BAN 675: Text Mining

PROJECT REPORT
Sentiment Analysis: IMDb Movie Reviews

Submission By: Group 4
Mudit Mishra
Sayali Mahamulkar
Vinod Parla
Rahuldeep Shenoy
Mahavir Parmar
Viral Patel

TABLE OF CONTENTS

I. INTRODUCTION

Executive Summary

Our project's goal is to predict the sentiment of a given movie review by using machine learning and deep learning models which we trained on the IMDB movie review dataset. We took an IMDB dataset which contains 50,000 movie reviews. We pre-processed and analyzed the data, developed, and trained a model that can categorize new reviews as positive or negative. We further extended our analysis to two different length datasets by further dividing our original IMDB dataset into two subsets based on the length of the reviews.

Motivation

As we know, movies are one of the most significant sources of entertainment in the industry. It influences people to some extent, whether it is positive or negative. Therefore, the movie rating is crucial to tell if a movie is good or bad. People tend to look at the review and evaluate before they start watching, so they save their time and money. Besides, parents can decide if their kids can watch the movie if it is not R-rated. It is very time-consuming as it will take at least two to three months to get an accurate rating based on how many people rated the movie. After some brainstorming sessions, we suggested some solutions to the problems. We can predict whether it is a good Review or a bad Review depending upon the historic Data set and give an accurate solution.

II. DATA DESCRIPTION

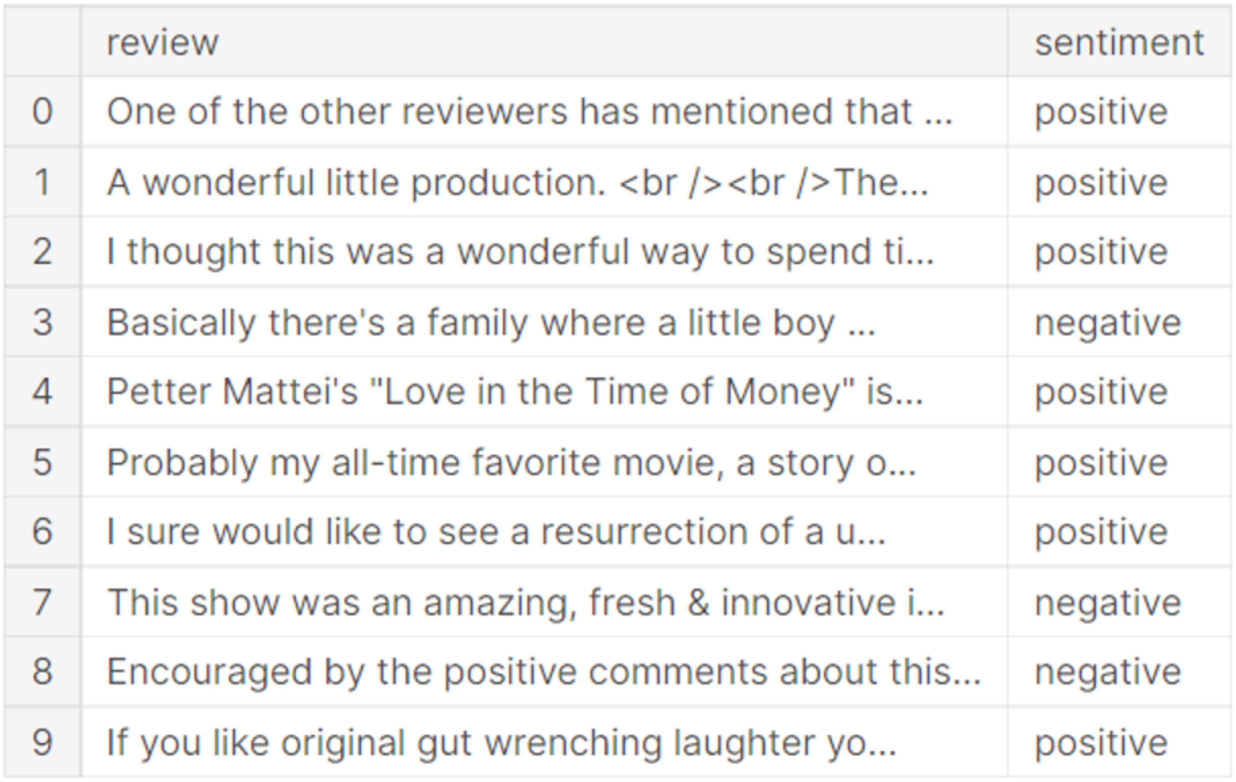
Data Source

We have collected this dataset from the Large Movie Review Dataset which is provided by ai.stanford.edu. This is real-time data provided by the site. It represents reviews by customers about movies which can either be positive reviews or negative ones.

About the Data

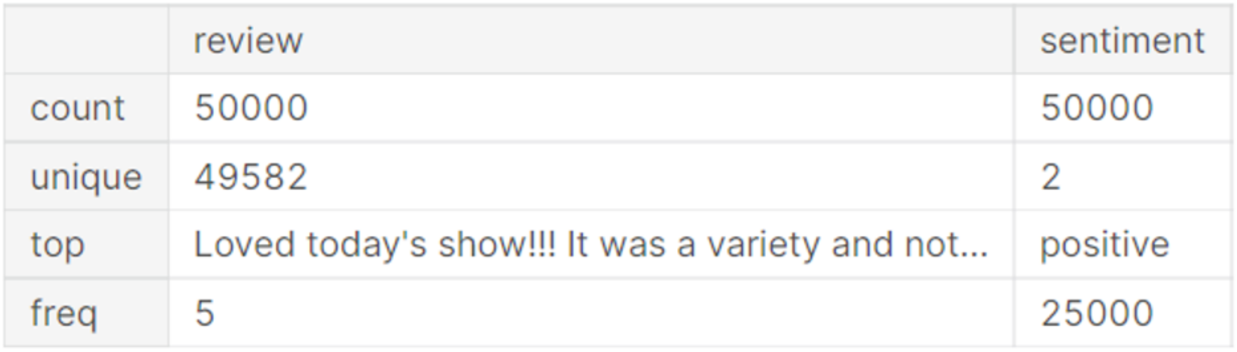
This Dataset contains about 50,000 rows that has 2 Columns named ‘Review’ And ‘Sentiment’. Based on the review type/sentiment, they are classified as positive or negative. This dataset can be called a balanced dataset. The Dataset didn’t have any null values which is a good thing. But the data does need a lot of cleaning since there is a lot of randomness in it. The Individual Reviews contain Html Tags, stop words (A, An, From, To, Etc), words with the same root word but used differently (Played, Playing, Play), special characters, etc. Thus, all these factors have no effect on deciding the sentiment which makes the dataset complex which in turn would affect overall model performance. Thus, this dataset needed a lot of cleaning before we used it for our analysis.

The Data is in the following format:



	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. <br /><br />The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
5	Probably my all-time favorite movie, a story o...	positive
6	I sure would like to see a resurrection of a u...	positive
7	This show was an amazing, fresh & innovative i...	negative
8	Encouraged by the positive comments about this...	negative
9	If you like original gut wrenching laughter yo...	positive

Fig 1: Movie Reviews in dataset



	review	sentiment
count	50000	50000
unique	49582	2
top	Loved today's show!!! It was a variety and not...	positive
freq	5	25000

Fig 2: Details of Dataset

Positive Words Word Cloud

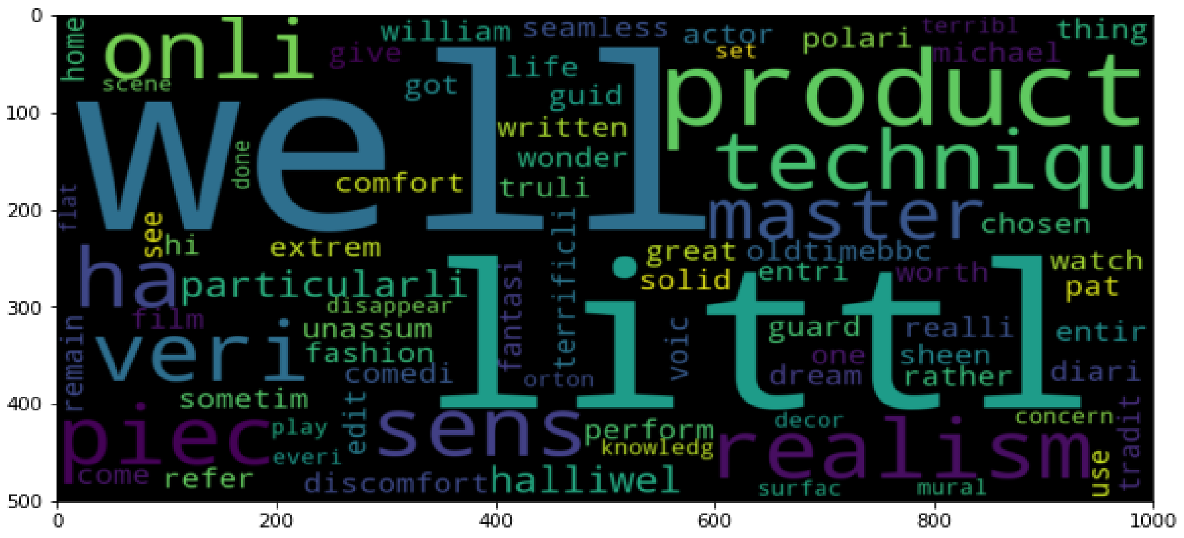


Fig 3: Positive Word Cloud

The maximum number of words used to generate a positive word cloud are 500. These words in this cloud are historic data and are all positive words that were used by reviewers previously.

Negative Words Word cloud

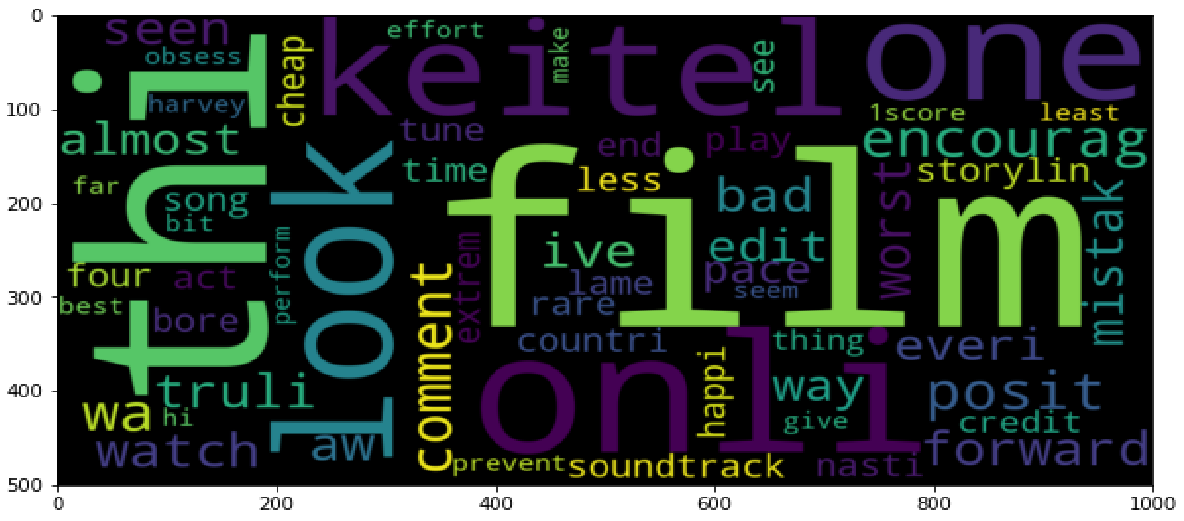


Fig 4: Negative Word Cloud

The maximum number of words used to generate this word cloud are 500. These words give us an understanding of negative words used in the reviews. This data is yet to be cleaned and it looks unstable with unwanted information and after the data is cleaned and pre-processed data would be more definite.

III. DATA CLEANING AND PREPROCESSING

Data Pre-processing in our project for the IMDb dataset involved removal of certain Stop words, html strips and noise text, unwanted symbols (&, $, #, etc.).

When to Remove stop words

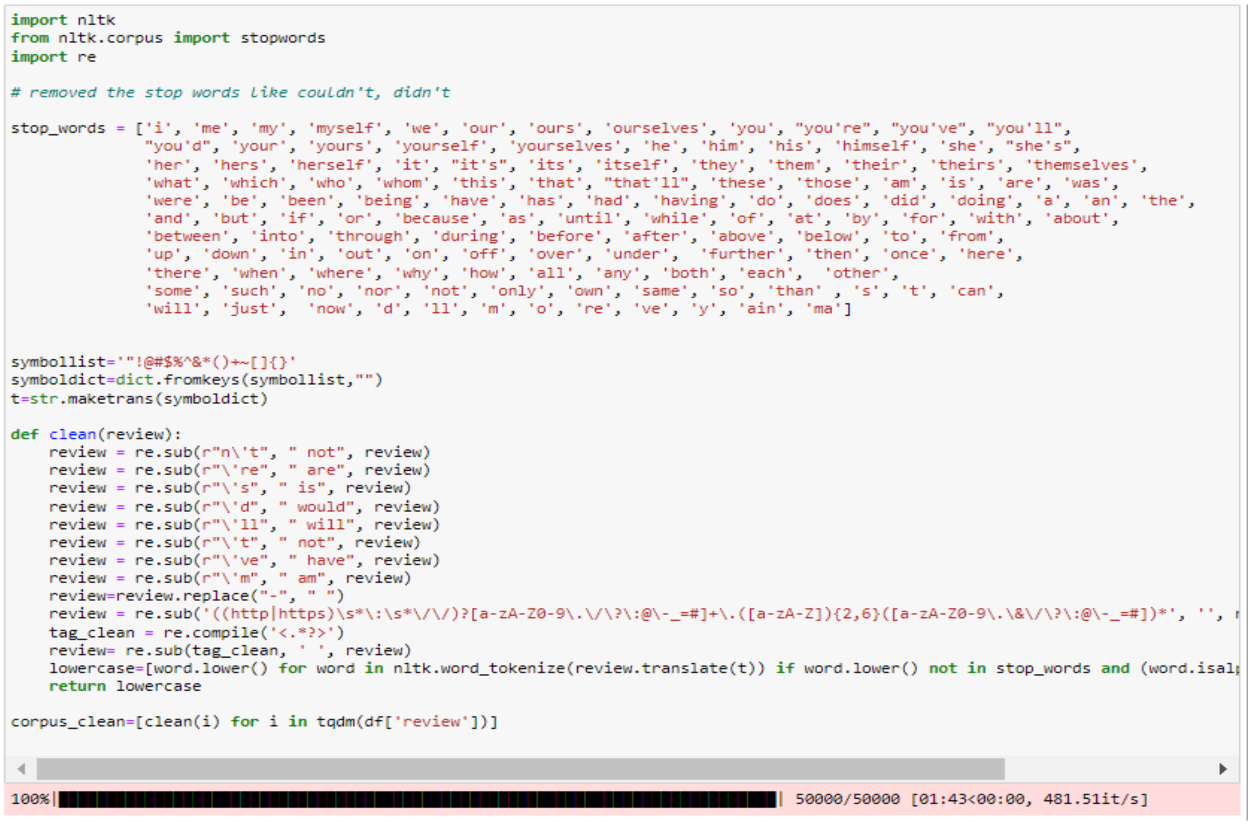
If we need to perform text classification or sentiment analysis then we should remove stop words as they do not provide any information to our model and this helps in keeping unwanted words out of our corpus.

Removing Stop words

Stop words are a major challenge in our project and generally stop words are any word in the stop list which are filtered out before or after processing of natural language text. Moreover, there is no single universal list of stop words used by all natural language processing tools. Stop words include (articles, prepositions, pronouns, conjunctions, etc.) which does not add much information to text. Examples of few stop words are 'the', 'what', 'a', 'but', 'will' and so on. Usually, these words are ignored when search results are displayed.

How we removed stop words in our Project

Using NLTK library:
The Natural language Toolkit or more commonly NLTK is a suite of libraries and programs for symbolic and statistical natural language processing for English written in Python programming language. It contains all text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning.
We have not removed all stop words like, "aren't", "couldn't", "very", "didn't", "doesn't. Rest all were removed from the reviews in the dataset.



```
import nltk
from nltk.corpus import stopwords
import re

# removed the stop words like couldn't, didn't

stop_words = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll",
              "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's",
              'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves',
              'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was',
              'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the',
              'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about',
              'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from',
              'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under',  'further', 'then', 'once', 'here',
              'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each',  'other',
              'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than' , 's', 't', 'can',
              'will', 'just',  'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'ma']


symbollist='"!@#$%^&*()+~[]{}'
symboldict=dict.fromkeys(symbollist,"")
t=str.maketrans(symboldict)

def clean(review):
    review = re.sub(r"n\'t", " not", review)
    review = re.sub(r"\'re", " are", review)
    review = re.sub(r"\'s", " is", review)
    review = re.sub(r"\'d", " would", review)
    review = re.sub(r"\'ll", " will", review)
    review = re.sub(r"\'t", " not", review)
    review = re.sub(r"\'ve", " have", review)
    review = re.sub(r"\'m", " am", review)
    review=review.replace("-", " ")
    review = re.sub('((http|https)\s*\:\s*\/\/)?[a-zA-Z0-9\.\/\?\:@\-_=#]+\.([a-zA-Z]){2,6}([a-zA-Z0-9\.\&\/\?\:@\-_=#])*', '', 
    tag_clean = re.compile('<.*?>')
    review= re.sub(tag_clean, ' ', review)
    lowercase=[word.lower() for word in nltk.word_tokenize(review.translate(t)) if word.lower() not in stop_words and (word.isal
    return lowercase

corpus_clean=[clean(i) for i in tqdm(df['review'])]
```

```
100%|██████████████████████████████████████████████████| 50000/50000 [01:43<00:00, 481.51it/s]
```

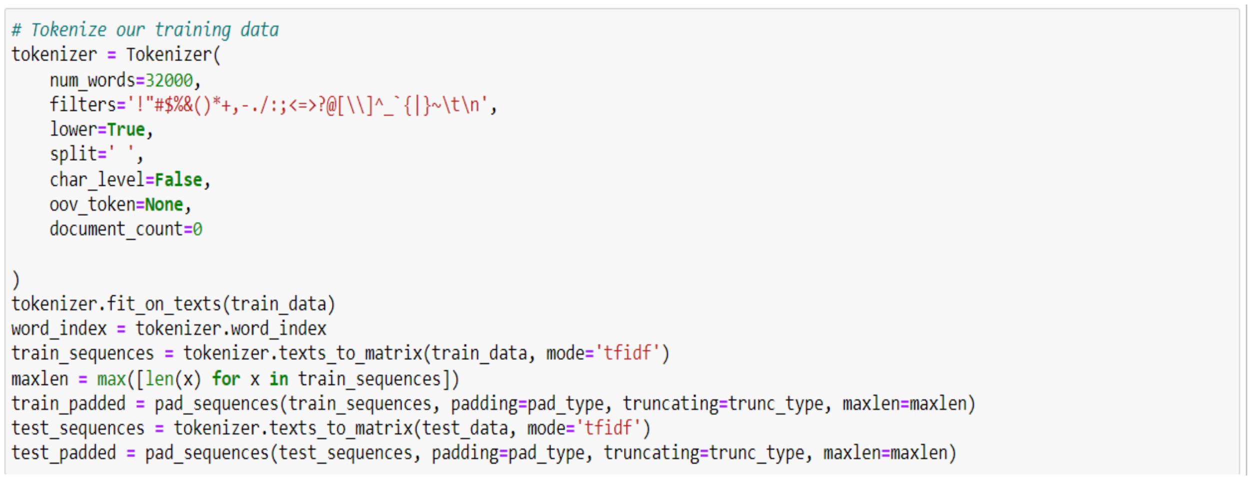
Fig 5: Cleaning Code

IV. DATA PREPARATION

Machine learning algorithms cannot work with raw text directly. Rather, the text must be converted into vectors of numbers. In natural language processing, a common technique for extracting features from text is tf-idf.

With tf-idf, instead of representing a term in a document by its raw frequency (number of occurrences) or its relative frequency (term count divided by document length), each term is weighted by dividing the term frequency by the number of documents in the corpus containing the word. The overall effect of this weighting scheme is to avoid a common problem when conducting text analysis: the most frequently used words in a document are often the most frequently used words in all of the documents. In contrast, terms with the highest tf-idf scores are the terms in a document that are distinctively frequent in a document, when that document is compared to other documents.

Below is how we have prepared our data for Artificial Neural Network:

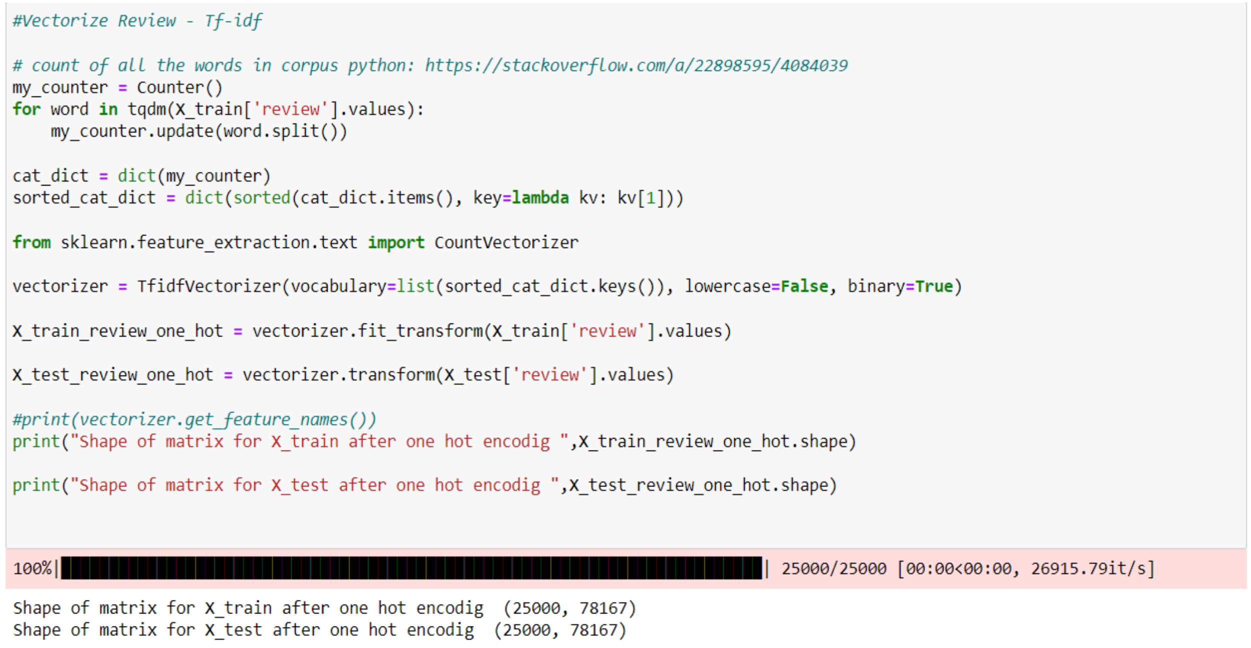


```
# Tokenize our training data
tokenizer = Tokenizer(
    num_words=32000,
    filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',
    lower=True,
    split=' ',
    char_level=False,
    oov_token=None,
    document_count=0

)
tokenizer.fit_on_texts(train_data)
word_index = tokenizer.word_index
train_sequences = tokenizer.texts_to_matrix(train_data, mode='tfidf')
maxlen = max([len(x) for x in train_sequences])
train_padded = pad_sequences(train_sequences, padding=pad_type, truncating=trunc_type, maxlen=maxlen)
test_sequences = tokenizer.texts_to_matrix(test_data, mode='tfidf')
test_padded = pad_sequences(test_sequences, padding=pad_type, truncating=trunc_type, maxlen=maxlen)
```

Fig 6: TF-IDF data preparation for ANN

Below is how we have prepared our data for Machine Learning Models:



```
#Vectorize Review - Tf-idf

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in tqdm(X_train['review'].values):
    my_counter.update(word.split())

cat_dict = dict(my_counter)
sorted_cat_dict = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))

from sklearn.feature_extraction.text import CountVectorizer

vectorizer = TfidfVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True)

X_train_review_one_hot = vectorizer.fit_transform(X_train['review'].values)

X_test_review_one_hot = vectorizer.transform(X_test['review'].values)

#print(vectorizer.get_feature_names())
print("Shape of matrix for X_train after one hot encodig ",X_train_review_one_hot.shape)

print("Shape of matrix for X_test after one hot encodig ",X_test_review_one_hot.shape)
```

```
100%|████████████████████████████████████████████████████████████████████████| 25000/25000 [00:00<00:00, 26915.79it/s]

Shape of matrix for X_train after one hot encodig  (25000, 78167)
Shape of matrix for X_test after one hot encodig  (25000, 78167)
```

Fig 7: TF-IDF data preparation for machine learning models

V. MODEL BUILDING

Model building refers to the process of deciding what models to use for the context.
Model building mainly concentrates on desired algorithms. The most famous technique in Model building is regression and there are many other techniques. The definition of a good model includes robustness and accuracy.
The model which we built in our project includes Text Blob, Multinomial Naive Bayes, Logistic Regression, Random Forest, SVM, Neural Network, LSTM and Accuracies were found to identify the best model.

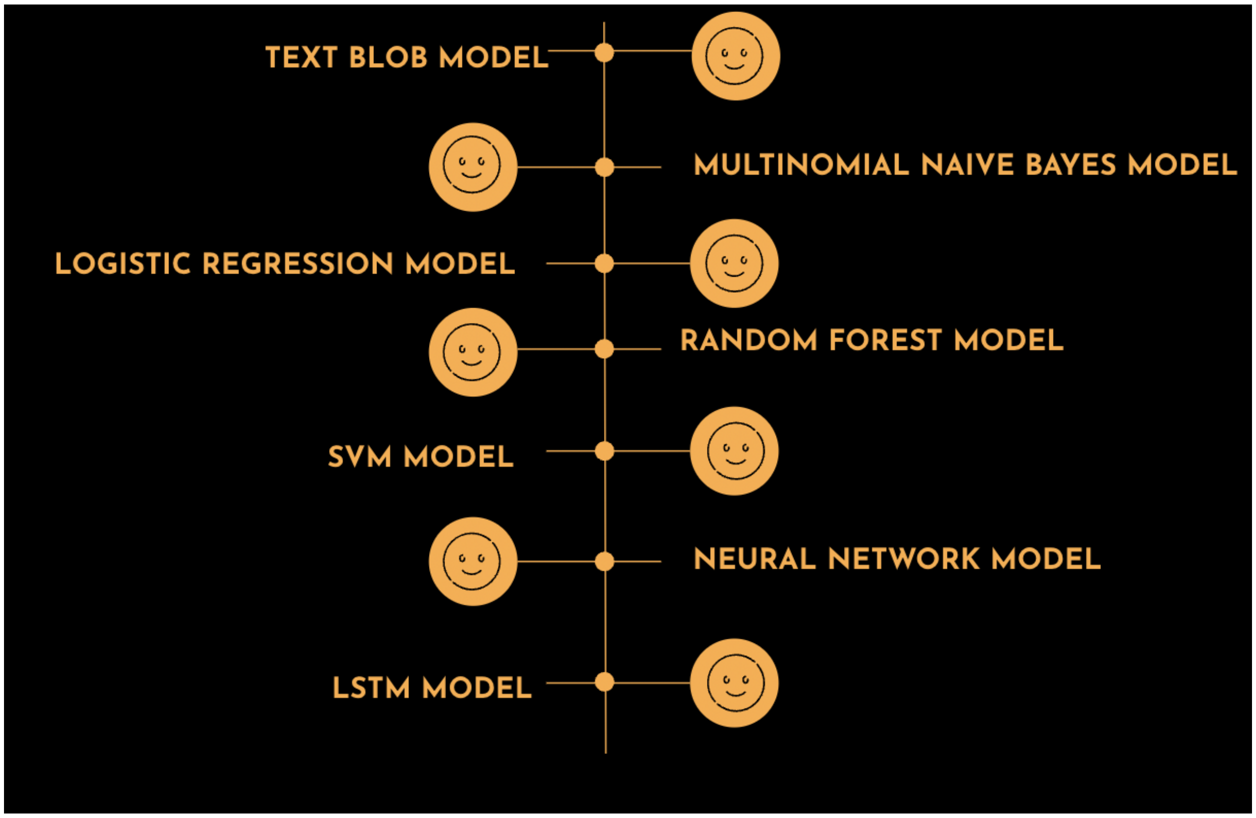


Fig 8: TF-IDF data preparation for machine learning models

TextBlob

Text Blob is a python library for Natural Language Processing (NLP).TextBlob actively uses Natural Language Toolkit (NLTK) to achieve its tasks. NLTK is a library which gives easy access to a lot of lexical resources and allows users to work with categorization, classification and many other tasks. TextBlob is a simple library which supports complex analysis and operations on textual data.

For lexicon-based approaches, a sentiment is defined by its semantic orientation and the intensity of each word in the sentence. This requires a pre-defined dictionary classifying negative and positive words. Generally, a text message will be represented by a bag of words. After assigning individual scores to all the words, final sentiment is calculated by some pooling operation like taking an average of all the sentiments.

TextBlob returns polarity and subjectivity of a sentence. Polarity lies between [-1,1], -1 defines a negative sentiment and 1 defines a positive sentiment. Negation words reverse the polarity. TextBlob has semantic labels that help with fine-grained analysis. For example — emoticons, exclamation marks, emojis, etc. Subjectivity lies between [0,1]. Subjectivity quantifies the amount of personal opinion and factual information contained in the text. The higher subjectivity means that the text contains personal opinion rather than factual information. TextBlob has one more parameter — intensity. TextBlob calculates subjectivity by looking at the 'intensity'. Intensity determines if a word modifies the next word. For English, adverbs are used as modifiers ('very good').

Model Accuracy: 76%

Multinomial Naïve Bayes

Multinomial Naive Bayes is one of the most popular supervised learning classifications that is used for the analysis of categorical text data.
Text data classification is gaining popularity because there is an enormous amount of information available in email, documents, websites, etc. that needs to be analyzed. Knowing the context around a certain type of text helps in finding the perception of a software or product to users who are going to use it.

Multinomial Naive Bayes algorithm is a probabilistic learning method that is mostly used in Natural Language Processing (NLP). The algorithm is based on the Bayes theorem and predicts the tag of a text such as a piece of email or newspaper article. It calculates the probability of each tag for a given sample and then gives the tag with the highest probability as output.

Naive Bayes classifier is a collection of many algorithms where all the algorithms share one common principle, and that is each feature being classified is not related to any other feature. The presence or absence of a feature does not affect the presence or absence of the other feature.

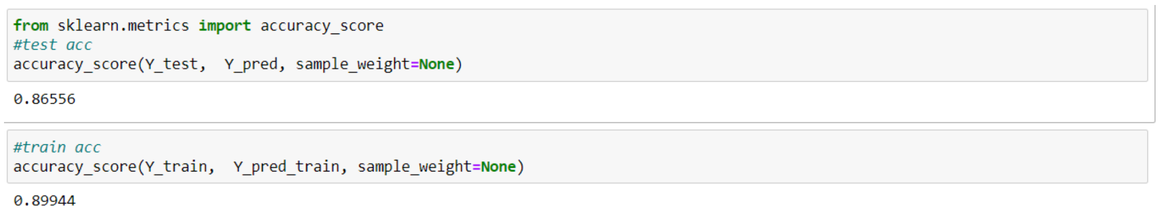
Naive Bayes is a powerful algorithm that is used for text data analysis and with problems with multiple classes. To understand Naive Bayes theorem's working, it is important to understand the Bayes theorem concept first as it is based on the latter.

Bayes theorem, formulated by Thomas Bayes, calculates the probability of an event occurring based on the prior knowledge of conditions related to an event. It is based on the following formula:

$P(A|B) = P(A) * P(B|A)/P(B)$

Where we are calculating the probability of class A when predictor B is already provided.
P(B) = prior probability of B
P(A) = prior probability of class A
P(B|A) = occurrence of predictor B given class A probability
This formula helps in calculating the probability of the tags in the text.

Output:



```
from sklearn.metrics import accuracy_score
#test acc
accuracy_score(Y_test,  Y_pred, sample_weight=None)
```

```
0.86556
```

```
#train acc
accuracy_score(Y_train,  Y_pred_train, sample_weight=None)
```

```
0.89944
```

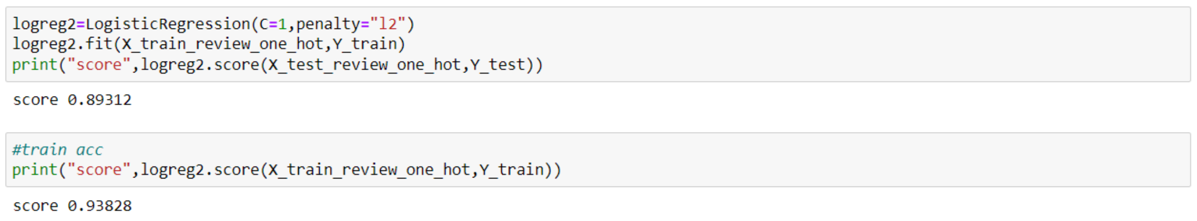
Fig 9: Multinomial Naive Bayes Accuracy Score

Model Accuracy: 86.55%

Logistic Regression

Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.). In other words, the logistic regression model predicts P(Y=1) as a function of X. It is used when the data is linearly separable, and the outcome is binary or dichotomous in nature.That means Logistic regression is usually used for Binary classification problems.
By default, logistic regression cannot be used for classification tasks that have more than two class labels, so-called multi-class classification. Instead, it requires modification to support multi-class classification problems.

Output:



```
logreg2=LogisticRegression(C=1,penalty="l2")
logreg2.fit(X_train_review_one_hot,Y_train)
print("score",logreg2.score(X_test_review_one_hot,Y_test))
```

```
score 0.89312
```

```
#train acc
print("score",logreg2.score(X_train_review_one_hot,Y_train))
```

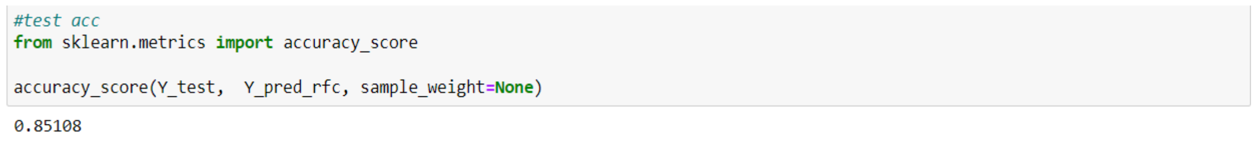
```
score 0.93828
```

Fig 10: Logistic Regression Accuracy Score

Model Accuracy: 89.31%

Random Forest

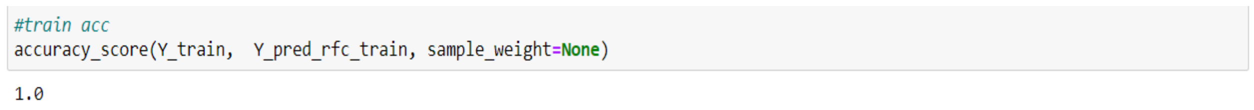
Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.
As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.
The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.
Output:



```
#test acc
from sklearn.metrics import accuracy_score

accuracy_score(Y_test,  Y_pred_rfc, sample_weight=None)
```

```
0.85108
```



```
#train acc
accuracy_score(Y_train,  Y_pred_rfc_train, sample_weight=None)
```

```
1.0
```

Fig 11: Accuracy Score of Random Forest Model

Model Accuracy: 85.10%

Support Vector Machine

A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an SVM model sets of labeled training data for each category, they're able to categorize new text.

Compared to newer algorithms like neural networks, they have two main advantages: higher speed and better performance with a limited number of samples (in the thousands). This makes the algorithm very suitable for text classification problems, where it's common to have access to a dataset of at most a couple of thousands of tagged samples.

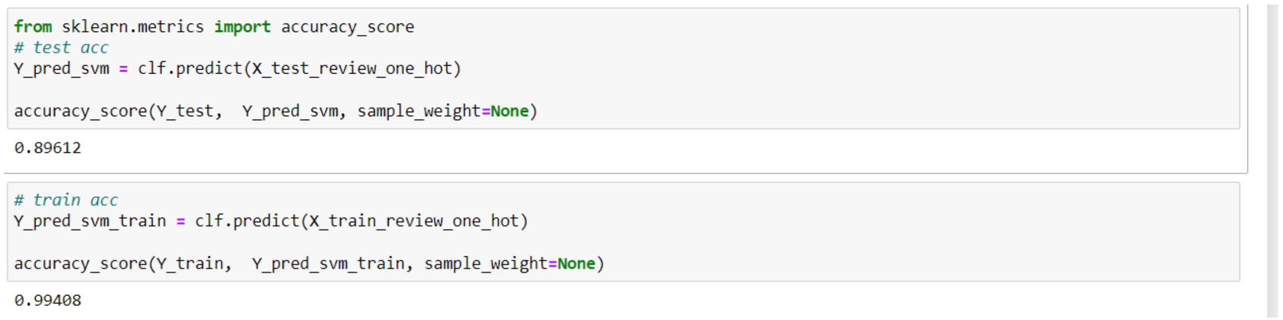
The SVM or Support Vector Machines algorithm just like the Naive Bayes algorithm can be used for classification purposes. So, we use SVM to mainly classify data but we can also use it for regression. It is a fast and dependable algorithm and works well with fewer data.

A very simple definition would be that SVM is a supervised algorithm that classifies or separates data using hyperplanes. So, this algorithm is a supervised algorithm in which we pass the data as well as the labels of the classes to the model.

Support vector machine is highly preferred by many as it produces significant accuracy with less computation power. Support Vector Machine can be used for both regression and classification tasks. But it is widely used in classification objectives.

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points.

Output:



```
from sklearn.metrics import accuracy_score
# test acc
Y_pred_svm = clf.predict(X_test_review_one_hot)

accuracy_score(Y_test,  Y_pred_svm, sample_weight=None)
```

```
0.89612
```

```
# train acc
Y_pred_svm_train = clf.predict(X_train_review_one_hot)

accuracy_score(Y_train,  Y_pred_svm_train, sample_weight=None)
```

```
0.99408
```

Fig 12: SVM Accuracy Score

Model Accuracy: 89.61%

Neural Network (ANN)

Neural Networks are a special type of machine learning algorithms that are modeled after the human brain. That is, just like how the neurons in our nervous system are able to learn from the past data, similarly, the NN is able to learn from the data and provide responses in the form of predictions or classifications.

NNs are nonlinear statistical models which display a complex relationship between the inputs and outputs to discover a new pattern. A variety of tasks such as image recognition, speech recognition, machine translation as well as medical diagnosis makes use of these artificial neural networks.

In a neural network, there are three essential layers –

1.
Input Layers
The input layer is the first layer of an ANN that receives the input information in the form of various texts, numbers, audio files, image pixels, etc.

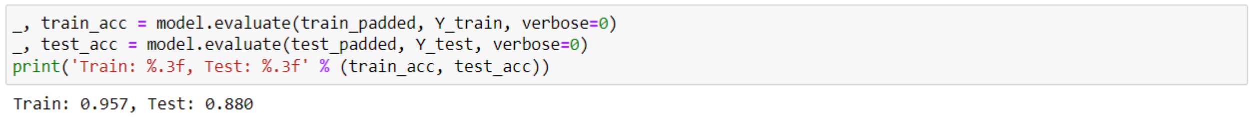
2.
Hidden Layers

In the middle of the ANN model are the hidden layers. There can be a single hidden layer, as in the case of a perceptron or multiple hidden layers. These hidden layers perform various types of mathematical computation on the input data and recognize the patterns that are part of.

3.
Output Layer
In the output layer, we obtain the result that we obtain through rigorous computations performed by the middle layer.

In a neural network, there are multiple parameters and hyperparameters that affect the performance of the model. The output of ANNs is mostly dependent on these parameters. Some of these parameters are weights, biases, learning rate, batch size etc. Each node in the ANN has some weight.
Each node in the network has some weights assigned to it. A transfer function is used for calculating the weighted sum of the inputs and the bias.

Output:



```
_, train_acc = model.evaluate(train_padded, Y_train, verbose=0)
_, test_acc = model.evaluate(test_padded, Y_test, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
```

```
Train: 0.957, Test: 0.880
```

Fig 13: Neural Network (ANN) Accuracy Score

Model Accuracy: 88%

LSTM

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems. It is a variety of recurrent neural networks (RNNs) that are capable of learning long-term dependencies, especially in sequence prediction problems. LSTM has feedback connections, i.e., it is capable of processing the entire sequence of data, apart from single data points such as images. This finds application in speech recognition, machine translation, etc. LSTM is a special kind of RNN, which shows outstanding performance on a large variety of problems.

Recurrent neural networks, of which LSTMs ("long short-term memory" units) are the most powerful and well-known subset, are a type of artificial neural network designed to recognize patterns in sequences of data, such as numerical times series data emanating from sensors, stock markets and government agencies (but also including text, genomes, handwriting and the spoken word). What differentiates RNNs and LSTMs from other neural networks is that they take time and sequence into account, they have a temporal dimension.

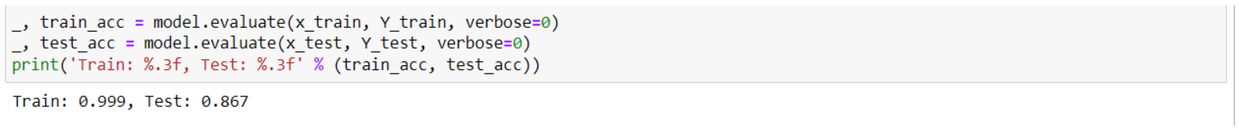
LSTM has a chain structure that contains four neural networks and different memory blocks called cells. Information is retained by the cells and the memory manipulations are done by the gates. There are three gates –

1.
Forget Gate: The information that is no longer useful in the cell state is removed with the forget gate. Two inputs x_t (input at the particular time) and h_t-1 (previous cell output) are fed to the gate and multiplied with weight matrices followed by the addition of bias. The resultant is passed through an activation function which gives a binary output. If for a particular cell state the output is 0, the piece of information is forgotten and for output 1, the information is retained for future use.

2.
Input gate: The addition of useful information to the cell state is done by the input gate. First, the information is regulated using the sigmoid function and filters the values to be remembered similar to the forget gate using inputs h_t-1 and x_t. Then, a vector is created using the tanh function that gives an output from -1 to +1, which contains all the possible values from h_t-1 and x_t. At last, the values of the vector and the regulated values are multiplied to obtain the useful information

3.
Output gate: The task of extracting useful information from the current cell state to be presented as output is done by the output gate. First, a vector is generated by applying tanh function on the cell. Then, the information is regulated using the sigmoid function and filtered by the values to be remembered using inputs h_t-1 and x_t. At last, the values of the vector and the regulated values are multiplied to be sent as an output and input to the next cell.

Output:



```
_, train_acc = model.evaluate(x_train, Y_train, verbose=0)
_, test_acc = model.evaluate(x_test, Y_test, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
```

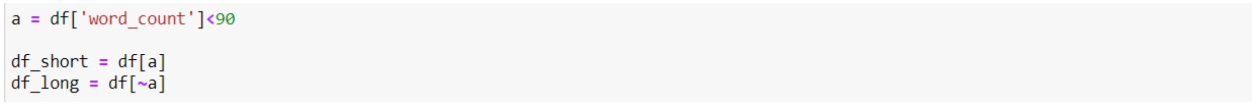
```
Train: 0.999, Test: 0.867
```

Fig 14: Accuracy Score of LSTM

Model Accuracy: 86.7%

Model building on separate dataset- IMDb_long and IMDb_short

Here we have divided our imdb dataset into two separate dataset on the basis of the number of words in each review. df_short contains reviews having word count less than 90 and df_long contains reviews having word count greater or equal to 90.



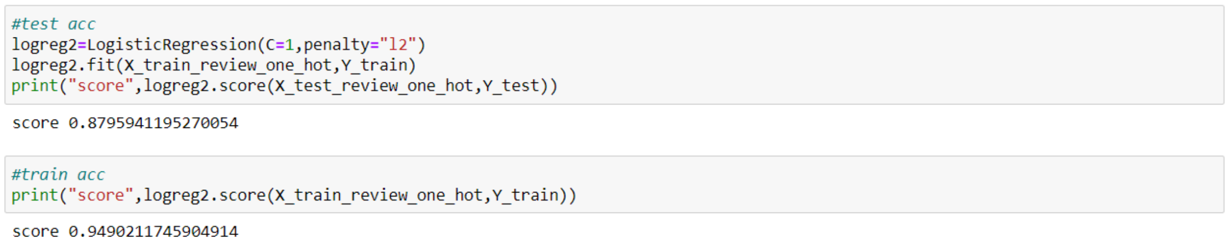
```
a = df['word_count']<90

df_short = df[a]
df_long = df[~a]
```

Here, we performed all the necessary steps and built all the models for both separate data sets as we did for the combined dataset.

The Logistic Regression performed best for both types of datasets.

Case1: df_long(where reviews word count >=90)



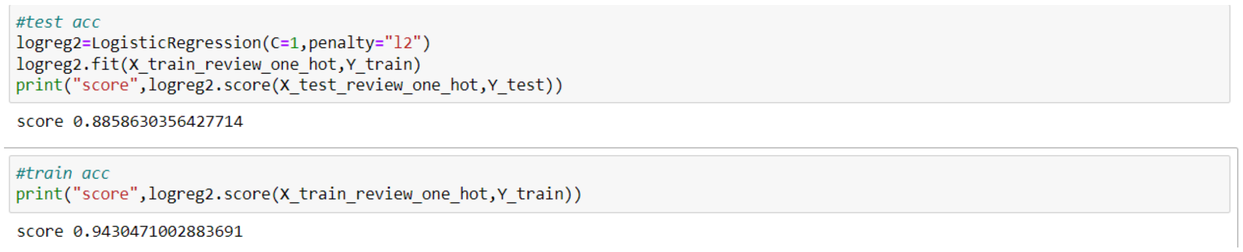
```
#test acc
logreg2=LogisticRegression(C=1,penalty="l2")
logreg2.fit(X_train_review_one_hot,Y_train)
print("score",logreg2.score(X_test_review_one_hot,Y_test))
```

```
score 0.8795941195270054
```

```
#train acc
print("score",logreg2.score(X_train_review_one_hot,Y_train))
```

```
score 0.9490211745904914
```

Case2: df_long(where reviews word count <90)



```
#test acc
logreg2=LogisticRegression(C=1,penalty="l2")
logreg2.fit(X_train_review_one_hot,Y_train)
print("score",logreg2.score(X_test_review_one_hot,Y_test))
```

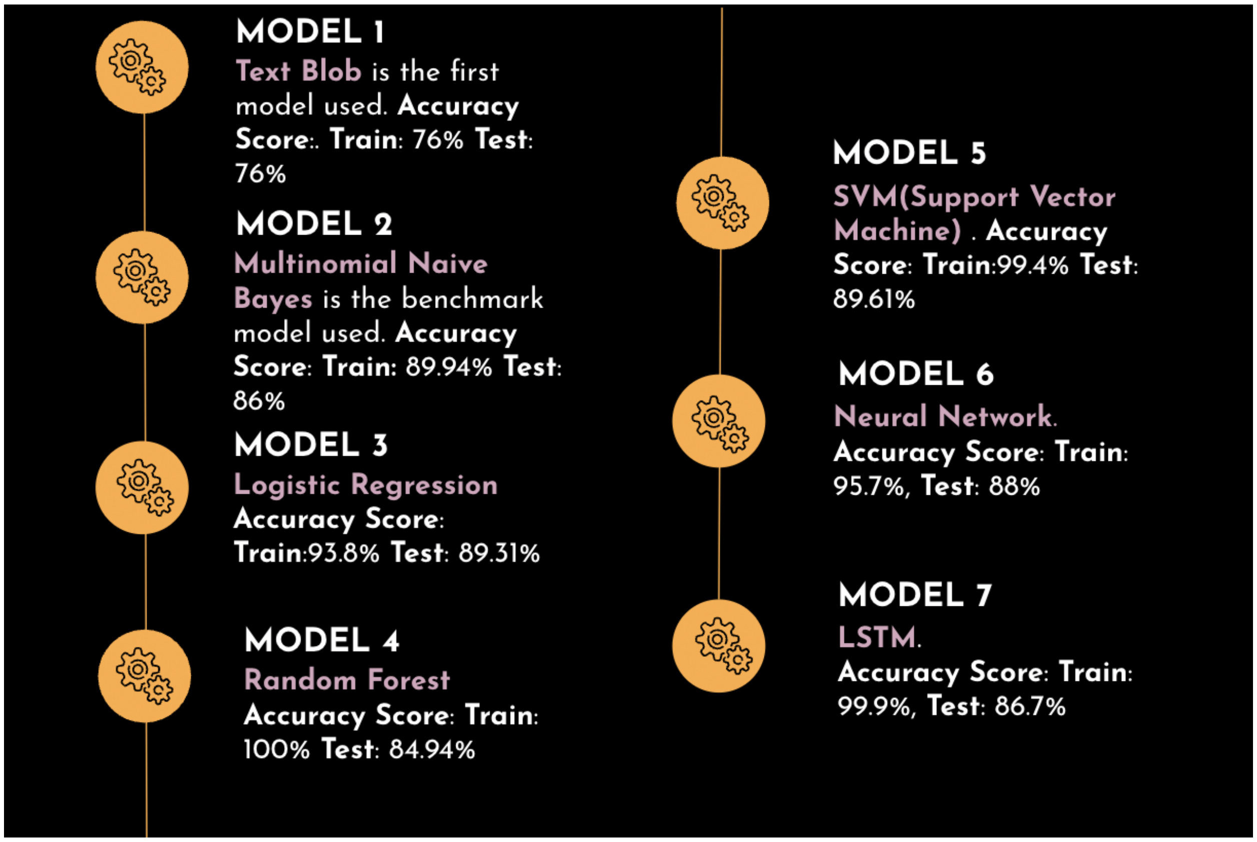
```
score 0.8858630356427714
```

```
#train acc
print("score",logreg2.score(X_train_review_one_hot,Y_train))
```

```
score 0.9430471002883691
```

VI. MODELS COMPARISON

Here, the following images displays the summary of all the models that we have used for analysis of the IMDb dataset in a manner that represents the Accuracy scores for training and test datasets.



MODEL 1
Text Blob is the first model used. **Accuracy Score**:. **Train**: 76% **Test**: 76%

MODEL 2
Multinomial Naive Bayes is the benchmark model used. **Accuracy Score**: **Train:** 89.94% **Test**: 86%

MODEL 3
Logistic Regression
Accuracy Score:
Train:93.8% **Test**: 89.31%

MODEL 4
Random Forest
Accuracy Score: **Train**: 100% **Test**: 84.94%

MODEL 5
SVM(Support Vector Machine) . **Accuracy Score**: **Train**:99.4% **Test**: 89.61%

MODEL 6
Neural Network.
Accuracy Score: **Train**: 95.7%, **Test**: 88%

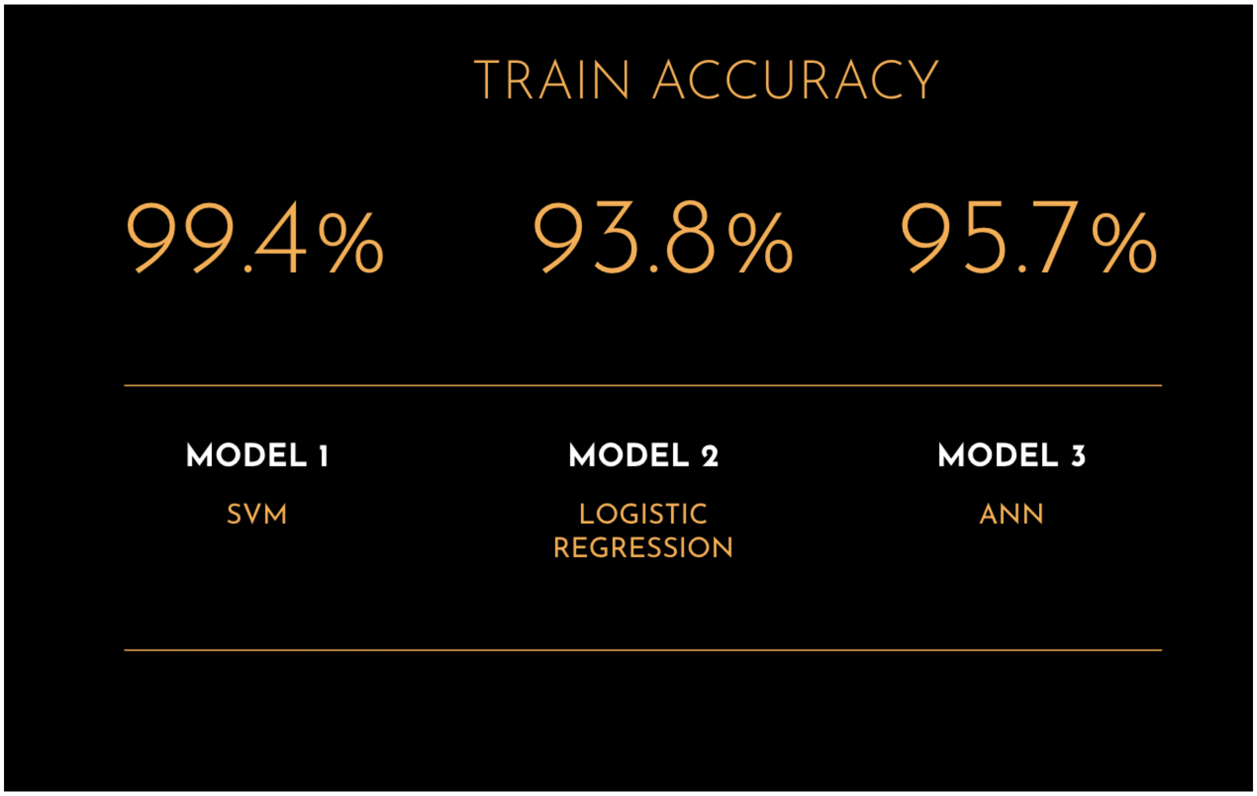
MODEL 7
LSTM.
Accuracy Score: **Train**: 99.9%, **Test**: 86.7%

Fig 15: Details of Models

VII. CONCLUSION

After Finding the Accuracies for all the Models mentioned above we choose the model with best accuracy in order to predict the output accurately. The top 3 model we have with best accuracy are
SVM, Logistic Regression and ANN.
Among all three models, the model which we chose is Logistic Regression because it overfits less as compared to ANN and SVM.

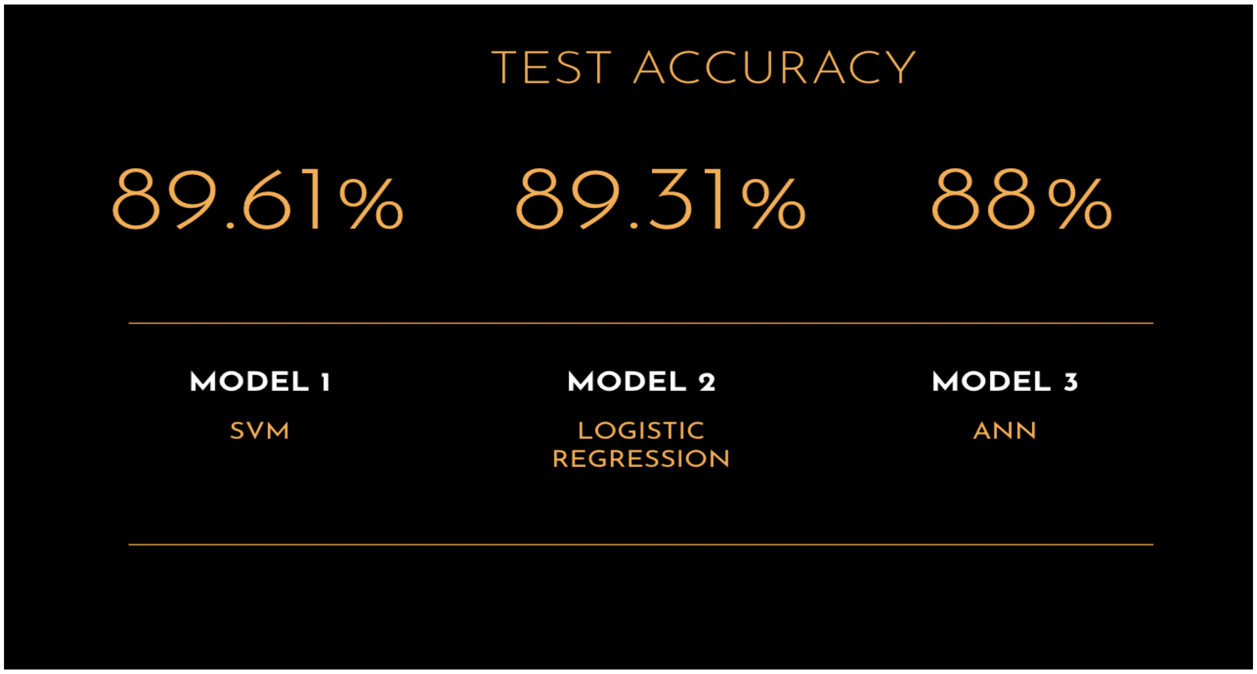
For our separate analyses we did on IMDb_long and on IMDb_short, we found that there is not much difference in the test accuracy and train accuracy for both the data set.
The models performed nearly the same on both the data sets, only with the difference of 0.6 percent.



TRAIN ACCURACY

99.4%	93.8%	95.7%
MODEL 1	**MODEL 2**	**MODEL 3**
SVM	LOGISTIC REGRESSION	ANN

Fig 16: Train Accuracy of top 3 models



TEST ACCURACY

89.61%	89.31%	88%
MODEL 1	**MODEL 2**	**MODEL 3**
SVM	LOGISTIC REGRESSION	ANN

Fig 17: Test Accuracy of top 3 models

VIII. APPENDIX

Performance of models used in this analysis is pictorially represented in the following image. It is represented in the best to worst format.

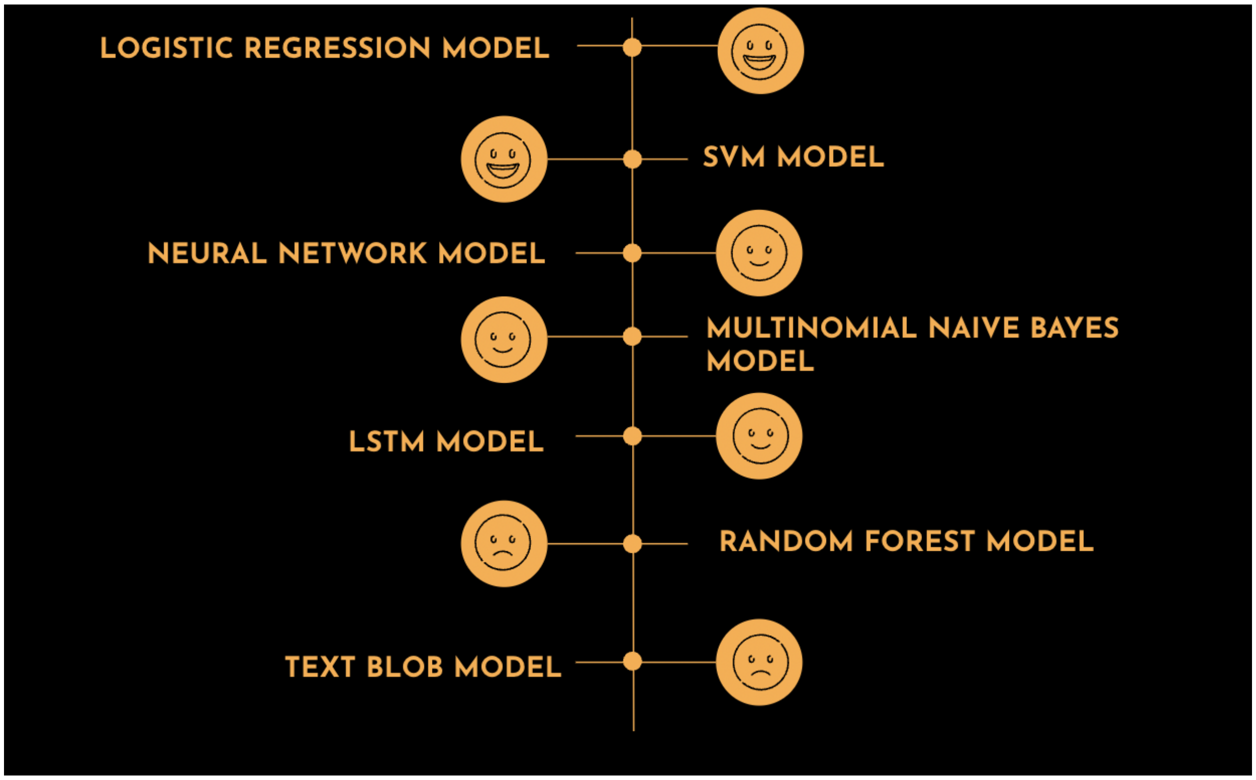


Fig 18: Performance of Models from best to worst

IX. REFERENCES

1. Large Movie Review Dataset by Stanford EDU: https://ai.stanford.edu/~amaas/data/sentiment/
2. Sentiment Analysis of IMDB Reviews with NLP: https://www.analyticsvidhya.com/blog/2022/02/sentiment-analysis-of-imdb-reviews-with-nlp/
3. Performing Sentiment Analysis on Movie Reviews - The essentials of TD-IDF and tokenization with scikit-learn: https://towardsdatascience.com/imdb-reviews-or-8143fe57c825
4. Sentiment Analysis: https://paperswithcode.com/sota/sentiment-analysis-on-imdb
5. IMDb Movie Reviews: https://paperswithcode.com/dataset/imdb-movie-reviews
6. Sentiment Analysis on IMDB Movie Review: https://medium.com/@pyashpq56/sentiment-analysis-on-imdb-movie-review-d004f3e470bd
7. Sentiment Analysis: https://github.com/Ankit152/IMDB-sentiment-analysis
8. Netflix Movies and TV Shows: https://www.kaggle.com/datasets/shivamb/netflix-shows
9. Sentimental Analysis of Movie Reviews: https://pianalytix.com/sentimental-analysis-of-imdb-movie-reviews/
10. Learning Word Vectors for Sentiment Analysis: http://ai.stanford.edu/~amaas/papers/wvSent_acl2011.pdf