratization of the Internet every individual now has an opportunity to express his or her opinion on any topic, event, company, indeed anything at all. They have an opportunity to express themselves through pictures, words, tweets, posts and numerous other ways.
The challenge then lies in assessing the emotions, feelings, passion, sentiments being conveyed by these individuals or the population as a whole. Given the large volumes of data, the only possible way to really assess these sentiments is computationally. Sentiment analysis is this process of examining the data and drawing conclusions of what the emotions are of the population on any given topic.
This project is going to analyze the sentiments of movie reviews from a Rotten Tomatoes dataset using various Machine Learning and Natural Language Processing algorithms.

Problem Statement

The goal of this project is to implement and train various ML and NLP algorithms to predict the sentiments on a Rotten Tomatoes movie reviews dataset provided by Kaggle [1] and compare them against a preset metrics to figure out which provides the best accuracy. The First step will be to find a benchmark model and set a benchmark metrics. Then for each algorithm the data will have to be pre-processed, split into training and validation sets, the classifier would then be trained on the training set and the model will then be tested on the validation set. This effort aspires to defining a model that is most optimal for datasets and problem statements that this project tackles. The hope is that future users will

have to contend with fewer choices and possibly improve upon the work done here.

Metrics

Since this is a classification problem, and the main goal is to get the maximum number of correct labels, **accuracy** is being used as the metric. Accuracy is defined as follows –

		Predicted class: P	Predicted class: N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

For Neural Networks, there are two entities that can be studied, the loss and the metric. The loss is the function that is used to optimize the model, and the metric is the function that is used to evaluate the quality of the model. In this study, for a multiclass classification problem, categorical cross-entropy is being used as the loss function and Adam as the optimizer.
For Supervised Algorithms, F1-score is also been generated and displayed. It could not be used for Neural Networks, because it was difficult to get F1-scores for these models.
This is a Sentiment Analysis problem with a multiclass classification and the aim is to get the maximum number of correct labels for each review. The main goal of this study is to compare different algorithms to find the best one for this purpose and so accuracy is the best metrics for the given use case.

Analysis

Data Exploration

The dataset for this project is a Rotten Tomatoes movie review dataset provided by Kaggle here [1]. It is a corpus of movie reviews used for sentiment analysis, and is in a tab separated file 'train.tsv'. 3

A sample data snapshot looks like –

	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	A series of escapades demonstrating the adage ...	1
1	2	1	A series of escapades demonstrating the adage ...	2
2	3	1	A series	2
3	4	1	A	2
4	5	1	series	2

There are four columns in the data –

1. PhraseId – The id of each Phrase. This acts like the table index.
2. SentenceId – The id of each Sentence. Each Sentence has been parsed into many phrases. Phrases that are repeated (such as short/common words) are only included once in the data.
3. Phrase – Collection of words that expresses the Sentiment. They all come from the Rotten Tomatoes dataset.
4. Sentiment – The actual sentiment value, the label or target variable.

The phrases are labelled on a scale of five values: negative, somewhat negative, neutral, somewhat positive and positive.

Each phrase has a sentiment label associated with it as follows:
0 - negative
1 - somewhat negative
2 - neutral
3 - somewhat positive
4 – positive

Total number of Phrases – 156060
Total number of Sentences – 8529 4

Phrases Count by label:

Phrase	Count
Negative	7072
Somewhat Negative	27273
Neutral	79582
Somewhat Positive	32927
Positive	9206

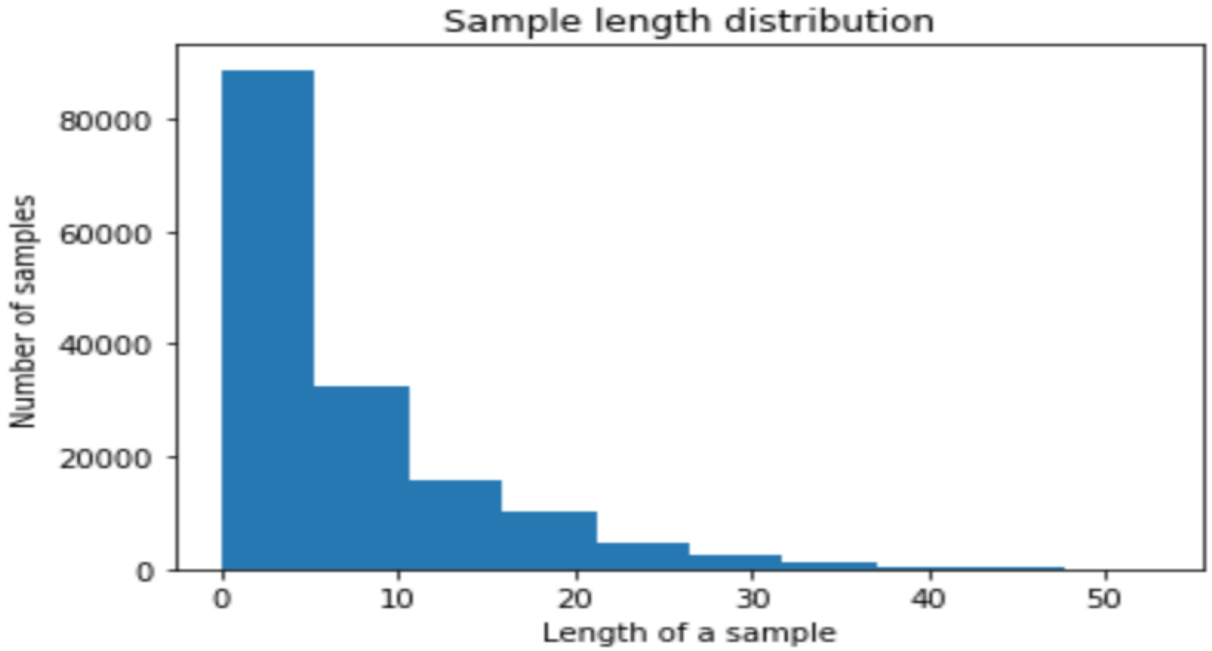
As seen from the table above, more phrases are neutral, followed by somewhat positives and somewhat negatives, and the least number are strictly positive or negative.

A simple analysis of the Phrases yields the following results –

Maximum number of words in a Phrase – 53
Minimum number of words in a Phrase – 0
Mean word length of a Phrase – 7.208
Median word length of a Phrase – 5

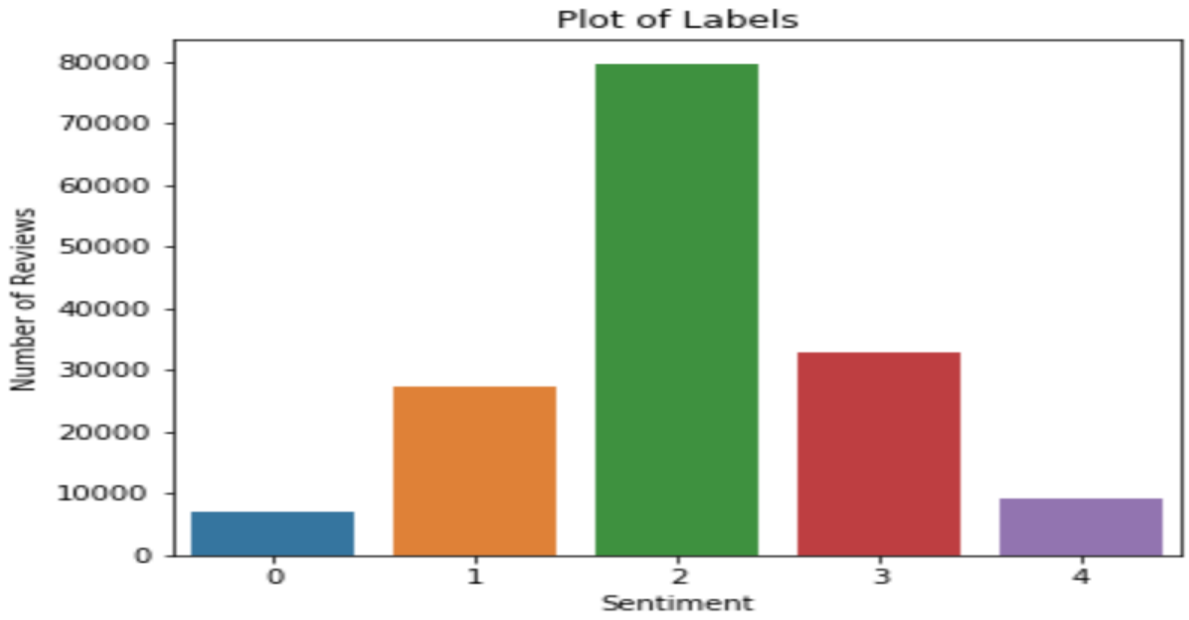
Exploratory Visualization

A plot of the number of words in phrases is shown below –



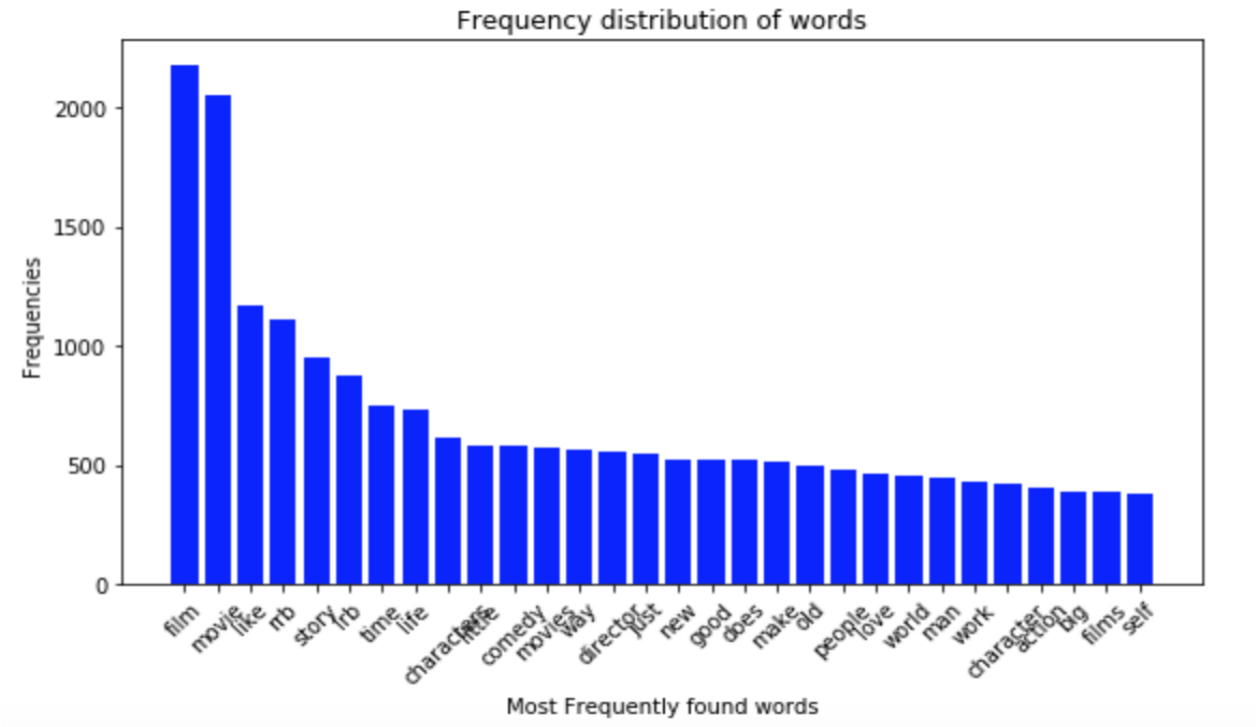
As can be seen from the plot above, most of the phrases are less than 10 words long which corroborates the fact that median phrase length is 5 words and mean is 7.2 words.

A plot of the distribution of Phrases and Sentiments is shown below –

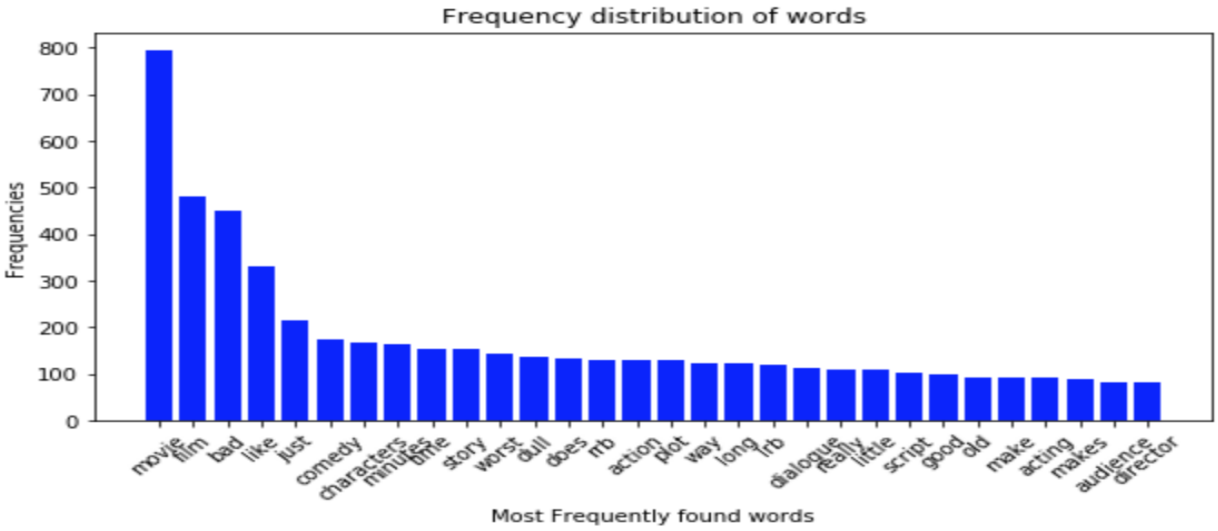


As seen once again from this plot there are many more phrases with neutral sentiments than strictly positive or negative sentiments.

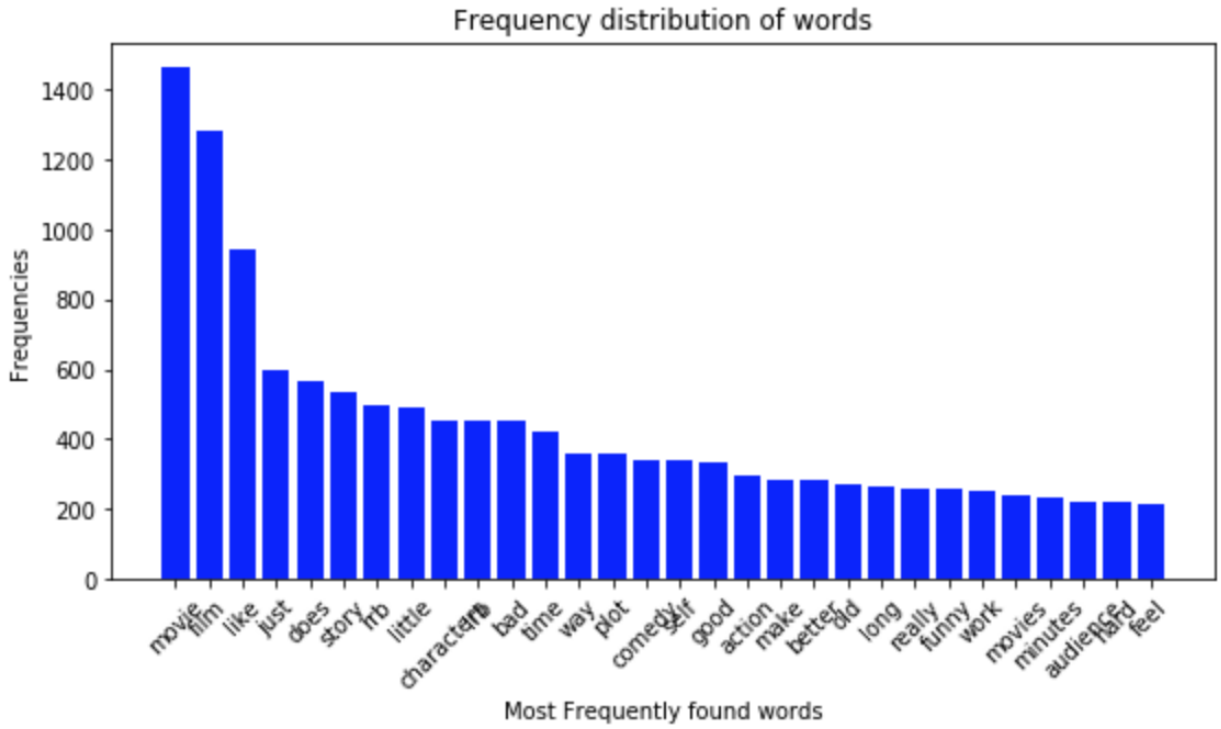
To look at some common words used in all the phrases let's look at the following plots with common stop words removed –
All Phrases –



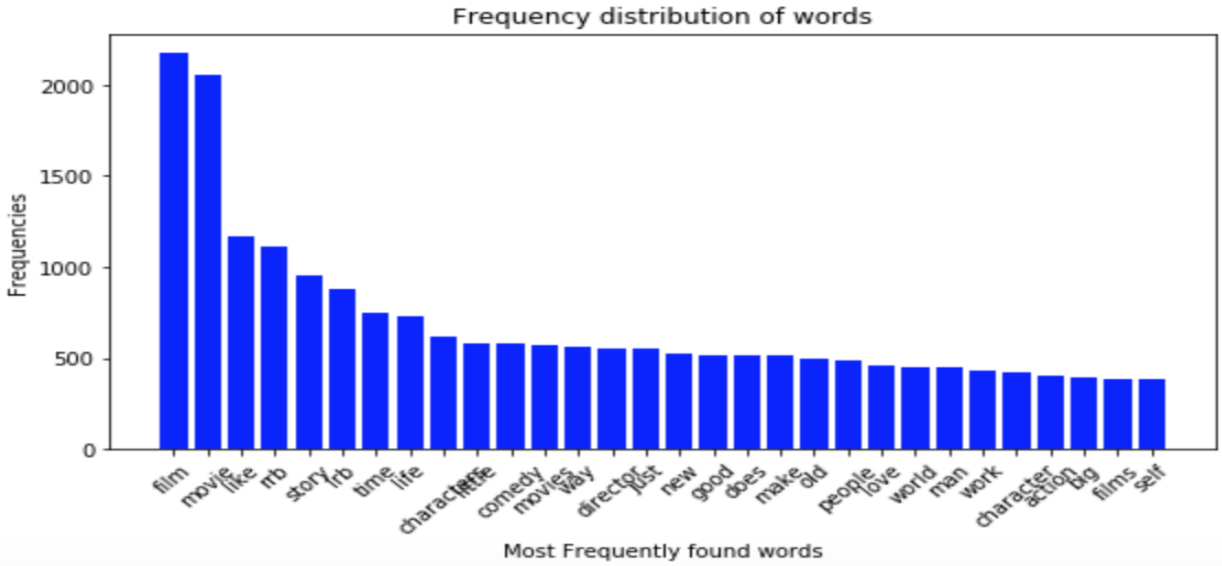
Negative Sentiment Phrases –



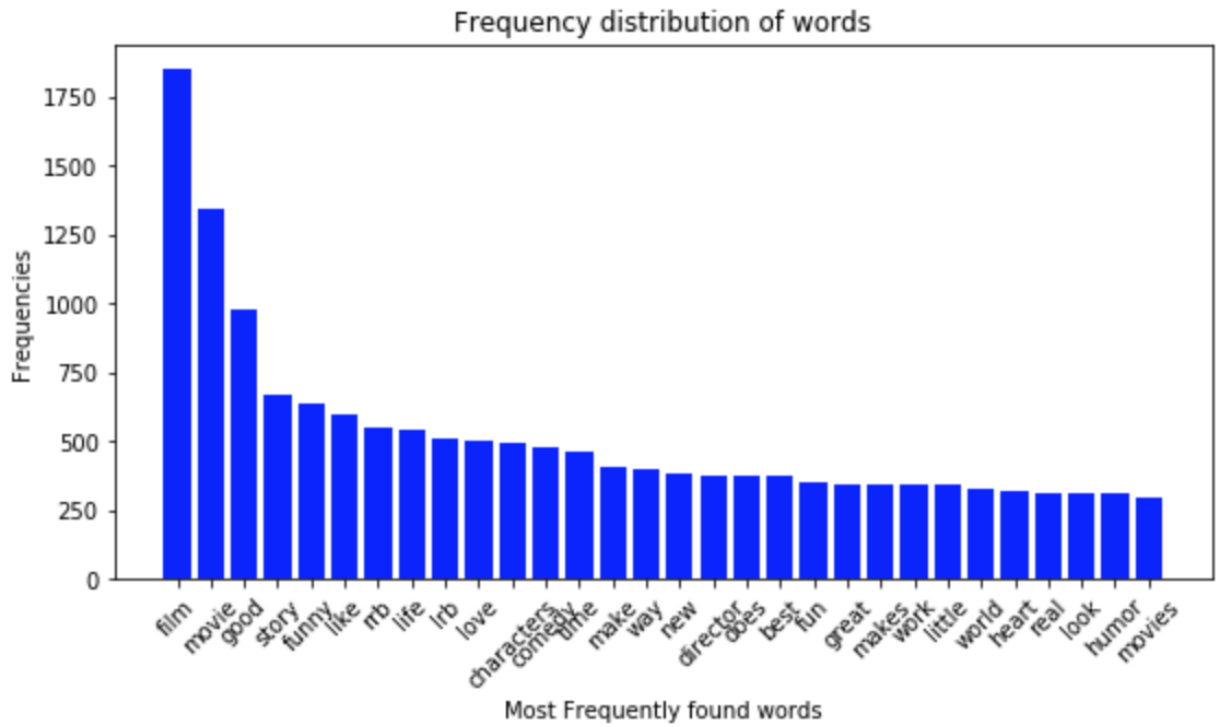
Mostly Negative Sentiment Phrases –



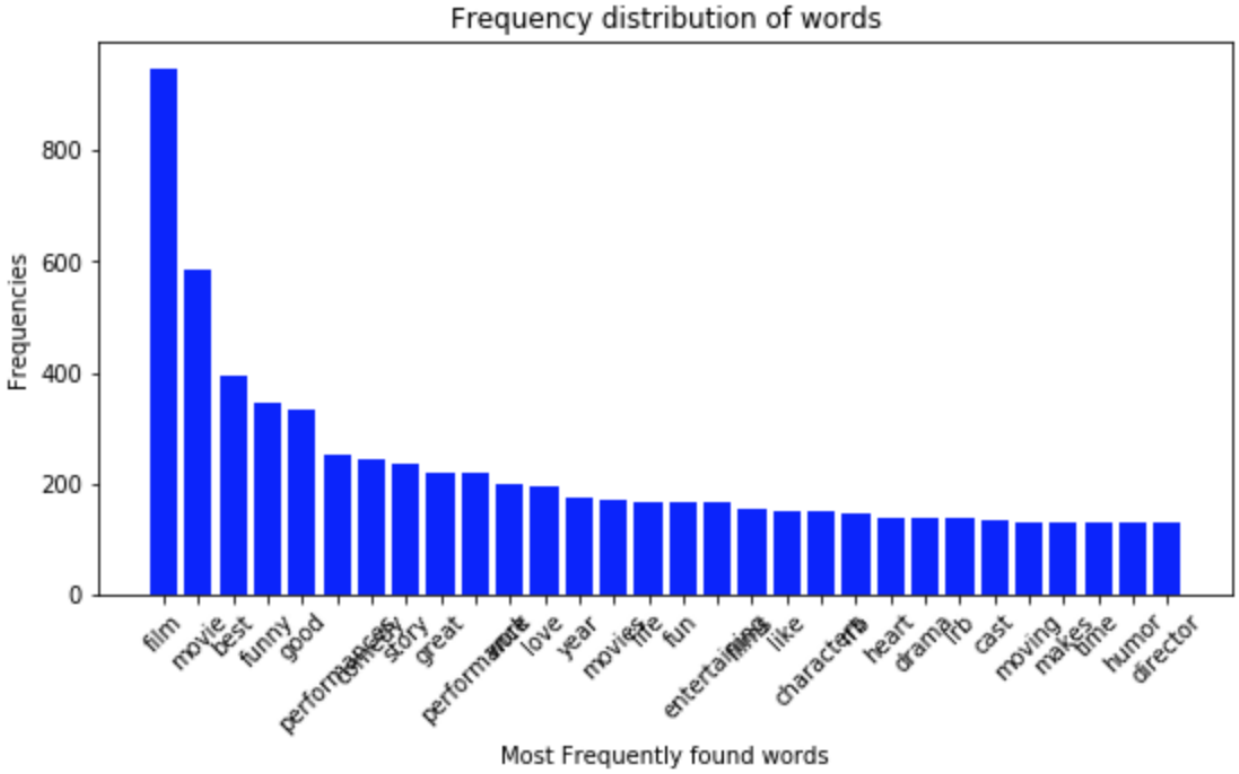
Neutral Sentiment Phrases –



Mostly Positive Sentiment Phrases –



Positive Sentiment Phrases –



Some Observations based on these graphs:

- Negative reviews include words like bad, worst, dull.
- Positive reviews include words like best, funny, good, great, love.
- Neutral reviews have words like story, time, life, characters.

Algorithms and Techniques

For this study the following algorithms have been analyzed –

1. **Logistic Regression**

Logistic regression is a Supervised Learning algorithm and despite its name is a linear model for classification. It uses statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. Logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function. It is one of the most widely used algorithms for twitter sentiment analysis.

- Strengths - widely used classification algorithm, fast training and prediction time, works well with fewer features
- Weakness - does not work well if the data is noisy

2. **Support Vector Machines (SVM)** – This is another popular Supervised Learning Algorithm for multiclass classification. These algorithms, in addition to performing linear classification, can efficiently perform non-linear classification using the 'kernel trick' which implicitly maps the inputs into high-dimensional feature spaces by constructing sets of hyperplanes.
   - Strengths - works really well when there is a clear margin of separation, works really well in high dimensional spaces and uses a subset of training points in the decision function, so is memory efficient.
   - Weakness - Training time is higher, does not perform well when there is more noise in the data or target classes are overlapping.

3. **Random Forest Classifier** – This is an ensemble learning method which combines the results of several simple decision trees to predict the outcome. Each of the decision trees are created using a random set of features and data set. It is a good fit for a large dataset with multiple categorical variables
   - Strengths - Suitable for large dataset, reduces variance without increasing bias, takes care of feature importance.
   - Weakness - Humans have very little control over what the model does. it can be too slow for prediction

4. **Neural Networks –** Next model chosen is a simple Neural Network with a couple of Dense layers and a couple of Dropout layers to prevent overfitting. This model is expected to show much better text accuracy because it can leverage information about text ordering.
5. **Convolutional Neural Networks** –These are a variant of Neural Network which are mostly used for Image Recognition and Computer Vision but they have been used very successfully in NLP applications as well. CNNs are a class of deep, feed forward neural networks which use a variation of multilayer perceptron designed to require minimal preprocessing. A CNN consists of an Input and an Output layer plus multiple hidden layers. These hidden layers could be convolutional layers, pooling layers, fully connected layers or normalization layers
6. **Recurrent Neural Networks with LSTMs** – Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network that has shown best results with sequence data. It has memory and remembers data from previous inputs. It then makes decisions based on that knowledge and the current input. These networks work better for NLP since each word in a sentence has meaning based on the context or surrounding words and are more capable of capturing the changing sentiments.

Benchmark Model

The Benchmark model chosen for the study is a basic Multinomial Naïve Bayes classifier which uses a simple 'Bag of Words' model [2] using CountVectorizer [3] from sklearn package. Naïve Bayes is a predictive model that is simple to implement and easy to understand. It is the typical 'first-go' model seen when working with NLP datasets and hence it was chosen as the benchmark model.
Naïve Bayes Results –
Data was split into train/test using 75/25 ratio. So, there were 117045 training instances and 39015 testing instances.
CountVectorizer vocabulary – 500 words
Time to train – 0.52 seconds
Accuracy – **49.058**%
F1-score – 0.384

Methodology

Data Preprocessing

For Supervised Learning Algorithms –

The data is first split into training and testing set using a 75/25 ratio and the stratify option is used. This is especially important because this is an unbalanced multiclass classification problem and the balance is maintained in the training and testing set.

Next step is to convert the data into feature vectors for the models. Two methods were used for this –

1. **CountVectorizer**
   Converts a collection of text documents to a matrix of token counts. In this case each row is a phrase and each column a token. Each entry in the matrix shows the frequency of that token in the review. It produces a sparse representation of the counts. For this implementation, the vocabulary size used is 500 and stop words are removed.

2. **TfIdfVectorizer**
   Another model used is Term Frequency, Inverse Document Frequency, abbreviated tf-idf. Whereas CountVectorizer gives a sparse matrix as output, TfIdfVectorizer, which is nothing but a CountVectorizer followed by a TfIdfTransformer gives a normalized output. It gives more weightage to lesser used features. Stop words are removed for this implementation.

For Neural Networks data is preprocessed a little differently. Instead of using the feature vector models described above, the data is programmatically cleaned. The stop words like a, an, the, if, etc. have to be removed. For this the stop words available in nltk package of python are downloaded and used. Next all reviews with zero length are dropped from the data.

Original number of data records – 156060

After dropping new number – 155250

3. **Same Length Input Sequence** –
   For Neural Networks all inputs should be sequences of the same length. To achieve this the following steps are performed using the keras library –
   a. Generate word tokens from review phrases using the keras tokenizer
   b. Use keras to generate sequences from words
   c. Convert all sequences to be of the same length

   d. Use keras to hot encode all the sequences

e. Use keras to hot encode all the labels so that 'softmax' activation function can be used in the output layer

4. **Word2Vec** –
   Word2Vec is a model used to produce Word Embeddings, a set of NLP techniques where words and phrases are mapped to vectors of real numbers. It involves a mathematical embedding using a shallow two-layer neural network that is trained to generate these word embeddings. Word2Vec has been shown to boost the performance of NLP tasks like sentiment analysis. For this study a pretrained word vector is used which contains one million word vectors trained on Wikipedia 2017, UMBC web base corpus and statmt.org news dataset (16B tokens). [7]

Implementation

The study can mainly be divided into two main steps –

1. The data preprocessing and model training of Supervised Learning Algorithms
2. The data preprocessing and model training of Neural Networks

For Supervised Algorithms

a. Data is first loaded and split into training and testing set in 75/25 ratio using stratify option to maintain the label ratio in both the sets.

b. Then two feature sets are created, one using CountVectorizer and other using TfIdfVectorizer. For both the stop words are removed.

c. Each of the four models (including the benchmark model) used for this study are trained using the sklearn library.

d. For each of the four models used in this study:
   The model is first trained on the CountVectorizer feature generated training data set and the training time is reported.
   The accuracy and f1-score are calculated using the CountVectorizer generated testing set.
   The process is then repeated with a new model and TfIdfVectorizer generated data set to get the training time and scores.

For Neural Networks

a. Data is first preprocessed to same length sequences using nltk stop words and keras tokenizer. Labels are also preprocessed to categorical sequences.
b. Word embeddings are created by loading the pre-generated Word2vec Wikipedia embedding.

c. The first architecture shown in this study is a Neural Network implemented using the keras library with Tensorflow backend. This network has one fully connected Dense layer with 230 nodes and one output layer with 5 nodes. It also has two Dropout layers.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dropout_1 (Dropout)          (None, 2000)              0
_________________________________________________________________
dense_1 (Dense)              (None, 230)               460230
_________________________________________________________________
dropout_2 (Dropout)          (None, 230)               0
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 1155
=================================================================
Total params: 461,385
Trainable params: 461,385
Non-trainable params: 0
```

The optimizer used is Adam and loss function is categorical cross entropy. It is trained for 5 epochs and a batch size of 50 (This is the only architecture shown, but this was reached after a few experiments as will be described below in refinements)

d. Then some Convolution Layers were added to the above model, the accuracy dropped a bit and results stayed about the same with efforts to refine and tune the hyperparameters, so they are not being reported here.

e. Next architecture tested was a Recurrent Neural Network with an embedding layer and an LSTM layer. This was also implemented using keras with Tensorflow backend. It used the word2Vec embeddings loaded in the embedding layer. It also used Spatial Dropout and Batch Normalization along with two Dense layers of 128 and 64 nodes respectively before connecting to the output layer of 5 nodes. There were also Dropout layers between each dense layer. The optimizer used is Adam and loss function is categorical cross entropy. It is trained for 5 epochs and a batch size of 50. Again, these numbers were achieved after some experiments as will be outlined below.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 32)                0
_________________________________________________________________
embedding_1 (Embedding)      (None, 32, 300)           4567200
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 32, 300)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 256)               570368
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024
_________________________________________________________________
dense_3 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_5 (Dense)              (None, 5)                 325
=================================================================
Total params: 5,180,069
Trainable params: 612,357
Non-trainable params: 4,567,712
_________________________________________________________________
```

Refinement

General refinements done for all algorithms –

1. Removing stop words from the phrases. These are words which are used a lot in the English language like ‘a’, ‘an’, ‘the’, ‘for’, etc. They do not add any value to the sentiment of the phrase but just act as noise in the data. The nltk stop words were used for this.
2. Remove the reviews with null phrase length because they do not add any value to the model.
3. During splitting the data into training and testing set, use common random_state argument, so that the split is consistent for each algorithm.
4. Also, for train/test split use the stratify argument which ensures that the labels in train and test data are balanced.

For the CountVectorizer feature vector generator, the vocabulary size had to be tuned to get the best accuracy and F1-score for benchmark model.

6. For Random Forest Classifier, did a sklearn GridSearch to tune hyperparameters but the base model with just a random state added gave the best accuracy.
7. For Neural Networks, tried adding different combinations of 'Dense' and 'Dropout' Layers to the model described above but the accuracy dropped. The model described in the implementation section gave the best results.
8. Tried adding Convolutional Neural Networks by adding Conv1D and MaxPool1D layers and trying various layer architectures but accuracy stayed in the same range.
9. The best improvement seen was when LSTM layer was added. So, time was spent finetuning that architecture. Improvements made in the final architecture were –

a. Batch Normalization Layer was added
b. Dropout factor was increase to 0.25 and 0.5

One of the biggest challenges was letting go of the notion that CNNs are not better than Supervised algorithms at NLP tasks. Also tuning Neural Networks, CNNs and RNNs is challenging, figuring out the right layer combination and hyperparameters takes a lot of time. Was not aware that sklearn and GridSearch could be used with Keras.

Results

Model Evaluation and Validation

The Support Vector Machine using architecture was chosen because it performed the best among all the models tested. It is described in detail in the implementation section above. Some other features of the model are –
1. The accuracy of this model is 64.62%
2. F1 Score of this model is F1 Score = 0.64%

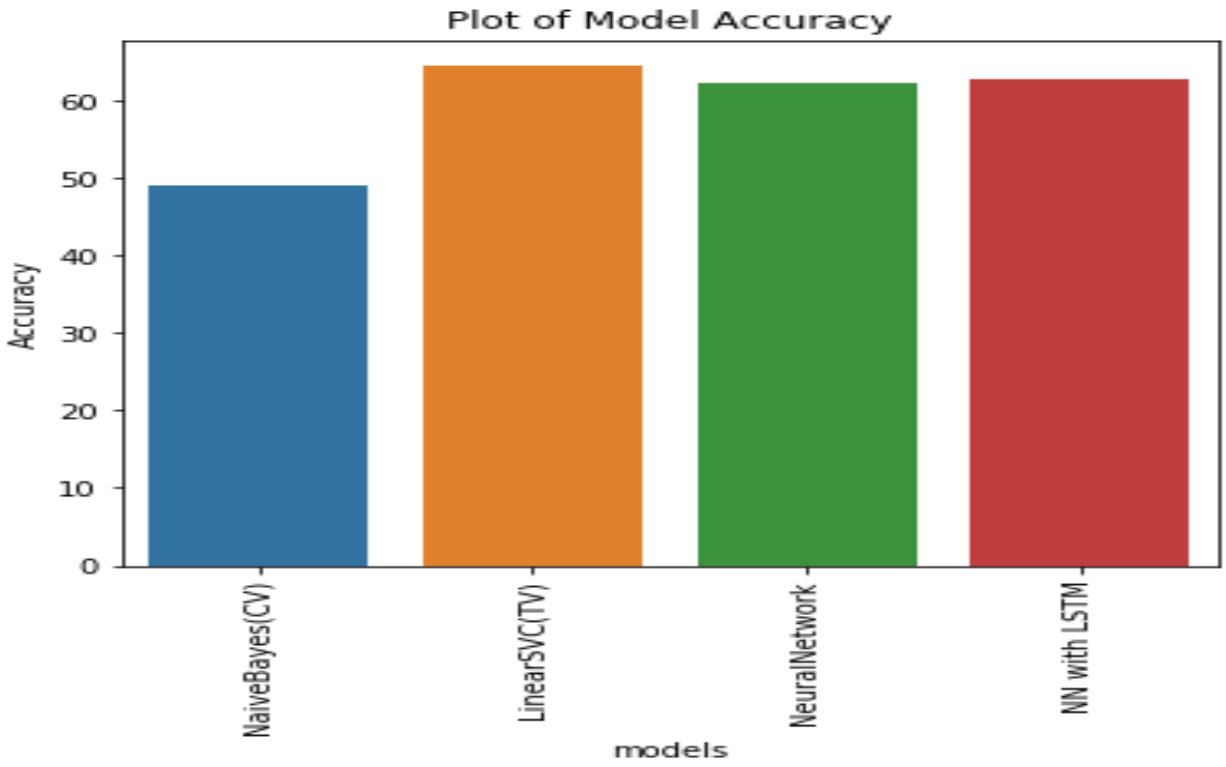
Justification

Accuracies obtained in the study

Model	Accuracy
Benchmark model - Naïve Bayes using Count Vectorizer	49.06%
Best Supervised model – Support Vector Machine using	64.62%

TfIdfVectorizer
Neural Network 62.26%
Final – Recurrent Neural Network with LSTM 62.82%

A plot shows these accuracies as below –

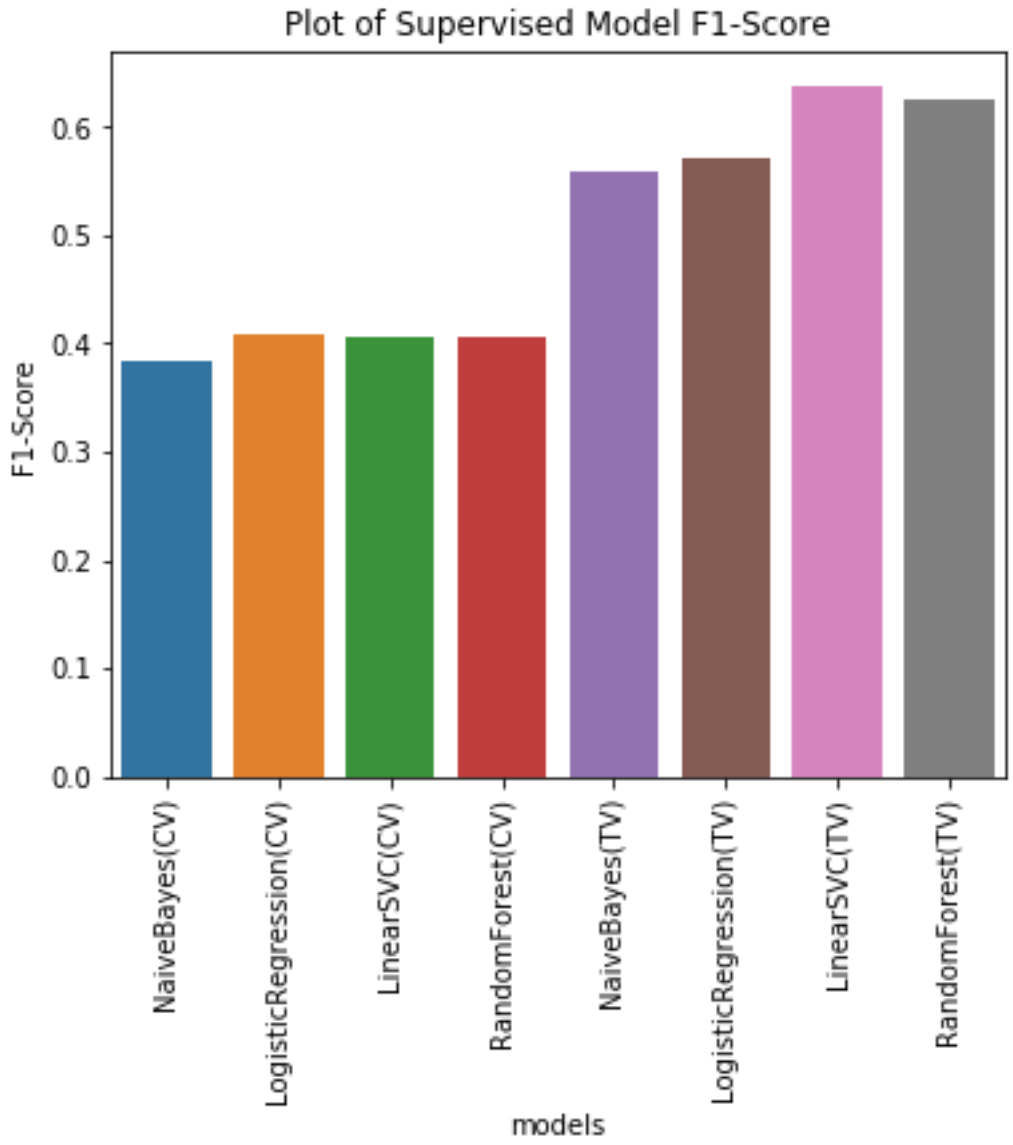


1. An improvement of about 13.76% in accuracy score was achieved using Neural Networks with LSTM over the Benchmark model of NaiveBayes.
2. The best supervised model also has 2.36% better accuracy than Recurrent Neural Network model with LSTM model found in this study.

Based on these observations it can be concluded that The Support Vector Machine using TfIdfVectorizer is the best model for Sentiment Analysis of the given Rotten Tomatoes movies dataset.

Conclusion

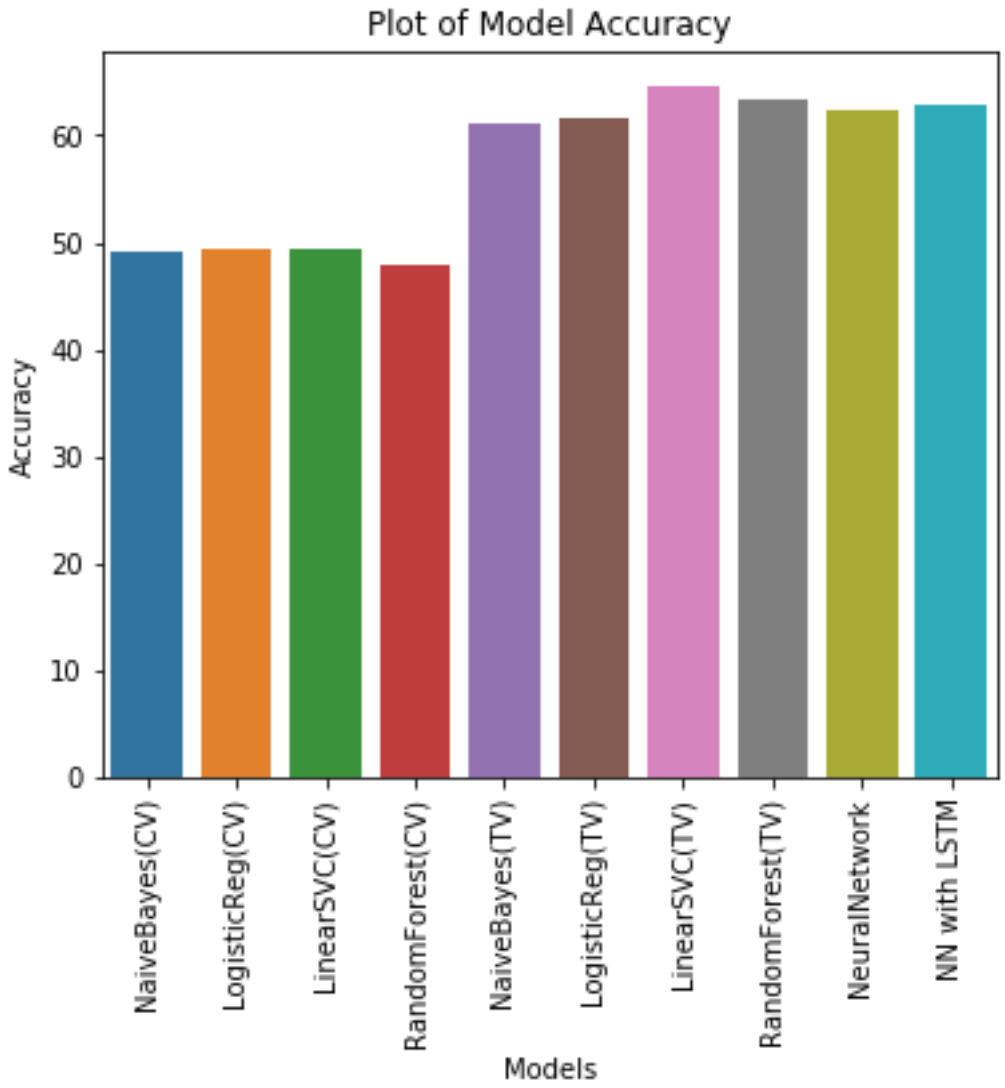
Free Form Visualization



This is a plot of the F1-scores of the Supervised Learning models implemented in this study. Some observations are –

1. The Benchmark model, Naïve Bayes using CountVectorizer has the lowest F1-score of 0.384. The Support Vector Machine using TfIdfVectorizer has the best f1-score of 0.638.
2. It was observed that data preprocessed with TfIdfVectorizer did better than CountVectorizer in all Supervised Learning algorithms and gave a much better F1-score.

A plot of **Accuracy Scores** of all models implemented is shown below **–**



Looking at the above plot –

1. As already discussed benchmark model has an accuracy of 50.89% and the best model, The Support Vector Machine using TfIdfVectorizer has an accuracy of 64.621%
2. Accuracy score of Random Forest with CountVectorizer was slightly less than the benchmark model.

3. Just like F1-score it was observed here also that Data preprocessed with TfIdfVectorizer performed much better in all the Supervised Learning Algorithms as compared to the CountVectorizer. The accuracy was better by about 10%.
4. Neural Networks did not do better. In fact, Support Vector Machines gave a better accuracy.
5. Also tried many different variations of CNN with Conv1D and MaxPool1D but got very comparable accuracies. So, they were not included here in the study.
6. LSTM gave the good accuracy, and this could be improved by more parameter tuning.

Reflection

The process used in this project can be summarized in the following steps –

1. Loaded and Analyzed the dataset
2. Pre-processed the data which involved removing stop words using nltk package, removing null reviews etc.
3. Feature Engineered the data including Count Vectors, Tf-Idf Vectors, Equal length sequences, keras tokenizer.
4. A benchmark model was created and trained for the classifier
5. Word vectors were loaded
6. Three Supervised Learning models were trained
7. Many combinations of Neural Network Models were trained. Only two best ones were included in the study.

This project was not just a learning experience, it was a journey. It began with the exploration of the relatively simpler Naïve Bayes, continued through the world of Supervised Learning algorithms from Logistic Regression to Support Vector Machines to Random Forest. Then it continued into the world of previously unexplored word embeddings like word2vec. Last came the most exciting and frustrating part of training neural networks, convolutional neural networks and recurrent neural networks. A few realizations here were that Neural networks and CNNs are not always better than Supervised algorithms, and they are a lot slower to train. RNNs with LSTM worked quite well even though they took a long time to train.

After working with this data set, trying different algorithms and tuning the hyperparameters, the best accuracy that could be achieved was only around 65%. Whereas in papers and other articles there are similar multi-class text

classification problems using similar techniques and models getting much higher accuracy of around 80%. The main difference observed is that this is an unbalanced dataset, with about 50% of the labels belonging to one category alone (Neutral), followed by 21% almost positive, 17% almost negative. Much fewer labels belong to strictly positive or negative category. This might have had an effect on the overall performance of the algorithms.

Improvement

Some suggestions for improvements –

1. Using more than one LSTM layer could have improved the model accuracy
2. There are other word embeddings that could be used instead of word2vec, like Spacy.
3. There are other metrics that can be used like area under ROC curves.
4. There are other models that might give better performance for NLP models that were not tried in this study like RNNs using GRU's or a combined CNN-LSTM or LSTM-CNN model [8]
5. Do some data preprocessing so that the dataset becomes more balanced, the number of labels in each class are more similar.

Resources

[1] https://www.kaggle.com/c/movie-review-sentiment-analysis-kernels-only/data
[2] https://en.wikipedia.org/wiki/Bag-of-words_model
[3] http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
[4] https://en.wikipedia.org/wiki/Logistic_regression
[5] https://en.wikipedia.org/wiki/Support_vector_machine
[6] https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd
[7] https://fasttext.cc/docs/en/english-vectors.html
[8] http://konukoii.com/blog/2018/02/19/twitter-sentiment-analysis-using-combined-lstm-cnn-models/