2020

University of Alaska
Fairbanks

Sree Nirmillo Biswash
Tushar

Sentiment Analysis in amazon movie review by naive bayes and random forest classifier

CS 480- Machine Learning Final Project
Sree Nirmillo Biswas Tushar **Date: April 24, 2020**

1. Introduction:

Text sentiment analysis through a natural language processing has many applications including reviewing social media status, customer satisfaction of a product etc. One of those applications involves predicting either a movie has made a positive and negative impact of people's mind without reading the whole review. *Hadi et al.* has argued how to develop movie review classifier (both binary and multi-scale) using existing machine learning algorithm in [1]. Another paper was reviewed on short massage sentiment analysis based on BOW (bag of word) model and random forest classifier [2]. The main objective of this project is to review naive bayes and random forest classifier using BOW model and BOW+Wod2Vec model and this whole work is very much inspired by these two works [1-2].

2. Working Procedure and Results:

2.1 Dataset:

The dataset I have used for this project has been taken from Kaggel [3]. It has separate training and testing dataset and there are no missing values there. For every instance, there are two columns- "Text" and "Sentiment". The dataset has equal number of positive and negative reviews which is shown in figure (1).

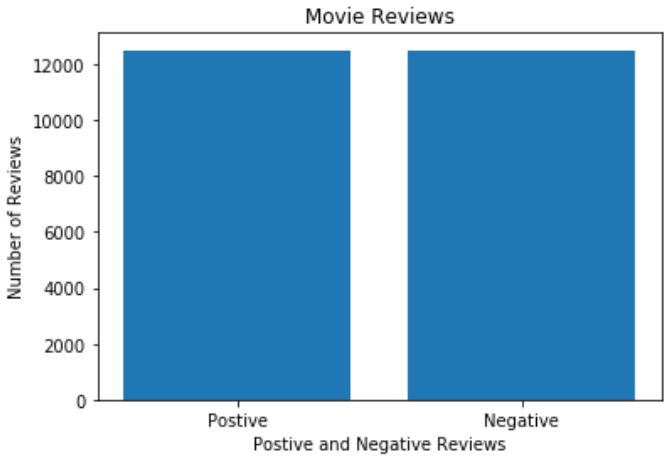

Figure 1: Movie Review for the entire movie review dataset.

But, the testing dataset was not labeled. Therefore, I split the training dataset into training dataset and testing dataset. In splitting the training data, I have divided the dataset into 50% training and 50% testing data. I have divided the dataset in this fashion since in [1], *Hadi et al.* have tried in the similar way and found good result. For splitting the dataset, I have used "train_test_split" from sklearn.model_selection which is a standard method for even distribution of the dataset. But the dataset needs to be made ready for training and therefore, I have cleaned the dataset in a number of stages as mentioned in [1-2].

2.2 Cleaning

The first stage cleaning was to remove unnecessary punctuations. I have uploaded the punctuations from string module and put nothing in the place of string using "translate" function. Thereby, I have removed all the punctuations. Then, the words are made lowercase so that same word starting with capital letter and small letter are considered same. In the next stage, I have removed all the words which are unnecessary in making decision about sentiment in a text. In my project, I have considered two types of stop-words removal. Firstly, I have removed the stop-words defined by natural language processing toolbox (nltk toolbox) [4]. There are total 179 stop-words in nltk and first 10 stop-words are given below. Secondly, it is also easily realizable which parts of speech do not polarize a sentence (give a sentence positive or negative connotation). Therefore, in this stage I have removed those words which carry no sentiment at all based on their parts of speech. In order to do that, I have figured out the parts of speech of all words using nltk.pos_tag (another natural language processing toolbox) and then remove words based on the parts of speech. For training, I have just taken adjective, verb, modal verb, preposition/subordinating conjunction, coordinating conjunction and predeterminer. Figure 5 illustrates the words left after each cleaning stage. In the figure, it is shown that parts of speech-based stop-words removal has reduced the word size by almost 85% and then ntlk toolbox stop-words removal has been also decreased the word size by another 10%. Therefore, cleaning seems very effective and helpful to reduce the feature size (discussed in next section).

The list of the parts of speech tag from nltk position tagger is given in table-1 [4]:

The removal of the unnecessary words is important for two reasons:

1. It makes the decision tree more robust
2. It reduces computation time.

In the third stage of cleaning, I have taken word root of every word so that words having same word root is considered as a single word. For an example, 'word' and 'words' are considered same. A well know word root finder is "ps.stem" (another natural language processing tool) which has been used for finding out the word root. Lastly, I have listed all the unique words in the training set in order to build the feature vector. But, the unique words size is too large to create the feature vector. For this reason, I have arranged all the unique words in terms of their word frequency in a descending order (highest frequent word first, second most frequent word next etc.) and from there selected n number of unique words a feature size. In the text, wherever, it is written that the feature size is 1000, it means that 1000 highest frequency words were taken to create the feature vector.

Tag	Parts of speech	Tag	Parts of speech
CC	coordinating conjunction	PRP$	possessive pronoun my, his, hers
CD	cardinal digit	RB	RB adverb very, silently,
DT	determiner	RBR	adverb, comparative better
EX	existential there (like: "there is" ... think of it like "there exists")	RBS	adverb, superlative best
FW	foreign word	RP	particle give up
IN	preposition/subordinating conjunction	TO	to go 'to' the store.
JJ	adjective 'big'	UH	interjection errrrrrrrm
JJR	adjective, comparative 'bigger'	VB	verb, base form take
JJS	adjective, superlative 'biggest'	VBD	verb, past tense took
LS	list marker 1)	VBG	verb, gerund/present participle taking
MD	modal could, will	VBN	verb, past participle taken
NN	noun, singular 'desk'	VBP	verb, sing. present, non-3d take
NNS	noun plural 'desks'	VBZ	verb, 3rd person sing. present takes
NNP	proper noun, singular 'Harrison'	WDT	wh-determiner which
NNPS	proper noun, plural 'Americans'	WP	wh-pronoun who, what
PDT	predeterminer 'all the kids'	WP$	possessive wh-pronoun whose
POS	possessive ending parent's	WRB	wh-abverb where, when
PRP	personal pronoun I, he, she		

Table-1: List of all position tagger

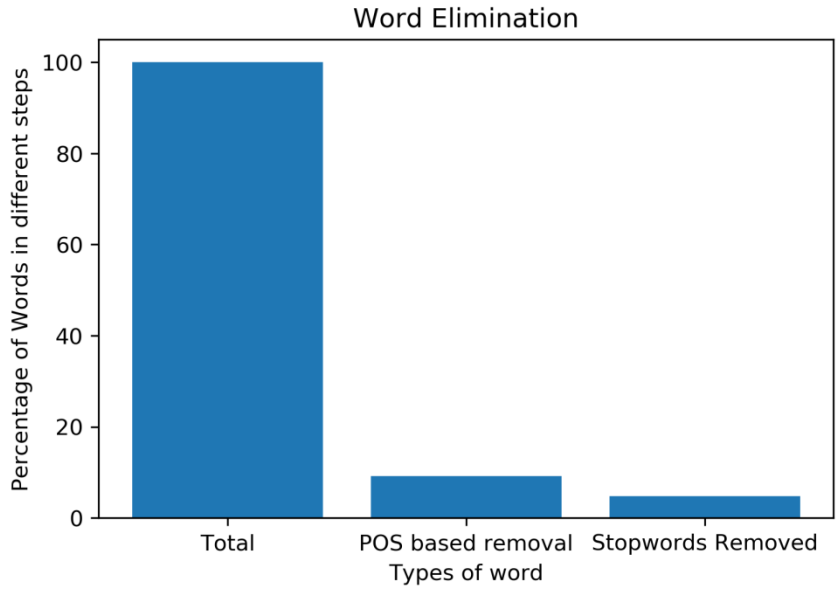

Figure 2: Elimination of words in different stage (Total words, the remaining words after removing words of unnecessary parts of speech, the remaining words after removing stopwords defined by nltk)

2.3 Feature Vector Creation:

For creating the Feature vector, I have bag of words model. For the bag of words mode, I have calculated how many time i^{th} feature word has appeared in j^{th} comment. Here is an example of a feature vector for jth instance:

$$N_{1j}, N_{2j}, \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots, N_{ij}, \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots, N_{nj}$$

Where N_{ij} is the number of times i^{th} word has appeared in j^{th} review.

It is also important that the feature size should cover a large portion of a review. But, it is not recommended to take a large feature size for computational complexity. Therefore, a trade-off is required between the size of the feature size and the review coverage. Figure 3 shows how many words in the total training sets are covered when we vary the feature size from 1000 words to 30000 words. It is very clear that, the slope started to decrease after 5000 words and it is our trade of point. It can be added that in make a further increase not much word is considered in the decision making.

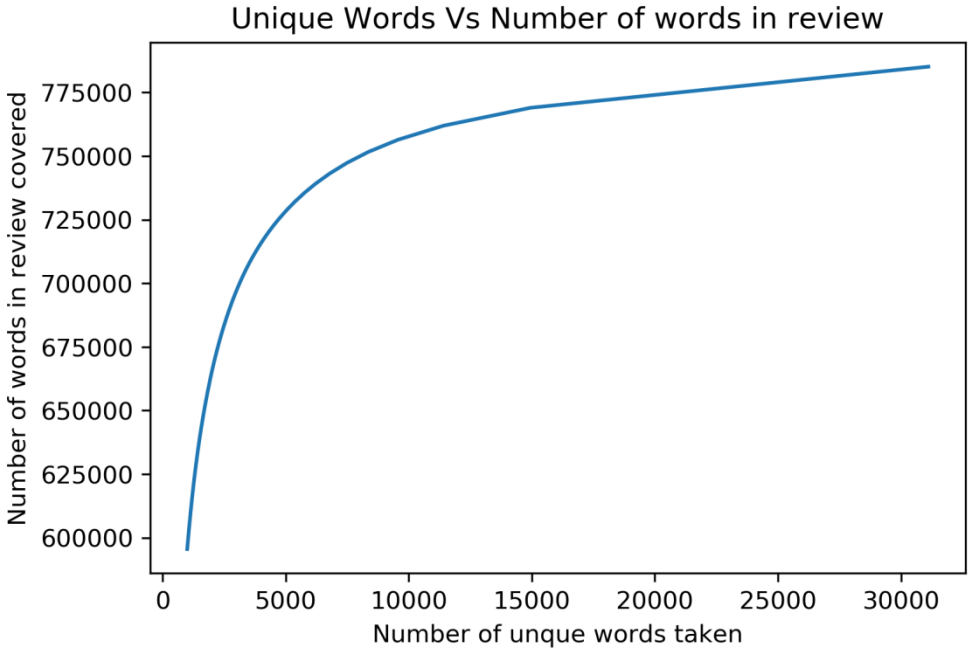

Figure 3: Number of words in the review covered by the feature size chosen

For the feature size of 5000 and for the first 5 training instances, the feature vectors appear like below:

	0	1	2	3	4	5	6	7	8	9	...	4990	4991	4992	4993	4994	4995	4996	4997	4998	4999
0	1	0	1	1	0	0	0	2	0	0	...	0	0	0	0	0	0	0	1	0	0
1	1	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	1	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	3	0	2	28	6	1	1	0	1	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

Column here is the feature (so total 5000 feature) and row is the review (here are 5 reviews). For every review, the algorithm counts how many times each word appears. For an example, for the first training instance (1st row), 1st word appears once, second word does not appear and 8th word appears twice. Now, our feature vector is ready to fit a classifier. I have used two classifier model for classifying the sentiment: Multinomial Naive Bayes and Random Forest Classifier

2.4 Classifications and results:

2.4.1. BOW + Naive Bayes Classifier:

Multinomial Naïve Bayes classifier is a very suitable classifier for discrete data like text. It is nothing but the discrete naive bayes classifier where every review is classified (positive/negative) by the likelihood of every word in the review times the prior (equation-1).. If the predictors in the naïve bayes classifier are not independent, then it gives bad result. But it is computation friendly and easy to use. Therefore, I have used it in this project.

$$\mathrm{P}(\mathrm{Reviw1} = \mathrm{positive}) = \mathrm{P}(\mathrm{sentiment} = \mathrm{postive}) \times \mathrm{P}(\mathrm{word1}|\mathrm{sentiment} = \mathrm{positive}) \times \dots \dots \dots \mathrm{P}(\mathrm{wordk}|\mathrm{sentiment} = \mathrm{positive}) \times \dots \dots \dots \times \quad (1)$$

The result of the naive bayes classifier is shown below in table 2 when feature size started to decrease

Case	Feature Size	Training Result	Testing Result
Test 0	5000	84.43%	82.47%
Test 1	4000	84.36%	82.44%
Test 2	2000	83.75%	82.48%
Test 3	1000	83.29%	82.79%
Test 4	500	82.16%	82.06%
Test 5	300	81.34%	81.34%
Test 6	200	79.16%	78.06%
Test 7	100	74.60%	73.78%

Table 2: Accuracy in prediction of the sentiment by bag of word model (BOW)+ naive bayes classifier in training and testing dataset for feature size 100, 200, 300, 500, 1000, 2000, 4000 and 5000.

The result can be better explained from figure 4. When feature size starts to increase (although testing result starts getting better), but the data starts to overfit. This really makes sense since larger feature size will start gaining unnecessary details about the data. For, feature size 500, training and testing accuracy is equal and therefore it is recommend for the naive bayes classifier model.

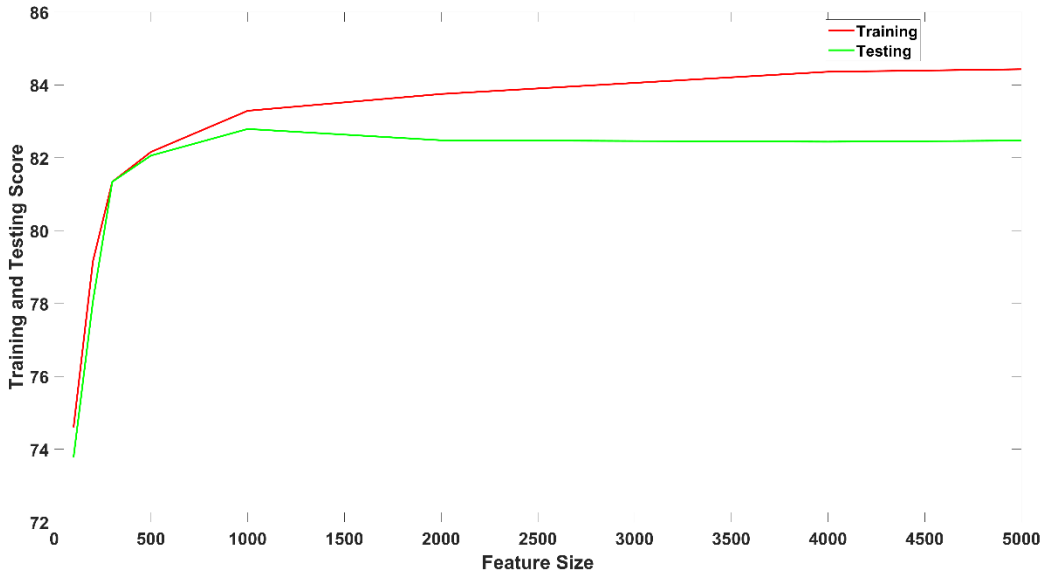

Figure 4: Training Vs Testing result while feature size is varied

The confusion matrix is given below for feature size 500, 1000 and 2000. In the confusion matrix, first row is for positive review and second column is for negative review. When the feature size starts to increase, false negative value starts to decrease (FN) whereas the false positive value (FP) starts to increase.

Feature Size 500:

Training: $\begin{bmatrix} 5088 & 1202 \\ 1028 & 5182 \end{bmatrix}$ Testing: $\begin{bmatrix} 4995 & 1215 \\ 1028 & 5262 \end{bmatrix}$

Feature Size 1000:

Training: $\begin{bmatrix} 5242 & 1048 \\ 1040 & 5170 \end{bmatrix}$ Testing: $\begin{bmatrix} 5124 & 1086 \\ 1065 & 5225 \end{bmatrix}$

Feature Size 2000:

Training: $\begin{bmatrix} 5371 & 973 \\ 1058 & 5152 \end{bmatrix}$ Testing: $\begin{bmatrix} 5126 & 1084 \\ 1106 & 5184 \end{bmatrix}$

2.4.2 BOW + Random Forest Classifier:

The main parameter to tune in the random forest classifier is the number of trees in the forest and depth of the tree. The number of trees in the forest is defined in the classifier as the n_estimator. The parameter "min_samples_split" means how many samples are required to split a branch node. The depth of the tree is defined by max_depth. It can be defined as an integer or can be kept as none (none means nodes keep splitting until all leaves are pure or until all leaves contain less than min_samples_split samples). In order to reduce the bias in the dataset, instead of using the whole dataset as once, I have used bootstrapping so that the whole dataset is divided into some subsets and

used in different tree model. The use of n different decision trees and bootstrapping decreases biasness in the dataset since decision about a new testing instance is made based on n different models and n different datasets. For an instance, if n_estimator=10, then even if three models are biased, other seven models will give correct result and the decision will be made by maximum voting. I did not impose any condition on the maximum leaf node so that it can grow as long as it reaches a pure node. The condition for splitting the node (measures of the impurity) is chosen as "gini". The feature size is very large and therefore, I have taken maximum feature size as the square root of the original size of the feature. Other parameters are kept as default value.

I had started this project with random forest as a classifier where feature size was 5000. Although feature size is high but the result gives some intuition about the over fitting. The results are Table -3 shows the training and testing accuracy for different test size and maximum depth. Although 80% testing set does not make sense, I have done so to show that the test size variation has no significant impact in the sentiment prediction. But, maximum depth of the tree helps to control the overfitting. It is also apparent from table-3 is that minimum sample split has some effect on the result.

Then, I have tuned 3 important parameters mainly in the decision tree (number of decision trees, maximum depth of the trees and minimum sample split). It is apparent from the table that the increase in the depth of the tree improves the result. Although it is normally suggested to keep the maximum depth of the tree constant, I have found that it has an impact in reducing over fitting. The reason could be that the optimum depth of the tree prevents gaining unnecessary information for making decision and thereby, reducing the over fitting.

Then, I was interested to see how maximum depth effects the training and testing set accuracy (depth=1 is just for showing the result). When the depth of the tree increases (figure-5), sentiment prediction accuracy on the training and testing dataset increases at first but after a certain point training accuracy rapidly increases whereas testing accuracy flattens. Therefore, maximum depth increment results in overfitting. Therefore, it is important to get the optimum point where testing accuracy saturates since a larger decision tree results in large computational cost.

Test size	Maximum depth	min_samples_split	Testing result	Training result
.20	None	2	0.9846	0.7652
.50	None	2	0.98496	0.7462
.80	None	2	0.9928	0.71945
.30	5	2	0.758	0.739
.50	5	4	0.7574	0.7041

Table 3: Accuracy in prediction of the sentiment in training and testing dataset by BOW+ random forest classifier for feature size 5000, max_features: sqrt, criterion: gini, max_depth:None, min_samples_split=2, min_samples_leaf=1, max_leaf_nodes=None; min_impurity_decrease=0; min_impurity_split=1e-7;

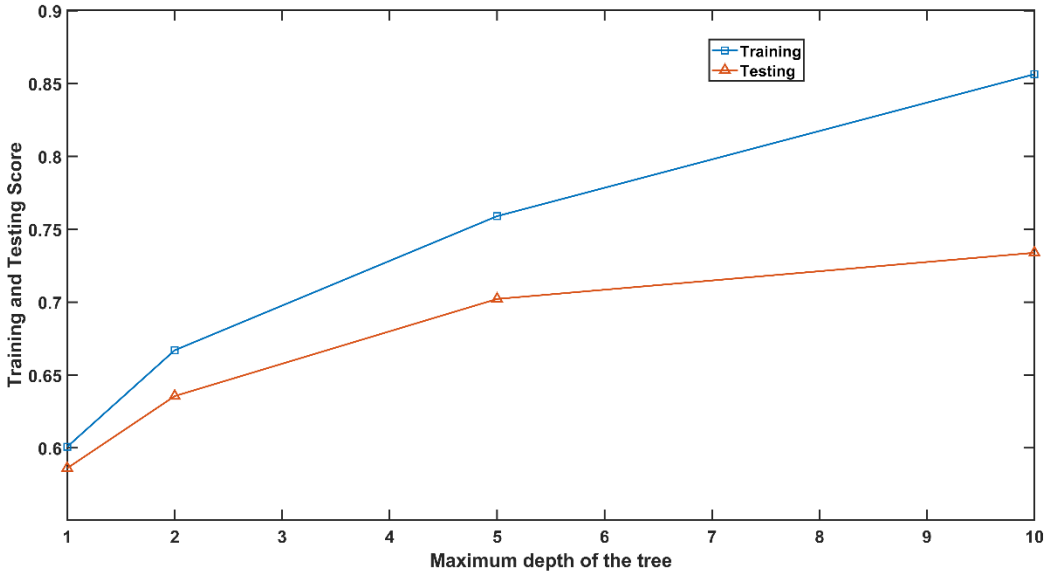

Figure 5: Training Vs Testing result while maximum depth of the tree is varied (Feature=5000, test size=0.50, min_samples_split=2, min_samples_leaf=1, number of decision trees=10)

After gaining this knowledge about the parameter impact on this dataset, I started to tune the depth of the tree, number of decision tree, minimum sample to split a node and minimum sample at the leaf node around the point where I have found the best result. For this purpose, a smaller feature size of 500 is taken since it reduces computational time and provides good results at the same time (apparent from naive bayes classifier result). In the first two cases (first two rows of table 4), I have only changed the depth of the tree. If the depth of the tree is increased, the overfitting increases but at the same time model gets better in predicting unknown testing dataset. I have chosen the depth of the tree as 15 which is basically a trade-off. Then, in the next three cases, I have increased the size of the tree by 50 in order to reduce the bias. Larger tree size increases the overfitting of the data and computational complexity but improves the prediction accuracy. The parameter "min_sample_split" controls the growth of the tree. In the next step, the min_sample_split is augmented. But, unfortunately, it makes the testing dataset prediction even worse. Lastly, I have increased the number of minimum samples at the leaf node which also does not appear effective.

Feature Size	Min_sample_split	Min_samples leaf	N_estimators	Max Depth	Training Score	Testing Score
500	2	1	30	10	83.44%	78.70%
500	2	1	30	15	89.53%	78.83%
500	2	1	50	15	89.52%	79.10%
500	2	1	100	15	89.99%	79.52%
500	2	1	150	15	90.18%	80%
500	5	1	150	15	89.54%	80%
500	10	1	150	15	88.78%	79.78%
500	5	5	150	15	87.14%	79.78%

Table 4: Accuracy in prediction of the sentiment in training and testing dataset BOW+ random forest classifier for feature size 500

I continued this quest for finding optimum parameter for feature size of 1000, 2000, 4000 (table 5). The results show that when we increase the feature size, the accuracy in predicting a sentiment improves. It also shows that the tuning the max_dept of the tree does not improve the result (row 3,4 and 5) after the optimum point but increasing the decision tree does improve the score a little bit. Although the increment in the decision tree increases the result a little bit, but it adds high computational cost. After all those trails, the best prediction result for testing dataset is found to be 82.08% for feature size= 4000, minimum sample split=5, minimum sample leaf=1, the number of decision tress=150, maximum depth of the tree=15.

Feature Size	Min_sample_split	Min_samples leaf	N_estimators	Max Depth	Training Score	Testing Score
1000	5	1	100	15	90.33%	80.72%
1000	5	1	150	15	90.63%	80.91%
1000	5	1	150	13	88.74%	80.4%
1000	5	1	150	17	92.28%	80.63%
1000	10	1	150	15	89.76%	80.58%
2000	5	1	150	15	91.99%	81.33%
2000	5	1	150	13	90.36%	81.35%
4000	5	1	150	13	91.27%	81.82%
4000	5	1	150	15	93.11%	81.92%
4000	5	1	150	20	96.54%	82.08%

Table 5: Accuracy in prediction of the sentiment in training and testing dataset for BOW+ random forest classifier feature size 1000, 2000 and 4000

The confusion matrix for the BOW+random forest classifier is given below for the best cases of feature size 500, 1000 and 2000. The error is much large for predicting false when actually it is true than FP.

Feature Size 500, n_estimator=150, maximum depth=15, minimum sample split=5:

Training: $\begin{bmatrix} 5352 & 938 \\ 369 & 5841 \end{bmatrix}$ Testing: $\begin{bmatrix} 4617 & 1593 \\ 932 & 5358 \end{bmatrix}$

Feature Size 1000, n_estimator=150, maximum depth=13, minimum sample split=5:

Training: $\begin{bmatrix} 5474 & 816 \\ 355 & 5855 \end{bmatrix}$ Testing: $\begin{bmatrix} 4709 & 1501 \\ 949 & 5341 \end{bmatrix}$

BOW+Word2Vec Model:

Word2Vec is another method of creating feature vector. The problem with bag of words that it does not consider the context of the word (which words are related to which words). Therefore, one efficient method of creating feature vector is to create a word vector of neighboring words with main word in the right.

The input to the word 2 vectors is sentence where every sentence is tokenized into words. For an instance, for a movie review "the movie is good. It is fun.", the input to the word2vec model will be [["the", "movie", "is", "good"], ["It", "is", "fun"]]. For creating word2vec, I have used word2vec tools from gensim [6]. The main parameters of the word2vec model are size word vector, the maximum distance between the word and the neighboring word (window size), minimum word count in the review needed to be considered for the word2vec model(min_count) [5]. I have set min_count=2, window size=5 and keep everything as default. In our dataset, there are total 249877 sentences which are cleaned, tokenized and then given as the input for the word2vec. Initially, feature size is chosen as 1000. I have used word to vector embedding called averaging since in [1], author claimed that this method overperforms the complex clustering method. In Word2Vec averaging method, for every feature word its neighbor also gets a unit increase in the feature vector if the neighboring word contains in the feature.

The algorithm that I came up with after understanding how it works is described below in a nutshell:

(1) Create a list like this:
$[[[N1, ID1], [N2, ID2] \dots [Nm, IDm], [Word1, IDw1]], \dots\dots\dots, [[N1, ID1], [N2, ID2] \dots [Nm, IDm], [Wordn, IDwn]]]$
(2) Find the word frequency of every Feature word, Word[i] for every review, Review[j].
(3) Find out whether Word[i] has appeared in comment[j].
(4) For every appearance set Word[IDw[i]]+=1, Word[ID[i]]+1 for every neighbors.
(5) Repeat this loop for every review.
(6) Average (normalize) every Review[j] by its total count.

After creating the feature vector, I have tested this algorithm by random forest classifier. For 50% testing data, feature size=1000, decision tree size=150, max_depth=15, min_sample_split=5, the training accuracy score is found to be 86.25% and testing accuracy is 78.04%. I did not run simulation for other parameters because of time constraints.

3. Conclusion:

I have processed training set and testing set separately so testing set remains independent of the training set. It is very difficult to make a conclusion based on this result since I have done such small number of iterations because of the time constraints. From the result, it is evident that naïve bayes classifier performs better than random forest classifier. This is probably because my feature selection technique was not right. If I would select principal component analysis technique then probably, random forest classifier would perform better. In my whole parameter tuning of random forest classifier only maximum depth of the tree shown significant influence over the result. It is obvious since the model was facing overfitting. Most of the time of this project has been spent in making the algorithm and therefore in future I would spend more time on it to improve the model by tuning parameters in a broad range.

4. References:

[1] Pouransari, Hadi, and Saman Ghili. "Deep learning for sentiment analysis of movie reviews." *Technical report, Stanford University, Tech. Rep.* (2014).

[2] Carmo, Rodrigo & Lacerda, Anísio & Dalip, Daniel. (2017)," A Majority Voting Approach for Sentiment Analysis in Short Texts using Topic Models", 449-455. 10.1145/3126858.3126861.

[3]https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews#IMDB%20Dataset.csv

[3] https://www.nltk.org/

[4] https://pythonprogramming.net/natural-language-toolkit-nltk-part-speech-tagging/

[5] https://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.XqLOVchKjIU

[6] https://github.com/RaRe-Technologies/gensim