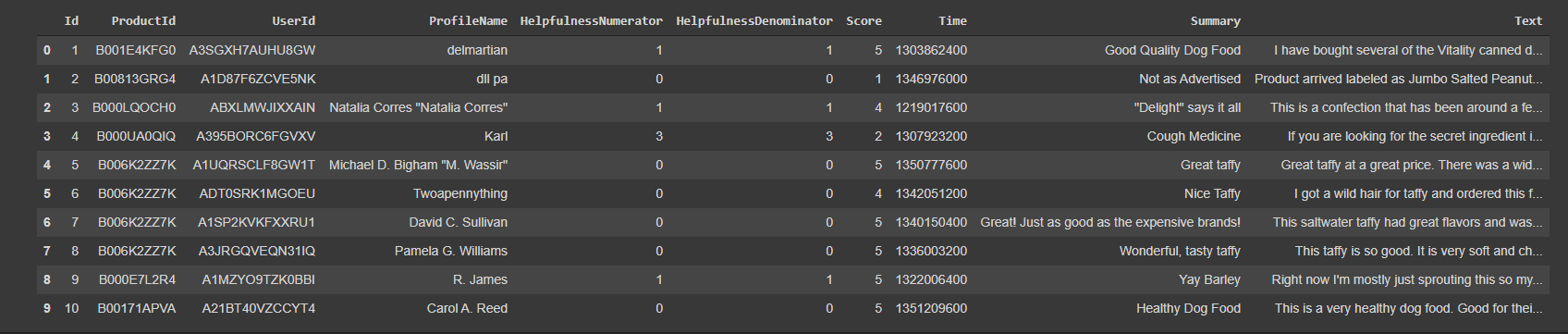
ML Project-Report

Title:

*Classification of Amazon Customer Review of Product Text into Positive or Negative using Machine Learning Algorithms.*

**Data set description:-**



ProductID:- Unique ID for a Product on Amazon.

User ID:- every user who enters a review has a unique ID.

Profile Name:- Name of the user.

Helpfulness Numerator:- The number of People who found the review helpful.

Helpfulness Denominator:- The number of People who found review to be un-helpful.

Score:-Numerical rating given by the user for the Product (1-2: Bad, 3:Neutral, 4-5: Good).

Summary:- Short 1 liner about the user experience about the product.

Review:- Actual experience of the user with the product.

Total initial number of rows in the Dataset= 568,454.

The number of rows of data after doing some data cleaning(removing duplicate reviews, removing neutral reviews, removing reviews that users did not find helpful(HN<HD))=364173.

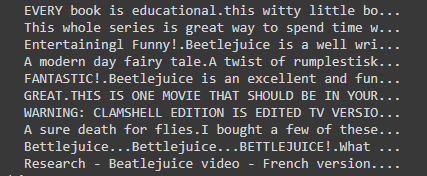
Our Data of interest is only the Summary + Review ( Textual data)-Acts as a set of features(Xi)

Score value acts as our Y values(score>3 then yi=1,score < 3 then yi=0)

Our Data of interest is only the Summary + Review ( Textual data)-Acts as a set of features(Xi)

Let us Combine these 2 columns into one single column.

The Dataset of interest looks somewhat like this.



Let us split the data into Train, Test and Cross-validation dataset.

Training data(50%)

Cross validation(20%)

Test(30%).

**Vectorization of Textual input.**

We cannot directly apply ML algorithms on the Textual data. We need to vectorize the data using some NLP algorithms.

The algorithms used are

1:Bag Of Words

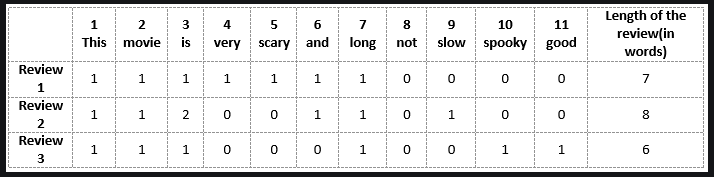
2: TF-IDF(Term Frequency- Inverse Document Frequency)

**Bag-of Words:-**

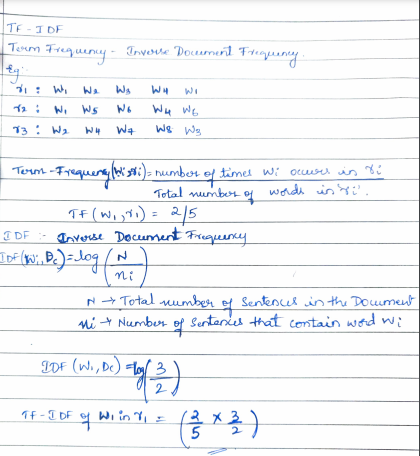
Suppose you have 3 sentences-

1. This movie is very scary and long.
2. This movie is long and is slow.
3. This spooky long movie is good.
4. Etc…..

These sentences are Vectorized as follow:-



**TF-IDF (Term Frequency-Inverse Document Frequency)**



Now Our data is Vectorized and we can experiment with various ML algorithms .

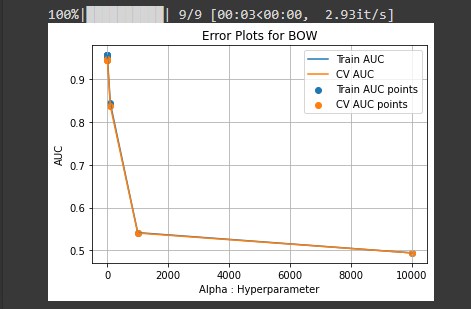
Algorithm 1- Naïve Bayes.

Applying on the Data that is Vectorized using the BOW technique.

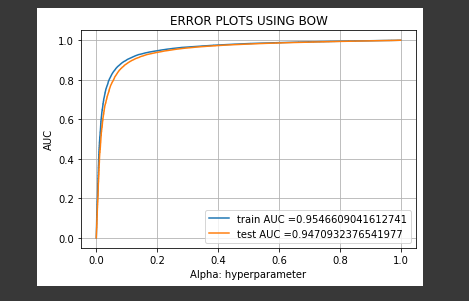
Naïve bayes is specifically known to work well on textual data.

So after Naïve Bayes with Alpha values:-[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

The AUC plot is as follows:



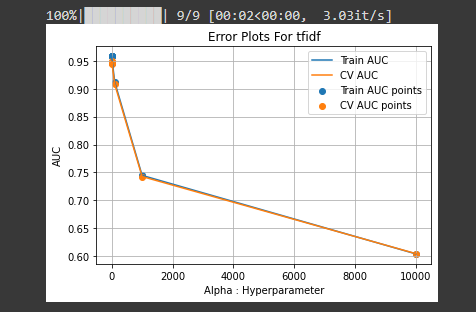
The Error plot on the Test data vectorized using the BOW Vectorization.



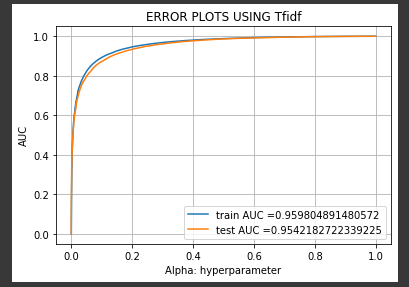
Applying on the Data that is Vectorized using the TF-IDF technique.

So after Naïve Bayes with Alpha values:-[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

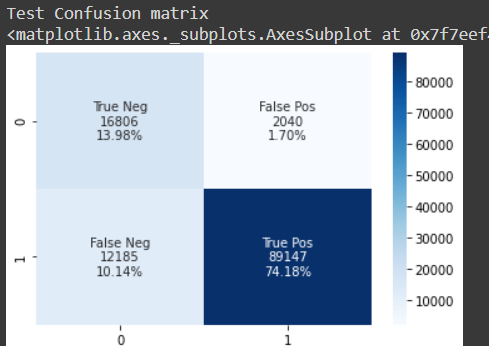
The AUC plot is as follows:



The Error plot on the Test data vectorized using the BOW Vectorization.



The test Confusion matrix is as follows



**Analysis**

Let us analyze why TF-IDF gave us a better accuracy than BOW.

Our Intuition is:-

The way TF-IDF is designed.

As we dig deeper into the way TFIDF works, we will notice that TFIDF gives more weightage to the words that appear rarely in the document than other words that appear frequently.

So, for example consider a word, “**Enthralling**” that is present in only one of the reviews in the train data and this is classified as a positive review.

Now, Because of the rareness of the word in the document, the IDF(Inverse Document frequency) of the word is going to be extremely large and a large weight will be given to the word.

When this word appears in any of the reviews in the Test data. It is definitely going to be classified as positive.

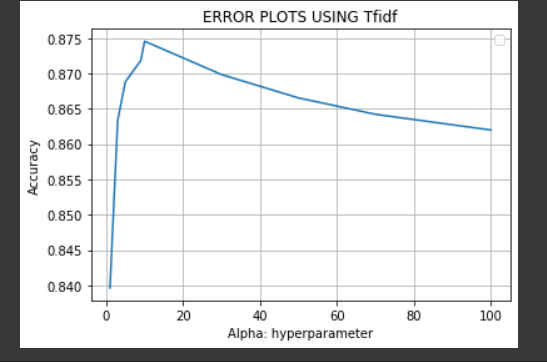
Now that we know TFIDF gives us a better accuracy for the above mentioned reason, let us use the data vectorized by using TFIDF across all future algorithms.

Algorithm 2

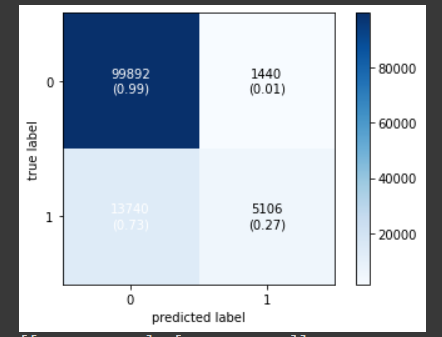
**K-nearest Neighbours.**

We executed the KNN algorithm with various values for the Hyperparameter k.

We got accuracy scores as shown in the below graph.



By using the Hyperparameter K=10 and testing on the unseen data we got an accuracy of 88% and the confusion matrix is as follows.



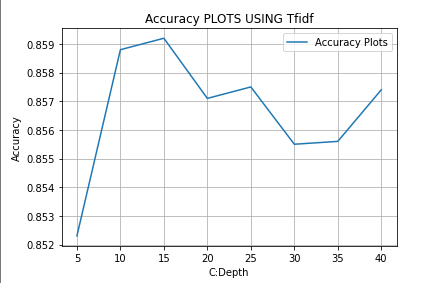
Algorithm 3

**Decision Trees**

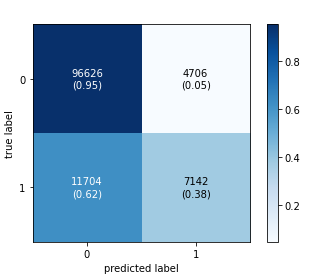
Let us see how decision tree woks on textual data.

We applied the Decision tree classifier with various Depth values such as [1,5,10,15,20,30,40,50]

The accuracy score graph can be seen below



Now by using the Depth value as 15, We apply the DT on Unseen Test data and we got an accuracy of 87% .The Confusion matrix is as shown below.



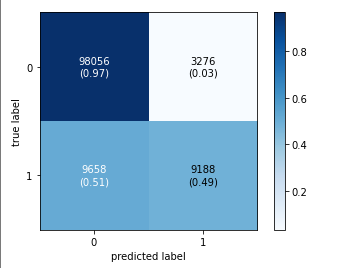
Algorithm 4

**Bagging with Decision Trees.**

Let us see how Bagging a number of decision trees improves the accuracy.

Here we use the Decision trees with depth of 15 and got an accuracy of 90% as compared to the 87% from Decision trees.

The Confusion matrix is as follows



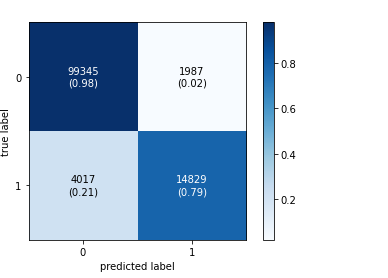
Algorithm 5

**Support Vector Machines.**

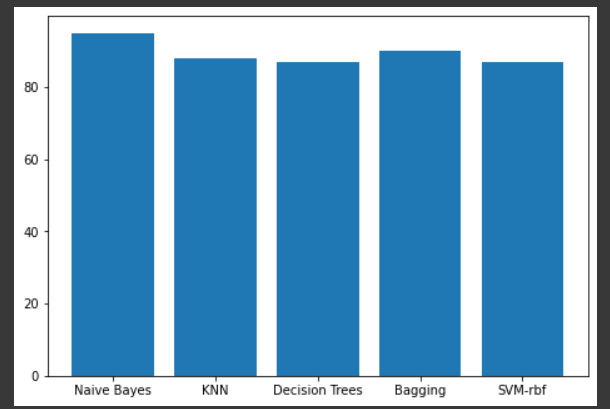
Lets see how SVM with RBF kernel performs on the textual data.

Here we have used the hyperparameter C to take values from 0.01,0.1,1,3,5,7 and we got the best accuracy of 88%.

The confusion matrix is as follows.



The overall performance of all algorithms on the data can be seen below:



**Analysis**

Why did Naïve Bayes perform the best on textual data?

As we learnt in class, Naïve Bayes assumes that, all the features are independent of each-other.

So, here every word in the review is itself responsible for a review to be classified into positive or negative. But in other techniques, this is not the case.

Hence, The values of True-Positives and True-negatives are greatly increased in NB when compared to other models.