IMDB Dataset - Classification Models Comparison

1. Problem Statement

Binary sentiment classification is analyzing customers opinion such as online reviews or survey responses as positive or negative. It helps to understand how well a product or service is doing in the market and helps stakeholders to take swift action based on the analysis. In this project we will analyze movie reviews from IMDB and classify them into positive or negative using various Classification models and compare their performances.

2. Data Wrangling

Dataset

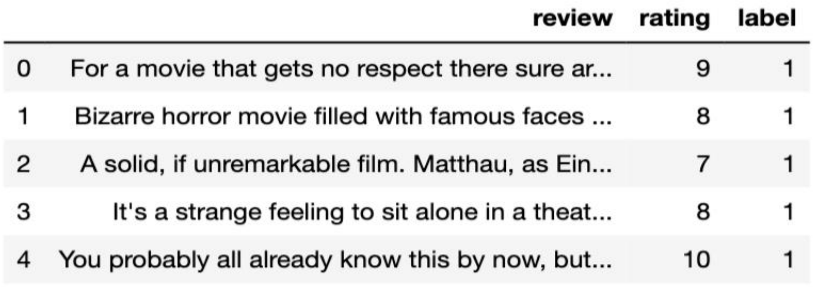
IMDB Dataset is a set of pre-classified movie reviews for training and testing provided by Stanford AI Lab. This dataset contains movie reviews along with their associated binary sentiment polarity labels that serves as a benchmark for sentiment classification.

There are totally 50,000 reviews split evenly into 25k train and 25k test sets with balanced distribution of labels (12500 positive and 12500 negative reviews in each set). Reviews are stored in text files named following the convention [[id]_[rating].txt] where [id] is a unique id and [rating] is the star rating for that review on a 1-10 scale. A review is labeled negative if it has a score <= 4 out of 10, and positive if it has a score >= 7 out of 10. Reviews with more neutral ratings are not included in the train/test sets.

Train and Test Dataframe

Train and test data frames *imdb_train* and *imdb_test* are created by reading individual review files using file *read()*. Both the datasets have these columns,

- 'review' consists of the review text
- 'label' is set to 1 or 0 for positive and negative reviews respectively
- 'rating' has the star rating extracted from the filenames of each review, as explained in the previous section



	review	rating	label
0	For a movie that gets no respect there sure ar...	9	1
1	Bizarre horror movie filled with famous faces ...	8	1
2	A solid, if unremarkable film. Matthau, as Ein...	7	1
3	It's a strange feeling to sit alone in a theat...	8	1
4	You probably all already know this by now, but...	10	1

Fig1: imdb_train and imdb_test data frame format

3. EDA

Distribution of Reviews

- It is mentioned in the readme file of the source dataset that, "A review is labeled negative if it has a score <= 4 out of 10, and positive if it has a score >= 7 out of 10. Reviews with more neutral ratings are not included in the train/test sets." This is verified below using Scatter plot of *rating* vs *label* on *imdb_train* and *imdb_test* dataset.

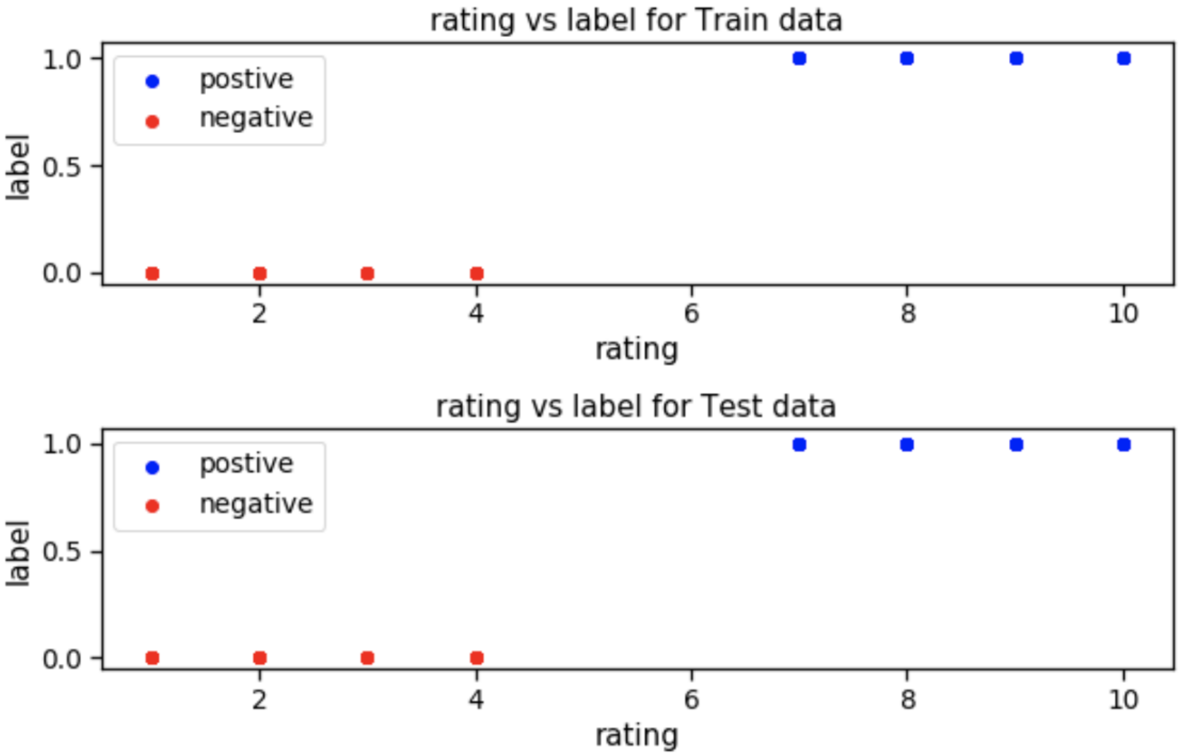


Fig 7: Scatter plot of rating vs label showing scores <=4 labeled as negative reviews and scores >=7 labelled as positive

- Below plot shows that we have high volume of reviews with ratings 1 & 10 in both train and test data.

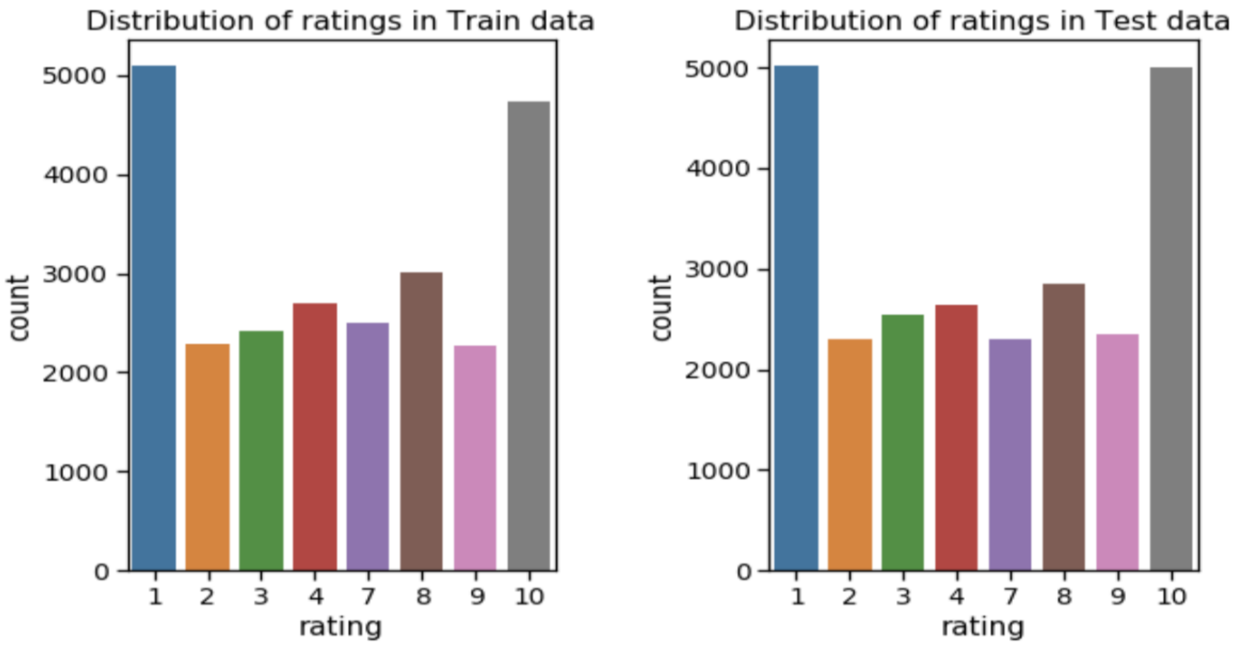


Fig 8: Plot of no. of reviews in each rating

- Positive and negative reviews are equally distributed in both train and test dataset.

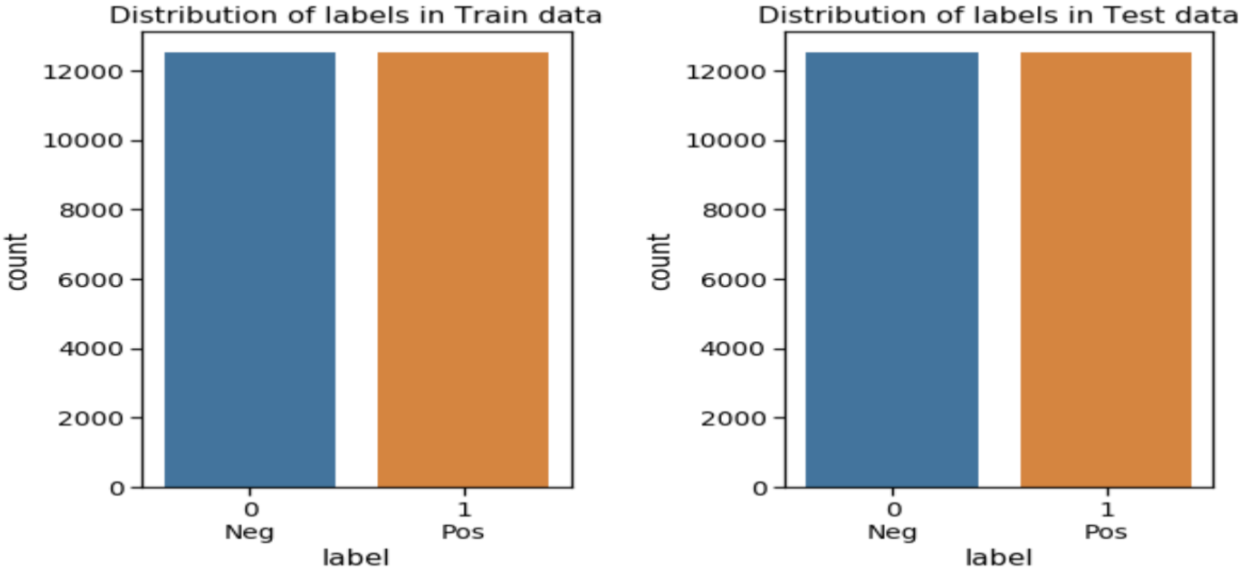
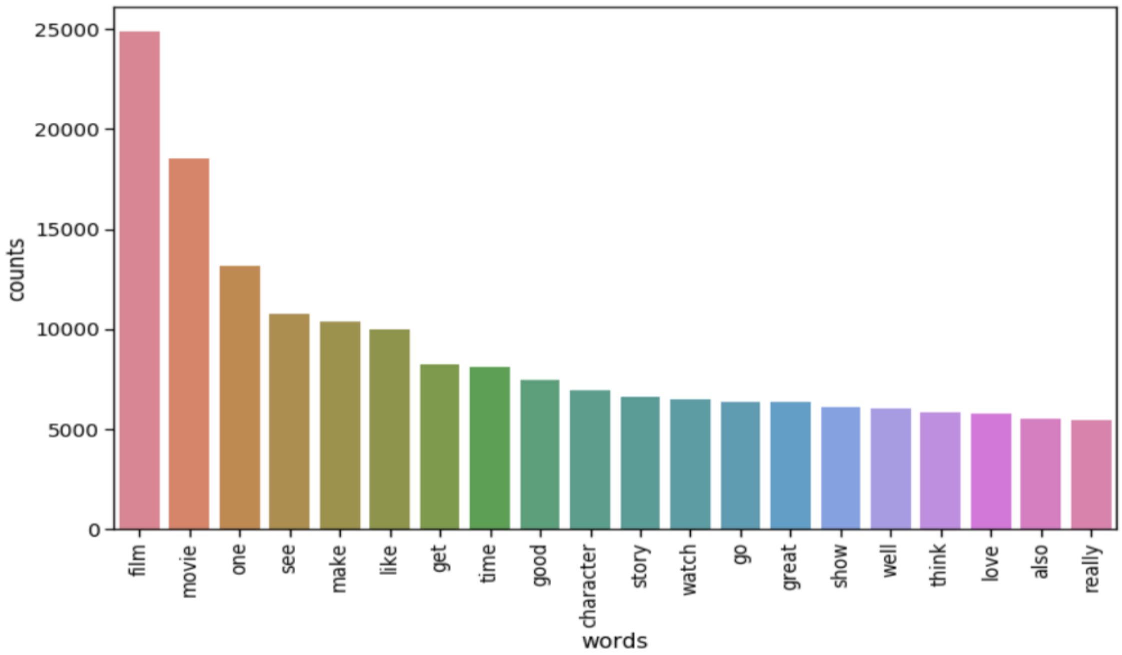


Fig 9: Distribution of positive and negative reviews

Most Common Words

20 most common words of the positive and negative reviews are plotted. Bi-grams can capture contextual information compared to just unigrams as seen below.

```
Top 20 unigrams in positive reviews:
```



```
Top 20 bigrams in positive reviews:
```

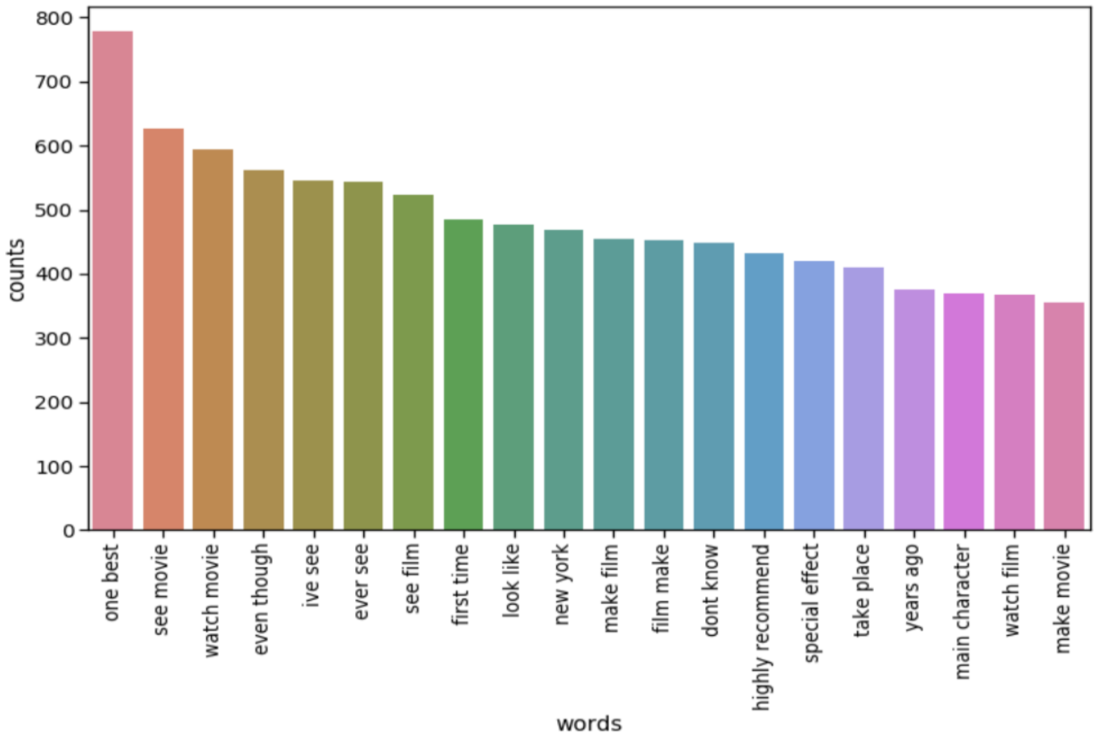
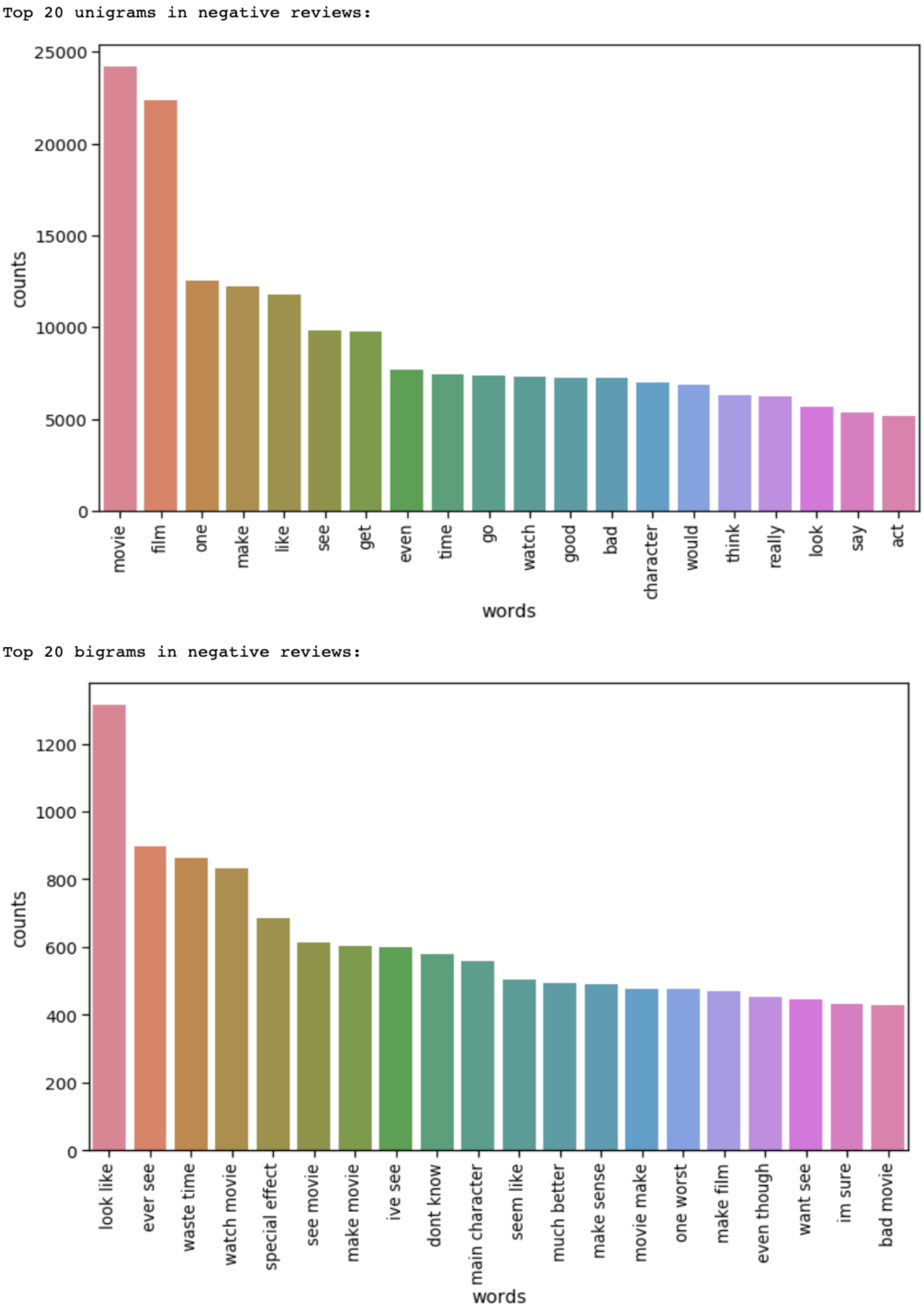


Fig 10a: 20 most common unigrams & bigrams in the positive reviews



```
Top 20 unigrams in negative reviews:
```

```
Top 20 bigrams in negative reviews:
```

Fig 10b: 20 most common unigrams & bigrams in the negative reviews

Word Cloud

Here is the mandatory word cloud that an NLP project must have. Word cloud is an image composed of words used in a particular text or subject, in which the size of each word indicates its frequency.

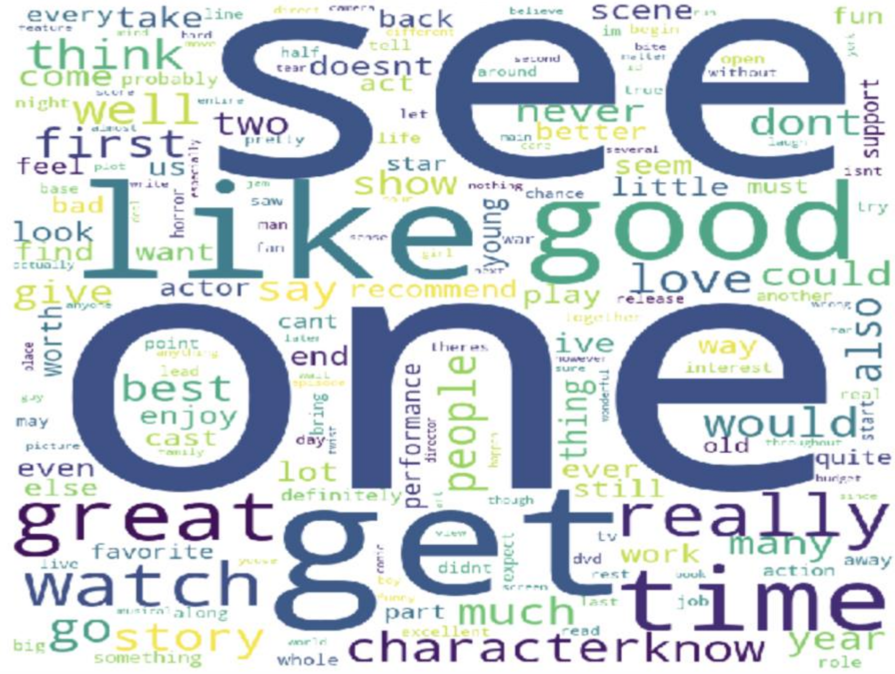


Fig 11a: Word cloud of positive reviews

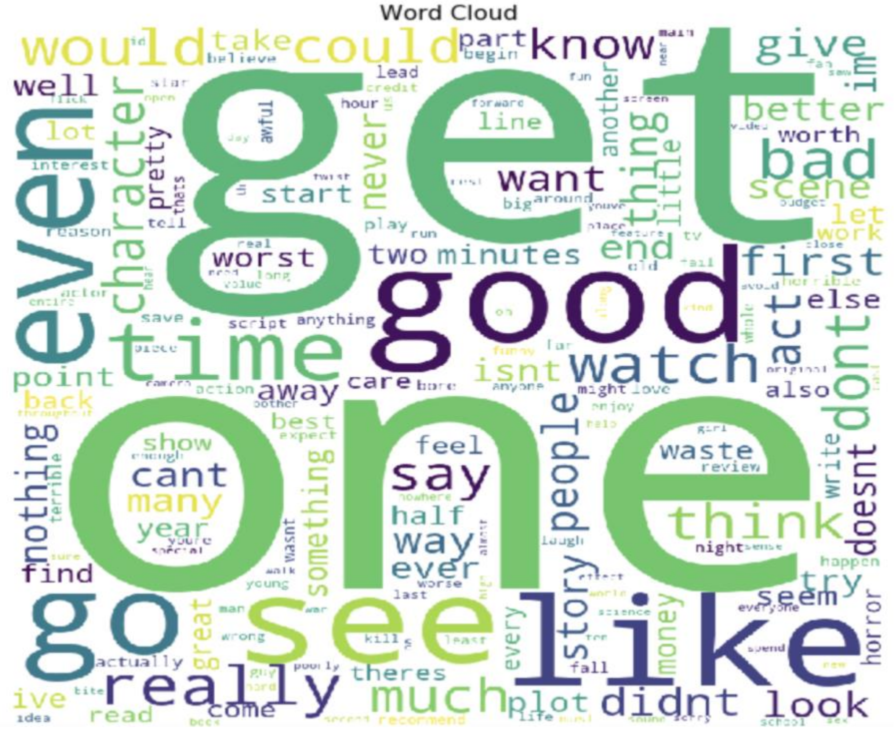
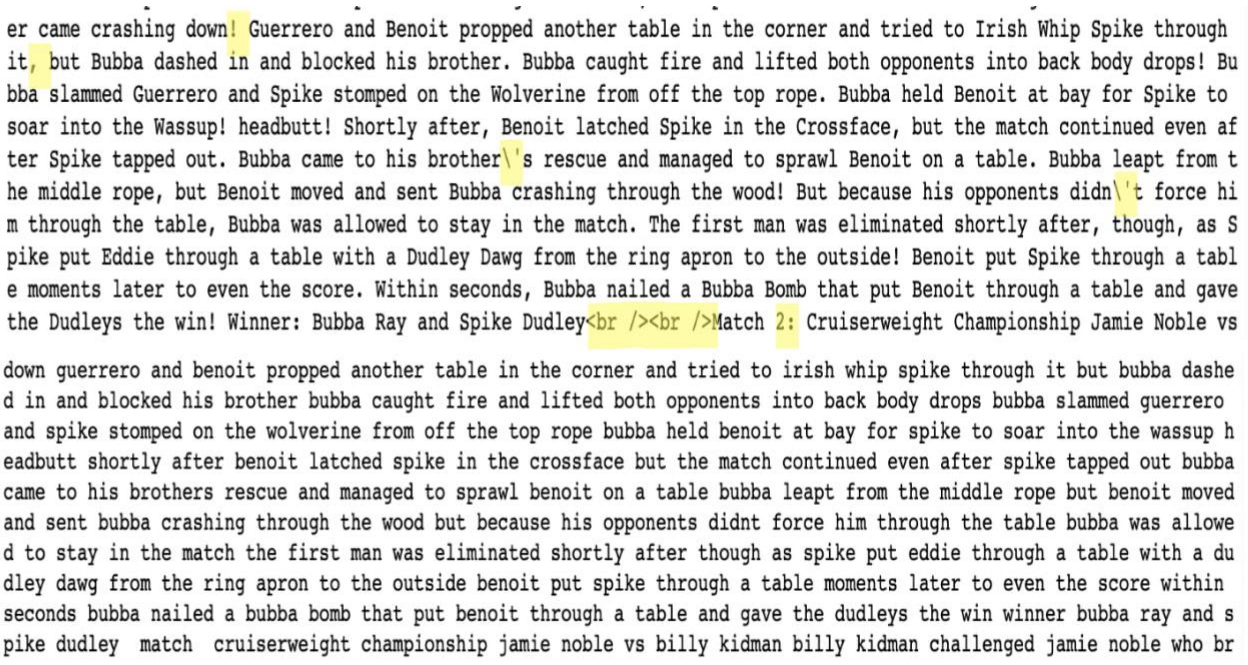


Fig 11b: Word cloud of negative reviews

4. Text Preprocessing

Text Normalization

user-defined function *normalize_text()* is applied to train and test dataset's review text, which converts all letters to lowercase and removes all unwanted characters like special symbols and numbers.



```
er came crashing down! Guerrero and Benoit propped another table in the corner and tried to Irish Whip Spike through
it, but Bubba dashed in and blocked his brother. Bubba caught fire and lifted both opponents into back body drops! Bu
bba slammed Guerrero and Spike stomped on the Wolverine from off the top rope. Bubba held Benoit at bay for Spike to
soar into the Wassup! headbutt! Shortly after, Benoit latched Spike in the Crossface, but the match continued even af
ter Spike tapped out. Bubba came to his brother\'s rescue and managed to sprawl Benoit on a table. Bubba leapt from t
he middle rope, but Benoit moved and sent Bubba crashing through the wood! But because his opponents didn\'t force hi
m through the table, Bubba was allowed to stay in the match. The first man was eliminated shortly after, though, as S
pike put Eddie through a table with a Dudley Dawg from the ring apron to the outside! Benoit put Spike through a tabl
e moments later to even the score. Within seconds, Bubba nailed a Bubba Bomb that put Benoit through a table and gave
the Dudleys the win! Winner: Bubba Ray and Spike Dudley<br /><br />Match 2: Cruiserweight Championship Jamie Noble vs
```

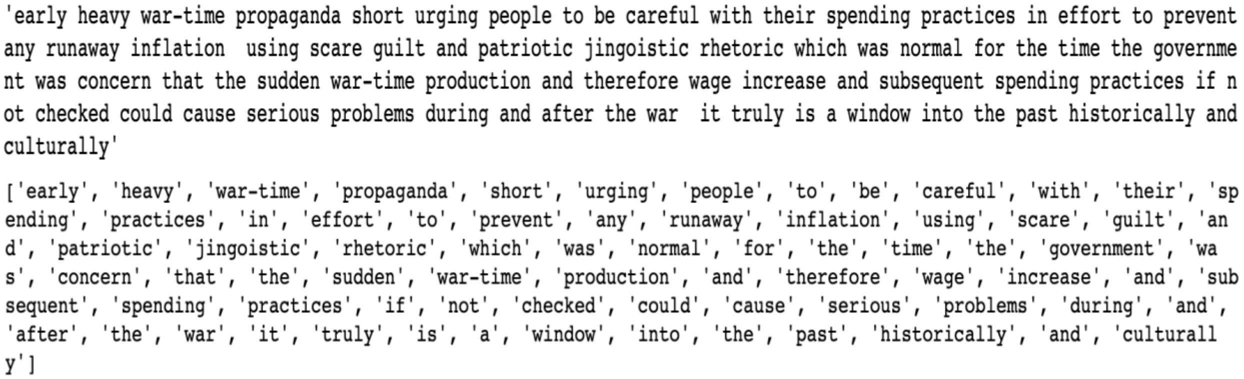
```
down guerrero and benoit propped another table in the corner and tried to irish whip spike through it but bubba dashe
d in and blocked his brother bubba caught fire and lifted both opponents into back body drops bubba slammed guerrero
and spike stomped on the wolverine from off the top rope bubba held benoit at bay for spike to soar into the wassup h
eadbutt shortly after benoit latched spike in the crossface but the match continued even after spike tapped out bubba
came to his brothers rescue and managed to sprawl benoit on a table bubba leapt from the middle rope but benoit moved
and sent bubba crashing through the wood but because his opponents didnt force him through the table bubba was allowe
d to stay in the match the first man was eliminated shortly after though as spike put eddie through a table with a du
dley dawg from the ring apron to the outside benoit put spike through a table moments later to even the score within
seconds bubba nailed a bubba bomb that put benoit through a table and gave the dudleys the win winner bubba ray and s
pike dudley  match  cruiserweight championship jamie noble vs billy kidman billy kidman challenged jamie noble who br
```

Fig 2: Normalization process on sample review text

nltk package

word tokenizer

word_tokenize() from *nltk.tokenize* package splits the reviews into list of words.



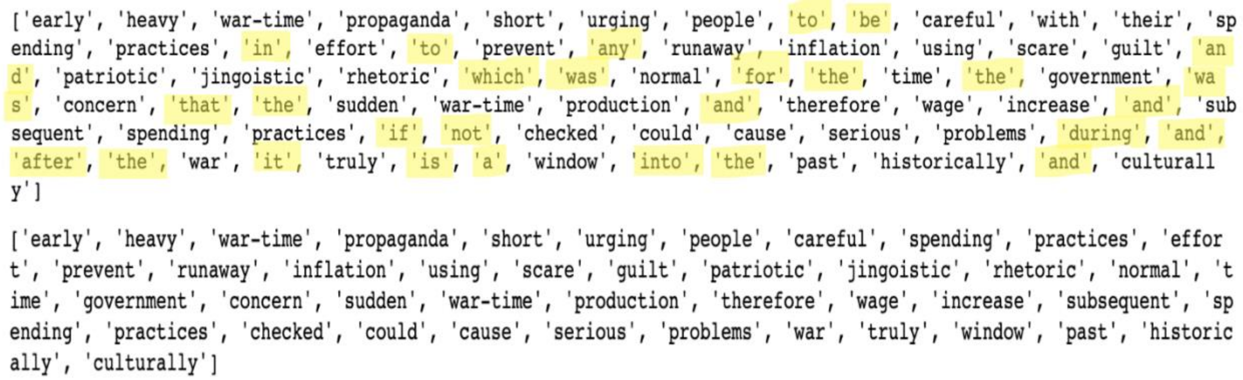
```
'early heavy war-time propaganda short urging people to be careful with their spending practices in effort to prevent
any runaway inflation  using scare guilt and patriotic jingoistic rhetoric which was normal for the time the governme
nt was concern that the sudden war-time production and therefore wage increase and subsequent spending practices if n
ot checked could cause serious problems during and after the war  it truly is a window into the past historically and
culturally'
```

```
['early', 'heavy', 'war-time', 'propaganda', 'short', 'urging', 'people', 'to', 'be', 'careful', 'with', 'their', 'sp
ending', 'practices', 'in', 'effort', 'to', 'prevent', 'any', 'runaway', 'inflation', 'using', 'scare', 'guilt', 'an
d', 'patriotic', 'jingoistic', 'rhetoric', 'which', 'was', 'normal', 'for', 'the', 'time', 'the', 'government', 'wa
s', 'concern', 'that', 'the', 'sudden', 'war-time', 'production', 'and', 'therefore', 'wage', 'increase', 'and', 'sub
sequent', 'spending', 'practices', 'if', 'not', 'checked', 'could', 'cause', 'serious', 'problems', 'during', 'and',
'after', 'the', 'war', 'it', 'truly', 'is', 'a', 'window', 'into', 'the', 'past', 'historically', 'and', 'culturall
y']
```

Fig 3: Word tokenization process on sample review text

Stop Words Removal

Stop words are commonly used words such as to, be, is etc. These stop words do not add any value to our models but just takes up space as features. In our review texts such words are removed using *stopwords* corpus from the *nltk* package.



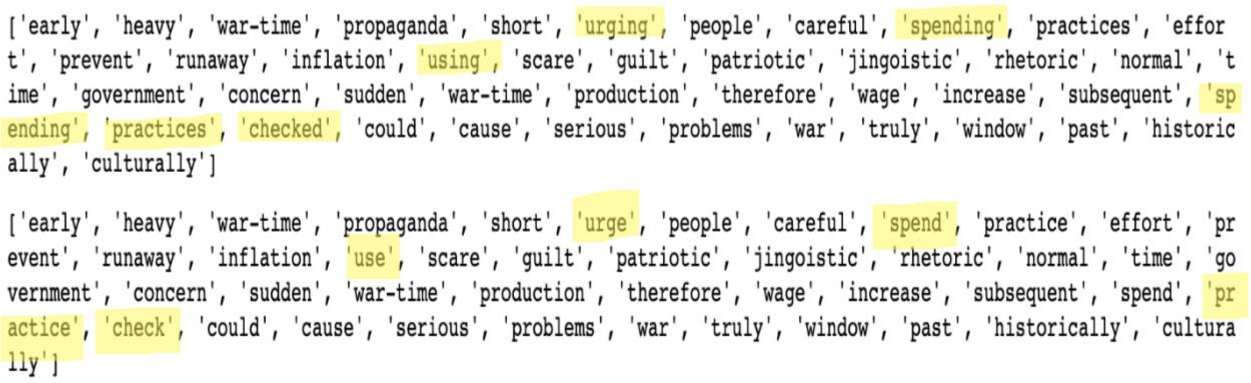
```
['early', 'heavy', 'war-time', 'propaganda', 'short', 'urging', 'people', 'to', 'be', 'careful', 'with', 'their', 'sp
ending', 'practices', 'in', 'effort', 'to', 'prevent', 'any', 'runaway', 'inflation', 'using', 'scare', 'guilt', 'an
d', 'patriotic', 'jingoistic', 'rhetoric', 'which', 'was', 'normal', 'for', 'the', 'time', 'the', 'government', 'wa
s', 'concern', 'that', 'the', 'sudden', 'war-time', 'production', 'and', 'therefore', 'wage', 'increase', 'and', 'sub
sequent', 'spending', 'practices', 'if', 'not', 'checked', 'could', 'cause', 'serious', 'problems', 'during', 'and',
'after', 'the', 'war', 'it', 'truly', 'is', 'a', 'window', 'into', 'the', 'past', 'historically', 'and', 'culturall
y']

['early', 'heavy', 'war-time', 'propaganda', 'short', 'urging', 'people', 'careful', 'spending', 'practices', 'effor
t', 'prevent', 'runaway', 'inflation', 'using', 'scare', 'guilt', 'patriotic', 'jingoistic', 'rhetoric', 'normal', 't
ime', 'government', 'concern', 'sudden', 'war-time', 'production', 'therefore', 'wage', 'increase', 'subsequent', 'sp
ending', 'practices', 'checked', 'could', 'cause', 'serious', 'problems', 'war', 'truly', 'window', 'past', 'historic
ally', 'culturally']
```

Fig 4: Stop Words Removal process on sample review text

Lemmatization

Lemmatization transforms the words to its root words called “lemmas” that belongs to the language. A lemma (plural lemmas or lemmata) is the canonical form, dictionary form, or citation form of a set of words. We use *WordNetLemmatizer()* from *nltk.stem* to perform lemmatization on our review text.



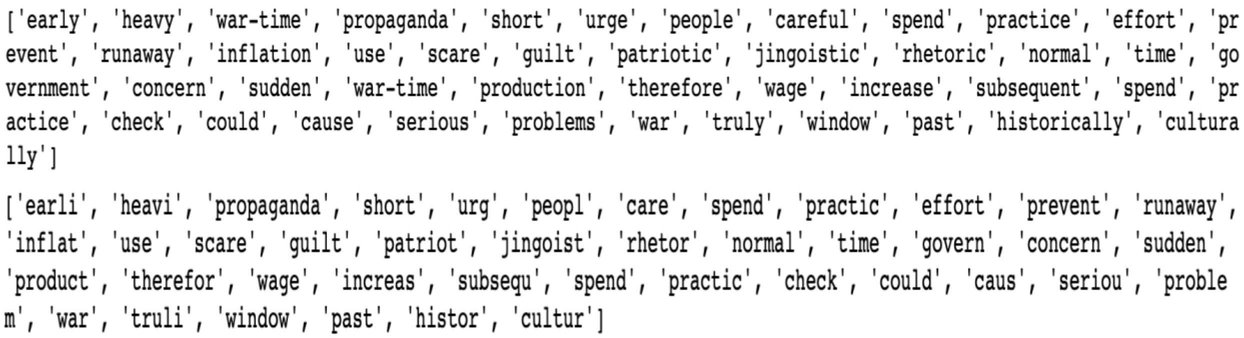
```
['early', 'heavy', 'war-time', 'propaganda', 'short', 'urging', 'people', 'careful', 'spending', 'practices', 'effor
t', 'prevent', 'runaway', 'inflation', 'using', 'scare', 'guilt', 'patriotic', 'jingoistic', 'rhetoric', 'normal', 't
ime', 'government', 'concern', 'sudden', 'war-time', 'production', 'therefore', 'wage', 'increase', 'subsequent', 'sp
ending', 'practices', 'checked', 'could', 'cause', 'serious', 'problems', 'war', 'truly', 'window', 'past', 'historic
ally', 'culturally']

['early', 'heavy', 'war-time', 'propaganda', 'short', 'urge', 'people', 'careful', 'spend', 'practice', 'effort', 'pr
event', 'runaway', 'inflation', 'use', 'scare', 'guilt', 'patriotic', 'jingoistic', 'rhetoric', 'normal', 'time', 'go
vernment', 'concern', 'sudden', 'war-time', 'production', 'therefore', 'wage', 'increase', 'subsequent', 'spend', 'pr
actice', 'check', 'could', 'cause', 'serious', 'problems', 'war', 'truly', 'window', 'past', 'historically', 'cultura
lly']
```

Fig 5: Lemmatization process on sample review text

Stemming

Stemming reduces a word to its root form using suffix stripping, even if the root word is not a valid word in the language. *PorterStemmer()* from *nltk.stem* is used for this. Although *PorterStemmer()* is known for its simplicity and speed. It is commonly useful in Information Retrieval Environments known as IR Environments for fast recall and fetching of search queries. This is not needed for our Classification models hence we skip this process.



```
['early', 'heavy', 'war-time', 'propaganda', 'short', 'urge', 'people', 'careful', 'spend', 'practice', 'effort', 'pr
event', 'runaway', 'inflation', 'use', 'scare', 'guilt', 'patriotic', 'jingoistic', 'rhetoric', 'normal', 'time', 'go
vernment', 'concern', 'sudden', 'war-time', 'production', 'therefore', 'wage', 'increase', 'subsequent', 'spend', 'pr
actice', 'check', 'could', 'cause', 'serious', 'problems', 'war', 'truly', 'window', 'past', 'historically', 'cultura
lly']
['earli', 'heavi', 'propaganda', 'short', 'urg', 'peopl', 'care', 'spend', 'practic', 'effort', 'prevent', 'runaway',
'inflat', 'use', 'scare', 'guilt', 'patriot', 'jingoist', 'rhetor', 'normal', 'time', 'govern', 'concern', 'sudden',
'product', 'therefor', 'wage', 'increas', 'subsequ', 'spend', 'practic', 'check', 'could', 'caus', 'seriou', 'proble
m', 'war', 'truli', 'window', 'past', 'histor', 'cultur']
```

Fig 6: Stemming process on sample review text

Sparse Matrix

From Wikipedia, a sparse array is a matrix in which most of the elements are zero. The number of zero-valued elements divided by the total number of elements (e.g., m × n for an m × n matrix) is called the sparsity of the matrix. A matrix is said to be sparse when its sparsity is greater than 0.5.

Each distinct word from our preprocessed review text is a feature of the classification models, whose algorithms understands numerical features only. Hence, we convert our text features to a sparse matrix of numerical vector representation using CountVectorizer.

CountVectorizer

CountVectorizer() from *sklearn.feature_extraction.text* converts a collection of text documents [review texts in our case] to a sparse matrix of token counts. This implementation produces a sparse representation of the counts using *scipy.sparse.csr_matrix*. Reviews are the observations. If a word exists in a review text then the counter for that word is incremented for that observation.
ngram_range parameter is set to (1,2) in order to use both unigrams and bigrams as features in our models.
min_df parameter is set to 50. When building the vocabulary, terms that have a document frequency strictly lower than 50 are ignored.

Applying this process generates 7957 features/tokens for our 25000 reviews.

TfidfTransformer()

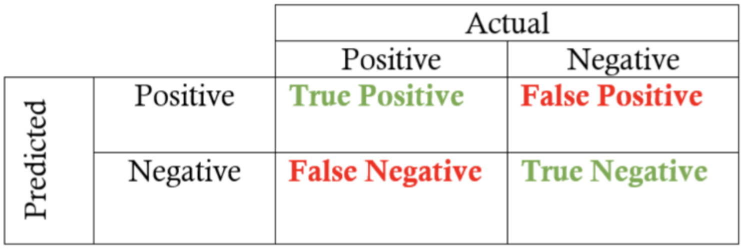
Tf means term-frequency while tf-idf means term-frequency times inverse document-frequency. The more common the word across reviews, the lower its score and the more unique a word is to one particular review the higher the score. The goal of using tf-idf instead of the raw frequencies of occurrence of a token in a given document is to scale down the impact of tokens that occur very frequently in a given corpus and that are hence empirically less informative than features that occur in a small fraction of the training corpus. For instance, words like “film”, “movie” that appears multiple times across all the reviews gets less weightage after TfidfTransformer process.

5. Classification Model Metrics

Confusion Matrix

sklearn.metrics.plot_confusion_matrix

A confusion matrix is a specific table layout that allows visualization of the performance of a classification model. It has two dimensions “Actual” and “Predicted”, each with the number of labels in the model. For our binary classification there are two labels in each dimension. It helps to see if the model is mislabeling the classes.



		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Accuracy

sklearn.metrics.accuracy_score

Accuracy is the proportion of true results among the total number of cases examined. But it becomes a bad metric in case of skewed datasets.

Accuracy = (TP+TN)/(TP+FP+FN+TN)

Precision

sklearn.metrics.precision_score

Out of all the items labeled as positive, how many truly belong to the positive class.

Precision = (TP)/(TP+FP)

Recall

sklearn.metrics.recall_score

Out of all the items that are truly positive, how many were correctly identified as positive.

Recall = (TP)/(TP+FN)

F1 score

sklearn.metrics.f1_score

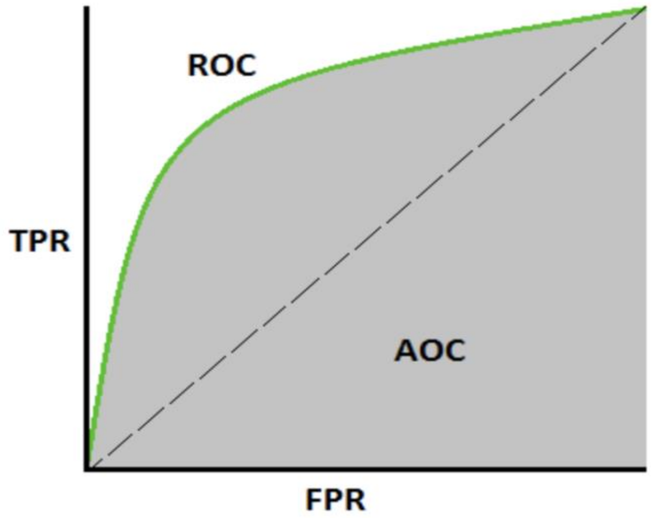
F1 score sort of maintains a balance between the precision and recall for your classifier. If your precision is low, the F1 is low and if the recall is low again your F1 score is low. The F1 score is a number between 0 and 1 and is the harmonic mean of precision and recall.

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

AUC-ROC

sklearn.metrics.roc_auc_score

AUC is the area under the ROC curve. ROC curve is a plot of, **1 – specificity(FPR) vs sensitivity(TPR)**. It tells how much the model is capable of distinguishing between classes.



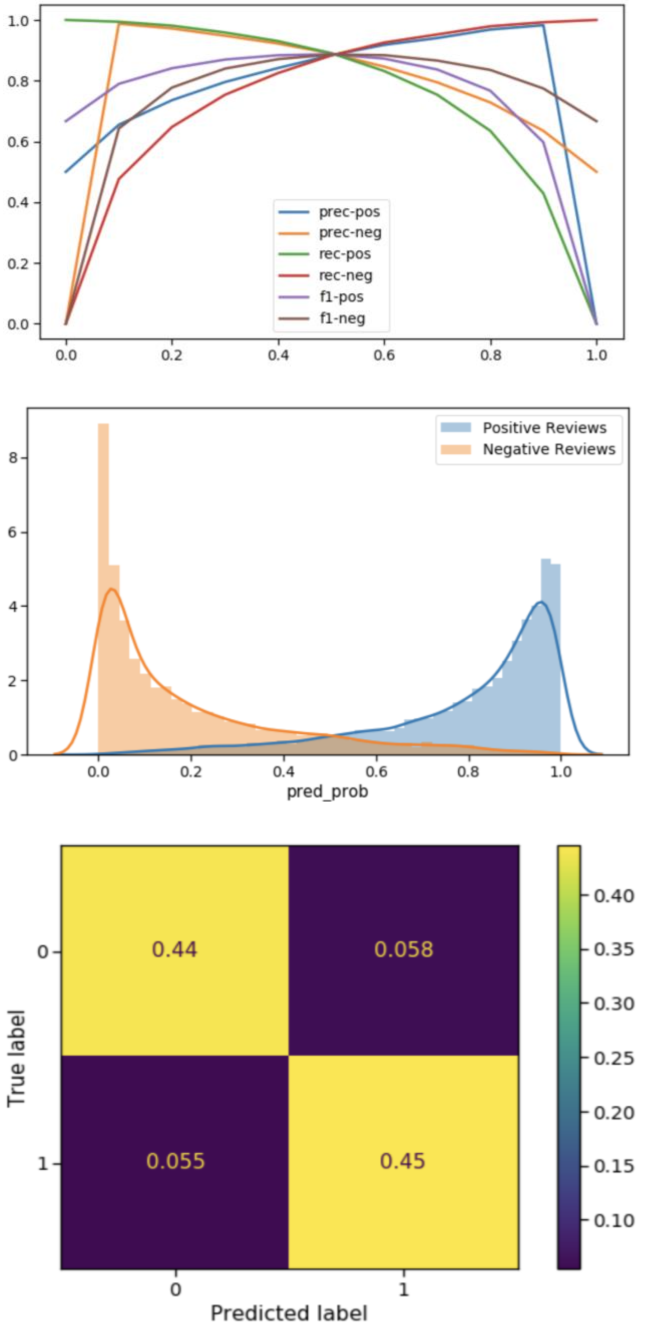
TPR/Sensitivity/Recall = (TP)/(TP+FN)

Specificity = TN/(TN + FP)

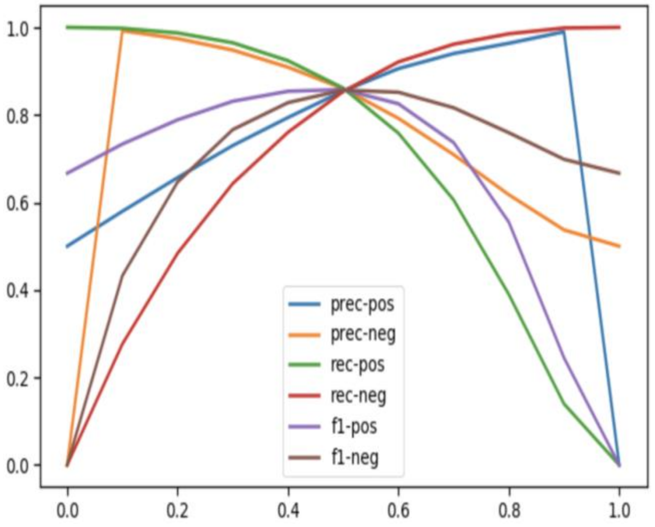
FPR/(1 - Specificity) = FP/(FP + TN)

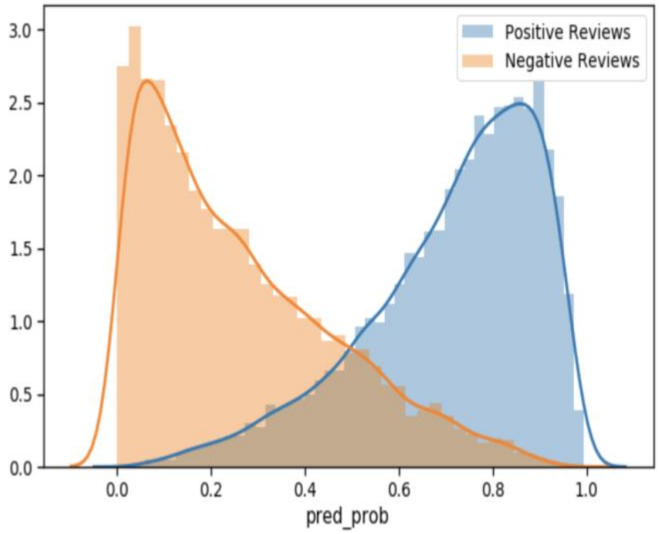
6. Classification Model Comparison

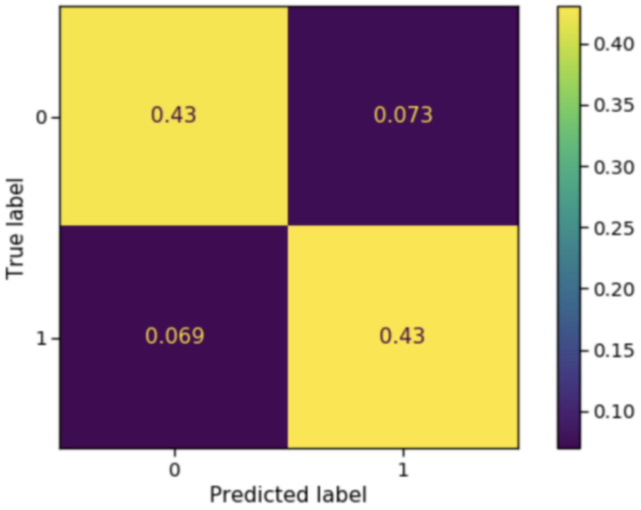
Here is a comparison of 6 classification models: Logistic Regression, Naïve Bayes, SVM, Decision Tree, Random Forest, Light GBM. Plot of *F1 score, recall* and *precision* for each class shows how well the model predicts and plot of *pred_proba* shows how well the model can distinguish between classes. Confusion matrix is also shown.

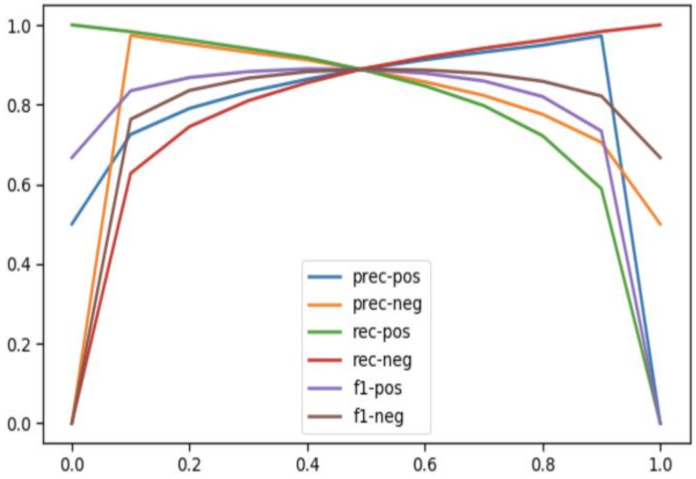
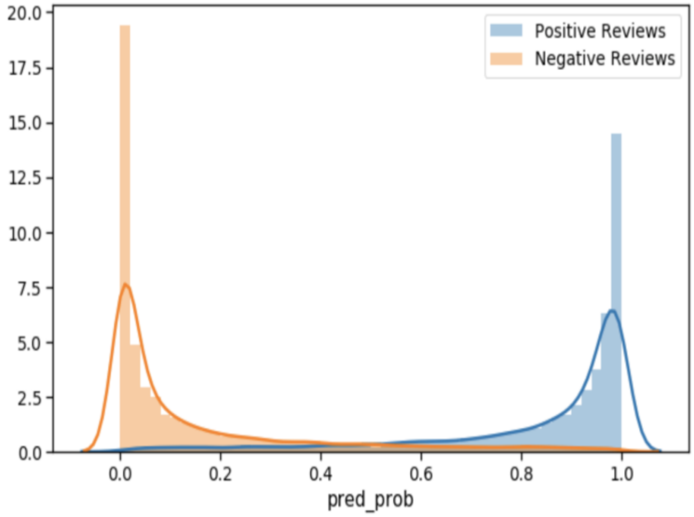
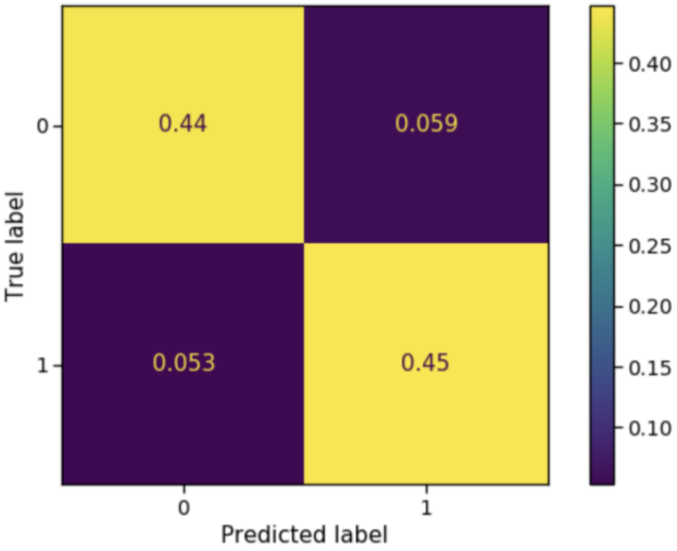
Classification Model	Performance
Logistic Regeression *sklearn.linear_model.LogisticRegression*	

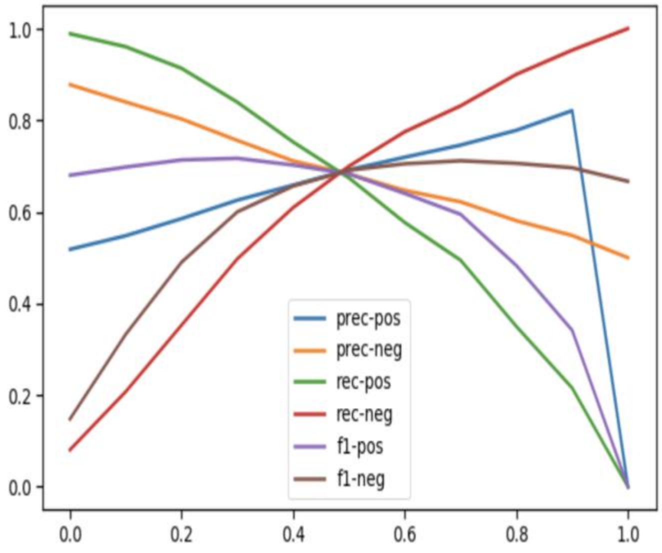
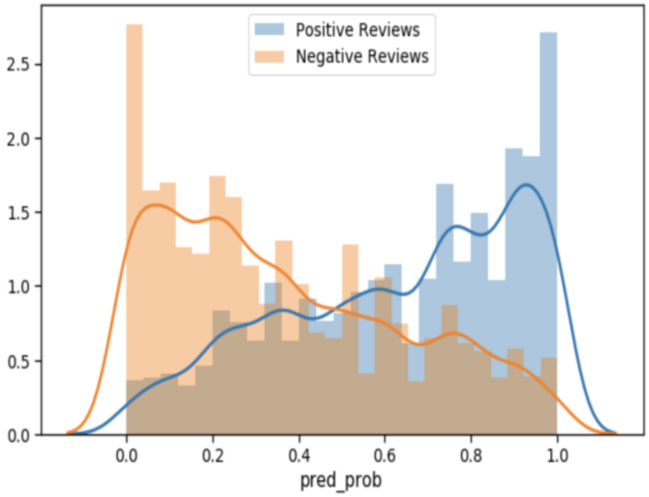
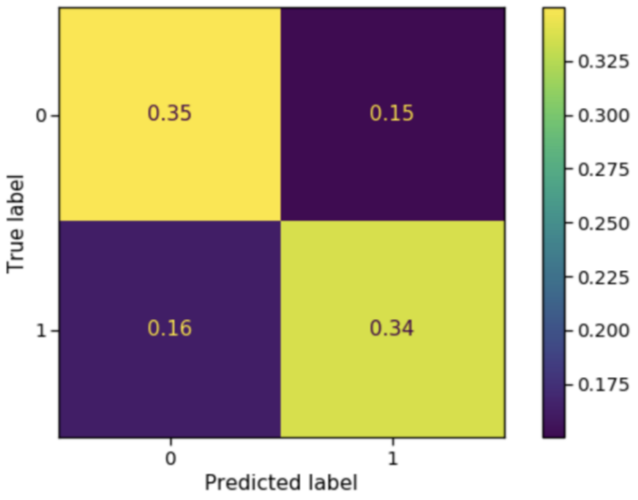
Naïve Bayes
sklearn.naive_bayes.MultinomialNB

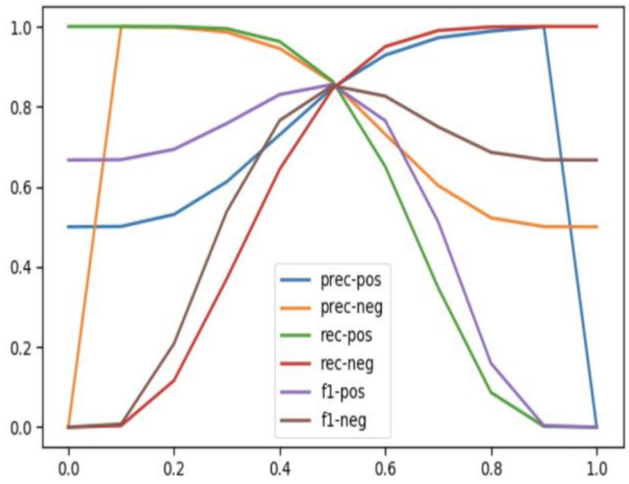
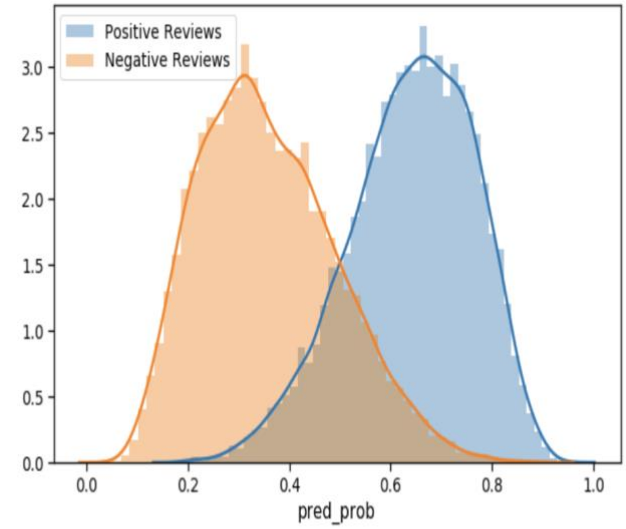
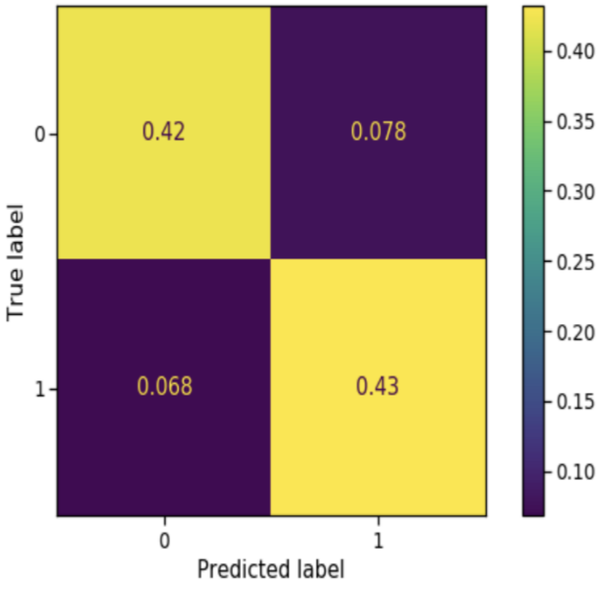




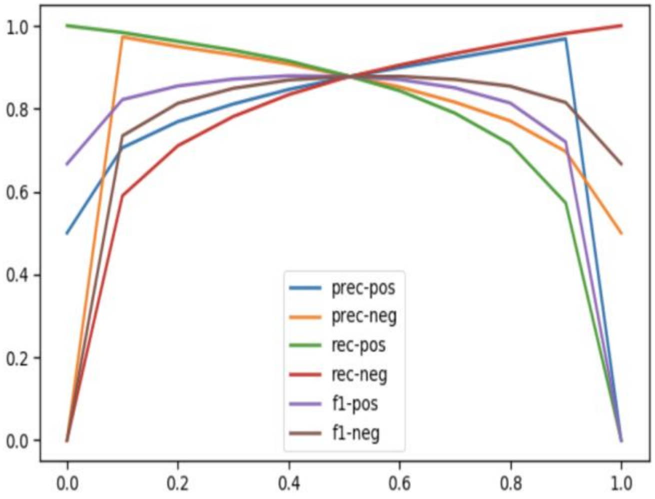


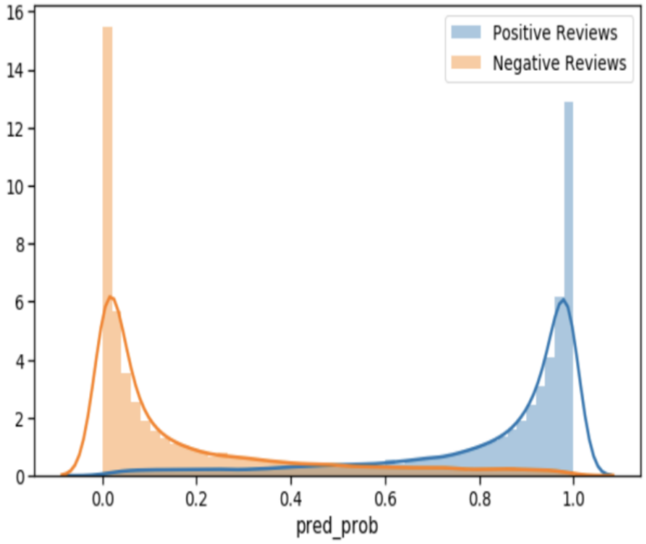
SVM *sklearn.svm.SVC*	  

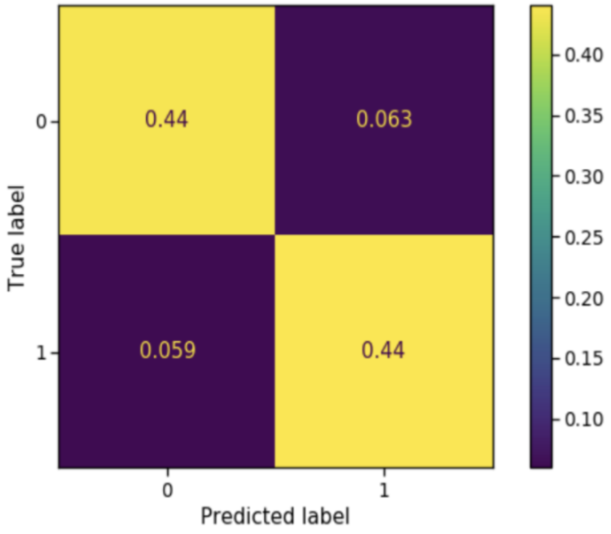
Decision Tree *sklearn.tree.DecisionTreeClassifier*	  

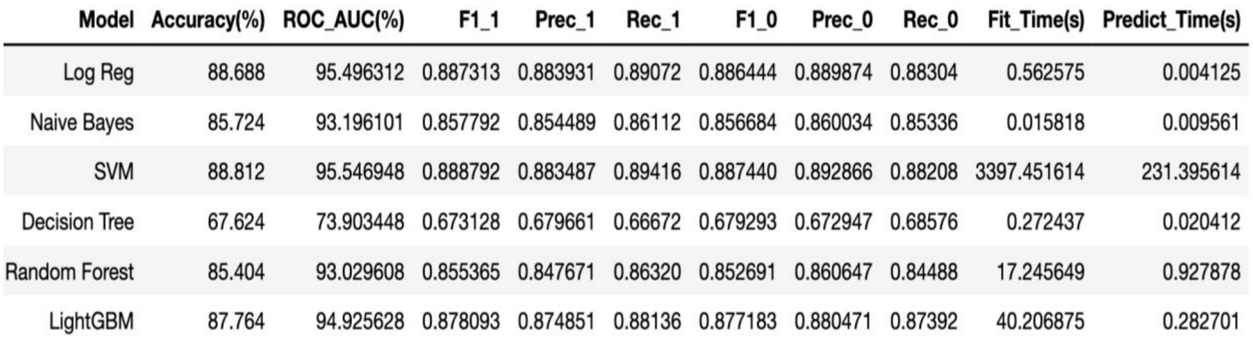
Random Forest *sklearn.ensemble.RandomForestClassifier*	  

Light GBM
lightgbm.LGBMClassifier









Model	Accuracy(%)	ROC_AUC(%)	F1_1	Prec_1	Rec_1	F1_0	Prec_0	Rec_0	Fit_Time(s)	Predict_Time(s)
Log Reg	88.688	95.496312	0.887313	0.883931	0.89072	0.886444	0.889874	0.88304	0.562575	0.004125
Naive Bayes	85.724	93.196101	0.857792	0.854489	0.86112	0.856684	0.860034	0.85336	0.015818	0.009561
SVM	88.812	95.546948	0.888792	0.883487	0.89416	0.887440	0.892866	0.88208	3397.451614	231.395614
Decision Tree	67.624	73.903448	0.673128	0.679661	0.66672	0.679293	0.672947	0.68576	0.272437	0.020412
Random Forest	85.404	93.029608	0.855365	0.847671	0.86320	0.852691	0.860647	0.84488	17.245649	0.927878
LightGBM	87.764	94.925628	0.878093	0.874851	0.88136	0.877183	0.880471	0.87392	40.206875	0.282701

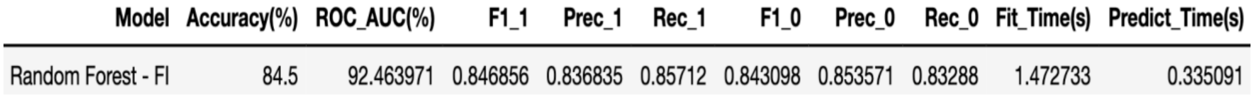
Fig 7: Comparison of different classification model's metrics for IMDB review dataset

7. Feature Selection Techniques

feature_importances_ attribute

*feature_importances_* attribute indicated by machine learning models, helps to remove variables that are not that significant and still have similar or better performance in much shorter training time.

Here we have used this attribute of the Random Forest model we built in the previous section to consider only those features that has a cumulative contribution of 95%. This reduces our number of features from 7957 to 1386 which is almost 86% less features, with comparatively less changes in the model performance. There is actually a significant improvement in the time taken to fit and predict.



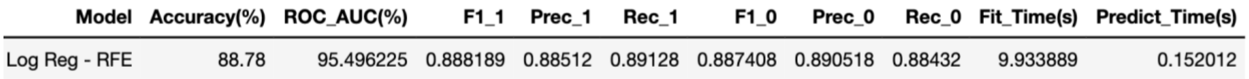
Model	Accuracy(%)	ROC_AUC(%)	F1_1	Prec_1	Rec_1	F1_0	Prec_0	Rec_0	Fit_Time(s)	Predict_Time(s)
Random Forest - FI	84.5	92.463971	0.846856	0.836835	0.85712	0.843098	0.853571	0.83288	1.472733	0.335091

Fig 8: Feature selection using feature_importances_ attribute of Random Forest model

Recursive Feature Elimination (RFE)

RFE works by searching for a subset of features by starting with all features in the training dataset and successfully removing features until the desired number remains. This is achieved by fitting the given machine learning algorithm used in the core of the model, ranking features by importance, discarding the least important features, and re-fitting the model. This process is repeated until a specified number of features remains.

sklearn.feature_selection.RFE is implemented on previously bench marked logistic regression model. The number of features can be reduced from 7957 to 6415.

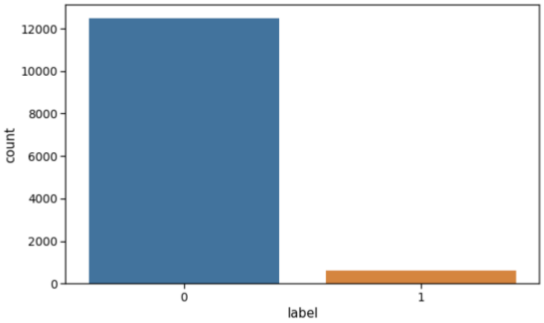


Model	Accuracy(%)	ROC_AUC(%)	F1_1	Prec_1	Rec_1	F1_0	Prec_0	Rec_0	Fit_Time(s)	Predict_Time(s)
Log Reg - RFE	88.78	95.496225	0.888189	0.88512	0.89128	0.887408	0.890518	0.88432	9.933889	0.152012

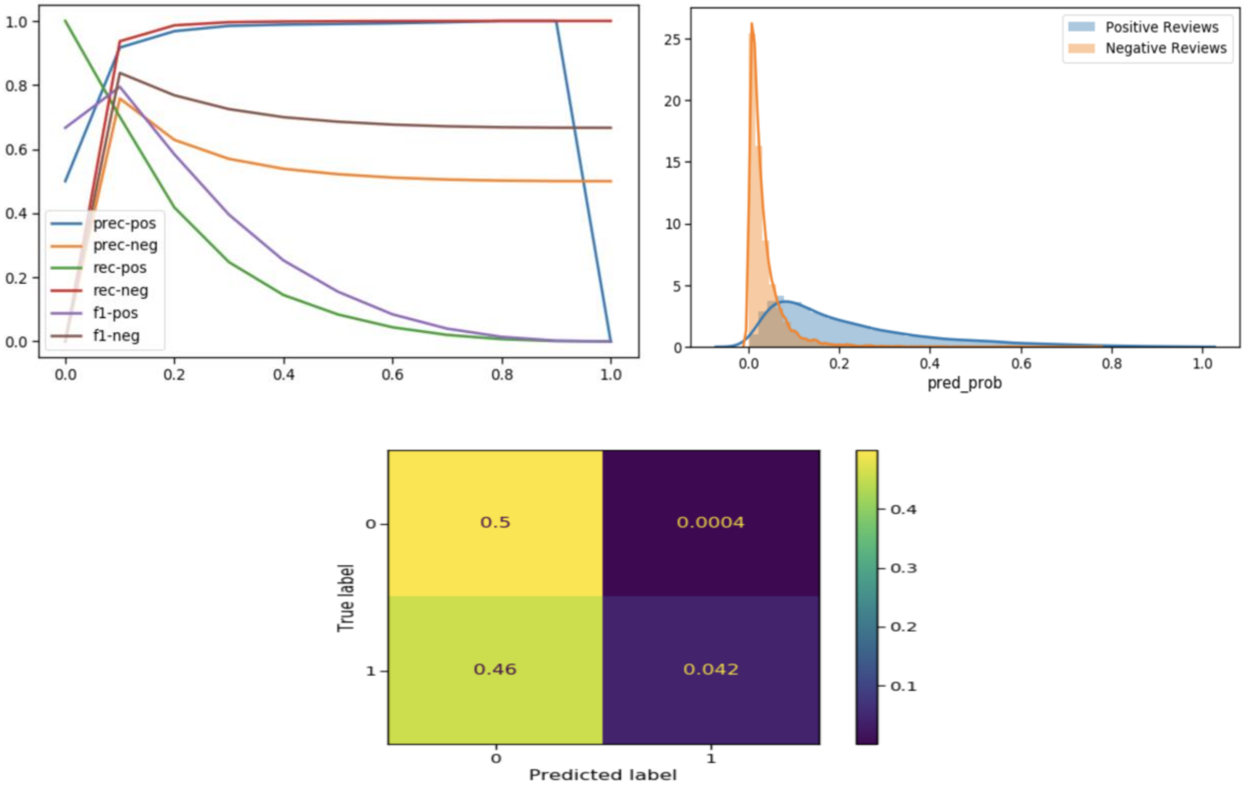
Fig 9: Feature selection using RFE & Logistic Regression model

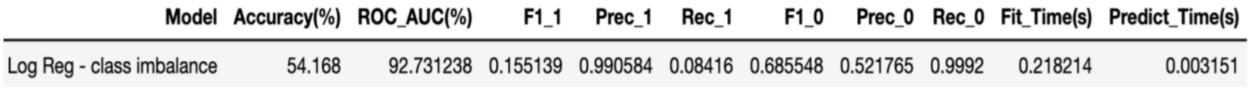
8. Class Imbalance

We have so far dealt with training dataset with equal distribution of labels, 12500 positive and negative reviews, thanks to Stanford AI Lab. But real time data are not distributed evenly and hence needs to be handles differently. We have induced imbalance in our class by randomly sampling 5% of our positive class, resulting in 625 positive reviews and we keep our 12500 negative reviews as it is.



Fitting Logistic Regression model to this dataset with imbalance classes leads to poor performance,



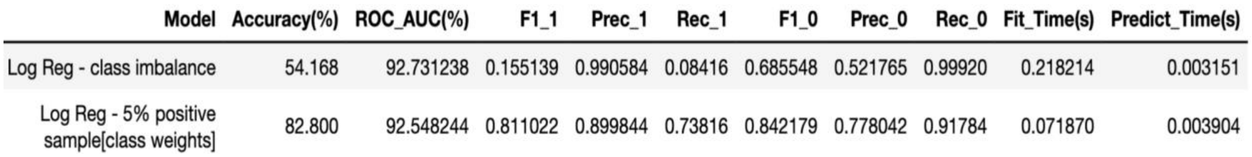


Model	Accuracy(%)	ROC_AUC(%)	F1_1	Prec_1	Rec_1	F1_0	Prec_0	Rec_0	Fit_Time(s)	Predict_Time(s)
Log Reg - class imbalance	54.168	92.731238	0.155139	0.990584	0.08416	0.685548	0.521765	0.9992	0.218214	0.003151

Fig 9: Poor performance of Logistic Regression model due to class imbalance

class_weight parameter

class_weight parameter of sklearn's Logistic Regression function penalizes mistakes in samples of a class with its assigned weight. So higher the class weight means more emphasis on that class. Setting this parameter to {0:.05, 1:.95} balances our dataset and improves the performance drastically.

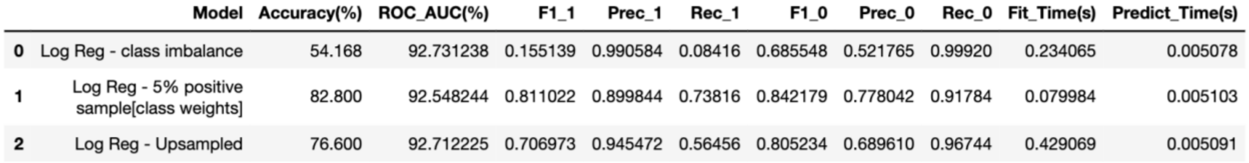


Model	Accuracy(%)	ROC_AUC(%)	F1_1	Prec_1	Rec_1	F1_0	Prec_0	Rec_0	Fit_Time(s)	Predict_Time(s)
Log Reg - class imbalance	54.168	92.731238	0.155139	0.990584	0.08416	0.685548	0.521765	0.99920	0.218214	0.003151
Log Reg - 5% positive sample[class weights]	82.800	92.548244	0.811022	0.899844	0.73816	0.842179	0.778042	0.91784	0.071870	0.003904

Resampling

Upsampling

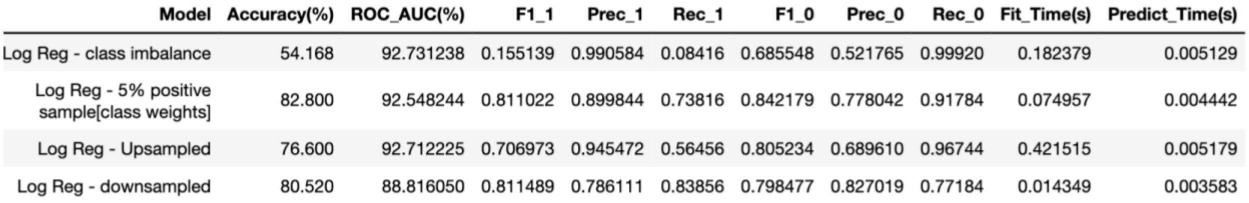
In this method the 625 positive samples are randomly sampled repeatedly to get 12500 samples.



	Model	Accuracy(%)	ROC_AUC(%)	F1_1	Prec_1	Rec_1	F1_0	Prec_0	Rec_0	Fit_Time(s)	Predict_Time(s)
0	Log Reg - class imbalance	54.168	92.731238	0.155139	0.990584	0.08416	0.685548	0.521765	0.99920	0.234065	0.005078
1	Log Reg - 5% positive sample[class weights]	82.800	92.548244	0.811022	0.899844	0.73816	0.842179	0.778042	0.91784	0.079984	0.005103
2	Log Reg - Upsampled	76.600	92.712225	0.706973	0.945472	0.56456	0.805234	0.689610	0.96744	0.429069	0.005091

Downsampling

Majority class is sampled randomly to match the number of samples in minority class. Observations are eliminated in the majority class.



Model	Accuracy(%)	ROC_AUC(%)	F1_1	Prec_1	Rec_1	F1_0	Prec_0	Rec_0	Fit_Time(s)	Predict_Time(s)
Log Reg - class imbalance	54.168	92.731238	0.155139	0.990584	0.08416	0.685548	0.521765	0.99920	0.182379	0.005129
Log Reg - 5% positive sample[class weights]	82.800	92.548244	0.811022	0.899844	0.73816	0.842179	0.778042	0.91784	0.074957	0.004442
Log Reg - Upsampled	76.600	92.712225	0.706973	0.945472	0.56456	0.805234	0.689610	0.96744	0.421515	0.005179
Log Reg - downsampled	80.520	88.816050	0.811489	0.786111	0.83856	0.798477	0.827019	0.77184	0.014349	0.003583

9. Citations

- Maas, Andrew L. and Daly, Raymond E. and Pham, Peter T. and Huang, Dan and Ng, Andrew Y. and Potts, Christopher - Learning Word Vectors for Sentiment Analysis
- Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

10. References

- https://www.datacamp.com/community/tutorials/stemming-lemmatization-python
- https://kavita-ganesan.com/how-to-use-countvectorizer/#.XwD3D55KiJk