Pradip Dharam

4. Naive Bayes on Amazon BoW TFIDF

forked from 3. k-NN on Amazon BoW TFIDF by Pradip Dharam (+0/−0)

last run 2 hours ago · IPython Notebook HTML

using data from multiple data sources · Private Make Public

3] Apply Naive Bayes on Amazon product reviews data-set [M]

Implement Naive Bayes on Amazon reviews dataset

1. Apply Naive Bayes both Bernaulli and Multinomial
2. Find right alpha using cross validation
3. Get the important features; feature importance for positive class and negative class. Which words are more prominent or most often found? which words have highest probability for negative review?
4. Get accuracy, precision, recall, f1 score, confusion matrix, TNR, FNR, FPR, TPR

SOLUTION:

Importing required lib's

In [1]:

```
%matplotlib inline

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

from sklearn.cross_validation import train_test_split
from sklearn.model_selection import TimeSeriesSplit
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import cross_val_score
from collections import Counter
```

```
from sklearn.metrics import accuracy_score
from sklearn import cross_validation
from sklearn import preprocessing
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/cross_validation.py:41: DeprecationWarning: Thi
s module was deprecated in version 0.18 in favor of the model_selection module into which all
the refactored classes and functions are moved. Also note that the interface of the new CV ite
rators are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

Creting connection to work with '../input/database.sqlite' sqllite database file

In [2]:
```
# using the SQLite Table to read data.
con = sqlite3.connect('../input/amazon-fine-food-reviews/database.sqlite')
```

#

TEXT PREPROCESSING Begins Here

Filtering only positive and negative reviews i.e. not taking into consideration those reviews with Score=3

In [3]:
```
filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3""", con)

# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative rating.
def partition(x):
    if x < 3:
        return 'negative'
    return 'positive'

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
```

Duplicate entries like below needs tobe removed and instead we need to beep one record

In [4]:
```
# Query to show Duplicate entries
display= pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 AND UserId="AR5J8UI46CUR
R" ORDER BY ProductID""", con)
display
```

Out[4]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	119957760(
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	119957760(
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	119957760(
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	119957760(
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	119957760(

Removing duplicates

- Sorting the data first
- filtering the records on features "UserId","ProfileName","Time" and "Text" then just keeping first occurence of rest of the features

In [5]:
```
#Sorting data according to ProductId in ascending order
```

In [5]:
```
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind=
'quicksort', na_position='last')

#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first',
inplace=False)
final.shape
```

Out[5]:
```
(364173, 10)
```

Checking to see how much % of data still remains after duplicate records removal

In [6]:
```
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[6]:
```
69.25890143662969
```

Indentified invad records as per business logic which needs to removed

- As per business logic, records which are valid where HelpfulnessNumerator is less than HelpfulnessDenomirator and invalid records should be removed

In [7]:
```
display= pd.read_sql_query("""SELECT * FROM Reviews WHERE Score != 3 AND Id=44737 OR Id=64422 O
RDER BY ProductID""", con)
display
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	122489280(
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	121288320(

Removing invalid records as per above comment

In [8]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
final.shape
```

Out[8]:
```
(364171, 10)
```

Identifying whether review text contains HTML tags

- One of the review text has HTML tags which needs to be removed as part of data pre processing

In [9]:
```
# find sentences containing HTML tags
import re
i=0;
for sent in final['Text'].values:
    if (len(re.findall('<.*?>', sent))):
        print(i)
        print(sent)
        break;
    i += 1;
```

```
6
I set aside at least an hour each day to read to my son (3 y/o). At this point, I consider mys
elf a connoisseur of children's books and this is one of the best. Santa Clause put this under
the tree. Since then, we've read it perpetually and he loves it.<br /><br />First, this book t
aught him the months of the year.<br /><br />Second, it's a pleasure to read. Well suited to
1.5 y/o old to 4+.<br /><br />Very few children's books are worth owning. Most should be borro
wed from the library. This book, however, deserves a permanent spot on your shelf. Sendak's be
st.
```

Avoiding stop words removal

- Removing stopwords like below will manipulates the meaning of scentense and this will spoil the r=end result of the trained

model up to some extent ^^so its bettet to keep stop words

- **Some stop words:** before, after, above, below, up, down, on, off, over under, again, further, all, any, each, no, nor, don't, not nor'

In [10]:
```
from nltk.corpus import stopwords
stopwords.words('english')
# stop = set(stopwords.words('english')) #set of stopwords
# if(cleaned_words.lower() not in stop):
```

Out[10]:
```
['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',
                'against',
                'between',
                'into',
                'through',
                'during',
```

```
                'before',
                'after',
                'above',
                'below',
                'to',
                'from',
                'up',
                'down',
                'in',
                'out',
                'on',
                'off',
                'over',
                'under',
                'again',
                'further',
                'then',
                'once',
                'here',
                'there',
                'when',
                'where',
                'why',
                'how',
                'all',
                'any',
                'both',
                'each',
                'few',
                'more',
                'most',
                'other',
                'some',
                'such',
                'no',
                'nor',
                'not',
                'only',
                'own',
                'same',
                'so',
                'than',
                'too',
                'very',
                's',
                't',
```

```
                'can',
                'will',
                'just',
                'don',
                "don't",
                'should',
                "should've",
                'now',
                'd',
                'll',
                'm',
                'o',
                're',
                've',
                'y',
                'ain',
                'aren',
                "aren't",
                'couldn',
                "couldn't",
                'didn',
                "didn't",
                'doesn',
                "doesn't",
                'hadn',
                "hadn't",
                'hasn',
                "hasn't",
                'haven',
                "haven't",
                'isn',
                "isn't",
                'ma',
                'mightn',
                "mightn't",
                'mustn',
                "mustn't",
                'needn',
                "needn't",
                'shan',
                "shan't",
                'shouldn',
                "shouldn't",
                'wasn',
                "wasn't",
                'weren',
```

```
 "weren't",
 'won',
 "won't",
 'wouldn',
 "wouldn't"]
```

Stemming, HTML removal, cleaning punctuation or special characters, Lemmatization, converting to lover case

- Also removing the words which has length 1
- **This section is commented since this takes time and one time processing. Pre-processing results are stored in csv file and every time we will get data from csv file**

In [11]:

```
"""
import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
#from nltk.stem import PorterStemmer
#from nltk.stem.wordnet import WordNetLemmatizer

sno = nltk.stem.SnowballStemmer('english') #initialising the snowball stemmer

def cleanhtml(sentence): #function to clean the word of any html-tags
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext
def cleanpunc(sentence): #function to clean the word of any punctuation or special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned

#Code for implementing step-by-step the checks mentioned in the pre-processing phase
# this code takes a while to run as it needs to run on 500k sentences.
i=0
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
for sent in final['Text'].values:
    filtered_sentence=[]
    #print(sent);
    sent=cleanhtml(sent) # remove HTML tags
    for w in sent.split():
```

```
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>1)):
                s=(sno.stem(cleaned_words.lower())).encode('utf8')
                #Comment above line to remove stemming an replce this line with below line
                #s=cleaned_words.lower()
                filtered_sentence.append(s)
                if (final['Score'].values)[i] == 'positive':
                    all_positive_words.append(s) #list of all words used to describe positive rev
iews
                if(final['Score'].values)[i] == 'negative':
                    all_negative_words.append(s) #list of all words used to describe negative rev
iews reviews
            else:
                continue
    #print("filtered_sentence: ",filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    #print("**********************************************************************")

    # Trimming b' and ' from the left and right respectively
    str2=str(str1).lstrip('b\'').rstrip('\'')
    #print("str2:",str2)
    final_string.append(str2)
    i+=1

final['CleanedText']=final_string #adding a column of CleanedText which displays the data after p
re-processing of the review
"""
```

Out[11]:

```
'\nimport re\n# Tutorial about Python regular expressions: https://pymotw.com/2/re/\nimport st
ring\n#from nltk.stem import PorterStemmer\n#from nltk.stem.wordnet import WordNetLemmatizer\n
\nsno = nltk.stem.SnowballStemmer(\'english\') #initialising the snowball stemmer\n\ndef clean
html(sentence): #function to clean the word of any html-tags\n    cleanr = re.compile(\'<.*?>
\')\n    cleantext = re.sub(cleanr, \' \', sentence)\n    return cleantext\ndef cleanpunc(sent
ence): #function to clean the word of any punctuation or special characters\n    cleaned = re.
sub(r\'[?|!|\'|"|#]\',r\'\',sentence)\n    cleaned = re.sub(r\'[.|,|)|(|\\|/]\',r\' \',cleane
d)\n    return  cleaned\n\n#Code for implementing step-by-step the checks mentioned in the pre
-processing phase\n# this code takes a while to run as it needs to run on 500k sentences.\ni=0
\nstr1=\' \'\nfinal_string=[]\nall_positive_words=[] # store words from +ve reviews here\nall_
negative_words=[] # store words from -ve reviews here.\ns=\'\'\nfor sent in final[\'Text\'].va
lues:\n    filtered_sentence=[]\n    #print(sent);\n    sent=cleanhtml(sent) # remove HTMl tag
s\n    for w in sent.split():\n        for cleaned_words in cleanpunc(w).split():\n
if((cleaned_words.isalpha()) & (len(cleaned_words)>1)):    \n                s=(sno.stem(clean
ed_words.lower())).encode(\'utf8\')\n                #Comment above line to remove stemming an
replce this line with below line\n                #s=cleaned_words.lower()\n                fi
ltered_sentence.append(s)\n                if (final[\'Score\'].values)[i] == \'positive\': \n
```

```
                    all_positive_words.append(s) #list of all words used to describe positive
reviews\n                if(final[\'Score\'].values)[i] == \'negative\':\n
all_negative_words.append(s) #list of all words used to describe negative reviews reviews
          \n            else:\n                continue \n    #print("filtered_sentence: ",fi
ltered_sentence)\n    str1 = b" ".join(filtered_sentence) #final string of cleaned words\n
#print("**********************************************************************")\n    \n    #
Trimming b\' and \' from the left and right respectively\n    str2=str(str1).lstrip(\'b\'\').r
strip(\'\'\')\n    #print("str2:",str2)\n    final_string.append(str2)  \n    i+=1\n
  \nfinal[\'CleanedText\']=final_string #adding a column of CleanedText which displays the dat
a after pre-processing of the review \n'
```

TEXT PREPROCESSING Ends Here

#

Storing pre-processed data to csv file

- **This section is commented since this takes time and one time processing. Pre-processing results are stored in csv file and every time we will get data from csv file**

In [12]:

```
"""
final_text_processed=pd.DataFrame()
final_text_processed['Text'] = final['Text']
final_text_processed['CleanedText'] = final['CleanedText']
final_text_processed['Time'] = final['Time']
final_text_processed['Score'] = final['Score']
final_text_processed.to_csv('final_text_processed.csv')
"""
```

Output

#

Defining below functions

1. Function NaiveBayes10fold_and_MSE

- This function takes training data both X_train and Y_train to train the model

- This function also takes argument algo where in we can provide whether to use brute or kd_tree
- We need to do time based splitting but not random while training 10 fold cross validation kNN model; I am using TimeSeriesSplit and passing 10 as argument and my traing data is already sorted in ascending order.
- I did added print statements to undestand the execution after writing code. Now those print statements are commented.
- This function plots the graph "Number of Neighbors K')" against "Misclassification Error"
- It takes average accuracy of 10 folds for each k and then calculates the mis-classification error as (1-average_accuracy_for_each_k)
- It then select the optimal k such that misclassification error is minimum and returns the same

In [13]:

```
def NaiveBayes10fold_and_MSE(x_train, y_train, algo='Bernoulli'):
    alphas = list(np.arange(0.01,1,0.01))
    cv_scores = []
    # perform 10-fold cross validation
    time_series_10foldcv = TimeSeriesSplit(n_splits=10)
    for alpha in alphas:
        #print("kNN for k=",k)
        #knn = KNeighborsClassifier(n_neighbors=k, weights='distance', algorithm=algo)
        #knn = KNeighborsClassifier(n_neighbors=k, weights='uniform', algorithm=algo)
        clf = BernoulliNB(alpha=alpha)
        if(algo == 'Multinomial'):
            clf = MultinomialNB(alpha=alpha)
        #Below function does random k' folds and not time based, that why this is not used
        #scores = cross_val_score(knn, x_train, y_train, cv=10, scoring='accuracy')
        this_cv_scores=[]
        #print("Working on 10 fold cross validation for k=",k)
        for train_index, cv_index in time_series_10foldcv.split(x_train):
            # tscv_data.split(x_train) : This will give 10 iterations
            # Each iteration has train and cross validation indexes based on time but not the ran
dom splitting
            # But not the random creation of train and cross validation data sets

            ######### We get the indexes in sequence since TimeSeriesSplit i.t time based splitti
ng #########
            ##print("Train set index min :", np.min(train_index))
            ##print("Train set index max :", np.max(train_index))
            ##print("CV set index min    :", np.min(cv_index))
            ##print("CV set index max    :", np.max(cv_index))

            X_train, X_cv = x_train[train_index], x_train[cv_index]
            Y_train, Y_cv = y_train[train_index], y_train[cv_index]

            #Randomly choosing data points to train the model
```

```
            #sample_indices = np.random.choice(train_index[-1],20000)
            #x_train_sample = X_train[sample_indices]
            #y_train_sample = Y_train[sample_indices]
            #knn.fit(x_train_sample, y_train_sample)

            #lab_enc = preprocessing.LabelEncoder()
            #training_scores_encoded = lab_enc.fit_transform(y_train_sample)

            clf.fit(X_train, Y_train)
            pred = clf.predict(X_cv)
            #print("Type Y_cv: ",type(Y_cv))
            #print("Type pred: ",type(pred))
            #print("Y_cv: ",Y_cv)
            #print("pred: ",pred)
            accuracy = accuracy_score(Y_cv, pred, normalize=True)
            #print("Accuracy:",accuracy)
            this_cv_scores.append(accuracy)

        ##print("k=",k, ", 10 folds cross validation scores ", this_cv_scores)
        ##print("k=",k, ", Agerage is: ", np.average(this_cv_scores))
        cv_scores.append(np.average(this_cv_scores))
    # changing to misclassification error
    MSE = [1 - x for x in cv_scores]
    # determining best k
    best_alpha_index = MSE.index(min(MSE))
    optimal_alpha = alphas[best_alpha_index]
    CV_Accuracy = cv_scores[best_alpha_index] * 100
    # plot misclassification error vs k
    plt.plot(alphas, MSE)
    for xy in zip(alphas, np.round(MSE,3)):
        plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
    plt.xlabel('Alpha Values ')
    plt.ylabel('Misclassification Error')
    plt.show()
    print("The misclassification error for each alpha value is : ", np.round(MSE,3))
    print('\nThe optimal number of alphas is %d and respective accuracy over training data or c
ross validation accuracy is %f%%' % (optimal_alpha,CV_Accuracy))

    return optimal_alpha,CV_Accuracy
#################################################################################################
```

2. Function knn_with_optimal_alpha

- Here, lets test the model over unknown data which is x_test and y_test here
- Train the model using x train and y train; then predict using x test

- Then get the accuracy using prected class labels and y_train and report the accuracy for optimal k

In [14]:

```
def NaiveBayes_with_optimal_alpha(x_train, y_train, x_test, y_test, optimal_alpha, algo='Bernou
lli'):
    # ============================== KNN with k = optimal_k ====================================
==========
    # instantiate learning model k = optimal_k
    #knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k, weights='distance', algorithm=alg
o)
    #knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k, weights='uniform', algorithm=algo)
    clf = BernoulliNB(alpha=optimal_alpha)
    if(algo == 'Multinomial'):
            clf = MultinomialNB(alpha=optimal_alpha)
    # fitting the model
    clf.fit(x_train, y_train)

    # predict the response
    #pred = knn_optimal.predict(x_test)
    # Splitting the test data and then predicting
    test_length=x_test.shape[0]
    splits=5000
    pred=[]
    #print("Predict function cannot entire test data at a time, splitting the test data into ",sp
lits," records in each iteration and then predicting")
    for i in range(0,test_length,splits):
        #print("Idex Numbers Processing : ", i, " ",i+splits)
        x_test_split = x_test[i:i+splits]
        #print("x_test_split shape: ",x_test_split.shape)
        pred_splitted=clf.predict(x_test_split)
        #print("pred_splitted: ",pred_splitted)
        #list_pred_splitted=list(pred_splitted)
        #print("list_pred_splitted : ", list_pred_splitted)
        #print("pred_splitted type: ",type(pred_splitted))
        #print("pred_splitted shape: ",pred_splitted.shape)
        pred=np.concatenate([pred,pred_splitted], axis=0)
        #print("pred: ",pred)
        #print("pred type: ",type(pred))
        #print("pred shape: ",pred.shape)
    #print('-----------------------------------')
    #print("pred: ",pred)
    #print("pred type: ",type(pred))
    #print("pred shape: ",pred.shape)

    # evaluate accuracy
```

```
    acc = accuracy_score(y_test, pred) * 100
    print('\nThe test accuracy or accuracy over unknown data of the ', algo ,' Naive Bayes for
 optimal alpha = %f is %f%%' % (optimal_alpha, acc))
    print('\nOBSERVATION: Able to achieve ', round(acc,2), '% of test accuracy' )
    print('\nCONFUSION METRIX :' )
    c_metrix = confusion_matrix(y_test,pred)
    tn, fp, fn, tp = confusion_matrix(y_test,pred).ravel()
    sns.heatmap(c_metrix.T, square=True, annot=True, fmt='d', cbar=True,cmap='RdYlBu_r',
        xticklabels=['negative','positive'], yticklabels=['negative','positive'])
    plt.title("Confusion Metrix")
    plt.xlabel('True Label')
    plt.ylabel('Predicted Label');

    print("True Negative Rate: ",(tn/(tn+fp))*100)
    print("False Negative Rate: ",(fn/(fn+tp))*100)
    print("False Positive Rate: ",(fp/(tn+fp))*100)
    print("True Positive Rate: ",(tp/(fn+tp))*100)

    Precision = (tp/(tp+fp))
    Recall = (tp/(tp+fn))
    F1_Score = (2 * ((Precision * Recall)/(Precision + Recall)) )
    print("Precision: ",Precision)
    print("Recall: ",Recall)
    print("F1 Score: ", F1_Score)

    print('\n')
    print('Model: ',algo,' Naive Bayes')
    print('Hyper Parameter: alpha =',optimal_alpha)
    print('Train Error: ',round(100-train_acc,2))
    print('Test Error: ',round(100-acc,2))

    return clf.feature_log_prob_
```

3. Function to get prominent features or words from product review data set

In [15]:

```
def showProminentWords(feature_log_prob, feature_log_prob1, feature_names, top_prominent):
    df=pd.DataFrame({
    "PROMINENT_FEATURES": feature_names,
    "LIKELIHOOD_OR_PROBABILITIES": feature_log_prob,
    "feature_log_prob1": feature_log_prob1,
    "feature_log_prob_diff": feature_log_prob - feature_log_prob1})
    df=df.sort_values('feature_log_prob_diff',ascending=False)
    print(df.iloc[:top_prominent][["PROMINENT_FEATURES","LIKELIHOOD_OR_PROBABILITIES"]])
```

#

Reading data from pre-processed csv file

- **Preprocessed data was stored in csv file. Reading the same into panda data frame variable data**

In [16]:
```
final_text_processed=pd.read_csv('../input/final-text-processed/final_text_processed.csv')
```

In [17]:
```
final_text_processed.shape
```

Out[17]:
```
(364171, 5)
```

#

Sorting data in ascending order of time since time based splitting needs to be done

In [18]:
```
#Sorting the data on ascending order of Time since time based splitting needs to be done further
final_text_processed=final_text_processed.sort_values(['Time'], ascending=1)
print("Time min:", final_text_processed['Time'].min())
print("Time max:", final_text_processed['Time'].max())
print("\nPrinting time feature to ensure whether data is sorted in ascending order of time")
final_text_processed['Time']
```

```
Time min: 939340800
Time max: 1351209600

Printing time feature to ensure whether data is sorted in ascending order of time
```

Out[18]:
```
0          939340800
30         940809600
424        944092800
330        944438400
423        946857600
245        947376000
```

308	948240000
215	948672000
261	951523200
325	959990400
427	959990400
241	961718400
242	962236800
485	965001600
837	965779200
868	965779200
249	966297600
296	970531200
844	975974400
360	977184000
329	978134400
845	982800000
425	992217600
342	997228800
270	1001289600
855	1003795200
353	1004054400
32	1009324800
998	1010275200
331	1012780800
...	...
331313	1351209600
274064	1351209600
941	1351209600
273941	1351209600
331386	1351209600
363346	1351209600
215652	1351209600
331387	1351209600
273648	1351209600
18387	1351209600
354060	1351209600
354041	1351209600
132404	1351209600
272372	1351209600
1637	1351209600
361679	1351209600
332259	1351209600
353879	1351209600
16845	1351209600
214181	1351209600
16642	1351209600

```
332848    1351209600
333441    1351209600
353804    1351209600
139151    1351209600
139298    1351209600
333542    1351209600
139690    1351209600
213087    1351209600
57313     1351209600
Name: Time, Length: 364171, dtype: int64
```

In [19]:

```
#import numpy as np
#from sklearn.cross_validation import train_test_split
#a = np.arange(10).reshape((5, 2))
#b = np.arange(5)
#print("Array a: \n",a); print("Array b: \n",b);

#a_train, a_test, b_train, b_test = train_test_split(a, b,  test_size=0.3,  random_state=0)
#print("a_train :\n", a_train)
#print("b_train :\n", b_train)
```

#

Selecting first 100k records for processing as a sample

In [20]:

```
num_of_points=100000
data = final_text_processed[0:num_of_points]

print("Min and max time values for sample data :")
print("Time min:", data['Time'].min())
print("Time max:", data['Time'].max())

X = data['CleanedText']
Y = np.array([1 if x=='positive' else 0 for x in data['Score']])
XYTime=data['Time']
print("X Shape : ",X.shape, " X Ndim: ",X.ndim)
print("Y Shape : ",Y.shape, " Y Ndim: ",Y.ndim)
#print(X); print(Y)
from scipy import stats
stats.describe(Y)
```

```
Min and max time values for sample data :
Time min: 939340800
Time max: 1277164800
X Shape :  (100000,)  X Ndim:  1
Y Shape :  (100000,)  Y Ndim:  1
```

Out[20]:

```
DescribeResult(nobs=100000, minmax=(0, 1), mean=0.87729, variance=0.10765333243332435, skewnes
s=-2.299819344191714, kurtosis=3.28916901591841)
```

#

Splitting dataset into train and test with 70:30 ratio**

In [21]:

```
# Train to test ratio is 70:30
boundry=int(num_of_points*0.7)
print("Boundry: ", boundry)
# split the data set into train and test based in time and not random splitting
x_train = X[:boundry]; x_test = X[boundry:]
y_train = Y[:boundry]; y_test = Y[boundry:]
XYTime_train = XYTime[:boundry]; XYTime_test = XYTime[boundry:]

#x_train, x_test, y_train, y_test = cross_validation.train_test_split(X, Y, test_size=0.3, random
_state=0)
print("x train Shape : ",x_train.shape, " x Ndim: ",x_train.ndim); print("y train Shape : ",y_t
rain.shape, " y Ndim: ",y_train.ndim)
print("x test Shape : ",x_test.shape, " x Ndim: ",x_test.ndim); print("y test Shape : ",y_test.
shape, " y Ndim: ",y_test.ndim)

print("\nMin and max time values for time for both train and test :")
print("Train Time min:", XYTime_train.min())
print("Train Time max:", XYTime_train.max())
print("Test Time min:", XYTime_test.min())
print("Test Time max:", XYTime_test.max())
```

```
Boundry:  70000
x train Shape :  (70000,)  x Ndim:  1
y train Shape :  (70000,)  y Ndim:  1
x test Shape :  (30000,)  x Ndim:  1
y test Shape :  (30000,)  y Ndim:  1
```

```
Min and max time values for time for both train and test :
Train Time min: 939340800
Train Time max: 1256083200
Test Time min: 1256083200
Test Time max: 1277164800
```

In [22]:

```
"""
num_of_points=20
x_test_temp = x_test[0:num_of_points]
x_test=x_test_temp
print("x_test shape: ",x_test.shape)

#BoW
count_vect_train = CountVectorizer(ngram_range=(1,2)) #in scikit-learn
final_counts_bow_train = count_vect_train.fit_transform(x_train.values)
final_counts_bow_train.shape
print("final_counts_bow_train shape: ",final_counts_bow_train.shape)

features_count_vect_train = count_vect_train.get_feature_names()

count_vect_test = CountVectorizer(ngram_range=(1,2), vocabulary = features_count_vect_train)
counts_bow_test = count_vect_test.fit(x_train.values)
final_counts_bow_test = counts_bow_test.transform(x_test.values)

print("final_counts_bow_test shape: ",final_counts_bow_test.shape)
test_length=final_counts_bow_test.shape[0]
print("test_length: ",test_length)
print("test_length type: ",type(test_length))

test_length=final_counts_bow_test.shape[0]
splits=4
pred=[]

for i in range(0,test_length,splits):
    print(i, " ",i+splits-1)
    x_test_split = final_counts_bow_test[i:i+splits]
    print("x_test_split shape: ",x_test_split.shape)
    pred_splitted=knn_optimal.predict(x_test_split)
    #print("pred_splitted: ",pred_splitted)
    #list_pred_splitted=list(pred_splitted)
    print("list_pred_splitted : ", list_pred_splitted)
    print("pred_splitted type: ",type(pred_splitted))
    print("pred_splitted shape: ",pred_splitted.shape)
    pred=np.concatenate([pred,pred_splitted], axis=0)
```

```
    print("pred: ",pred)
    print("pred type: ",type(pred))
    print("pred shape: ",pred.shape)
print('-----------------------------------')
print("pred: ",pred)
print("pred type: ",type(pred))
print("pred shape: ",pred.shape)
"""
```

Out[22]:

```
'\nnum_of_points=20\nx_test_temp = x_test[0:num_of_points]\nx_test=x_test_temp\nprint("x_test
shape: ",x_test.shape)\n\n#BoW\ncount_vect_train = CountVectorizer(ngram_range=(1,2)) #in scik
it-learn\nfinal_counts_bow_train = count_vect_train.fit_transform(x_train.values)\nfinal_count
s_bow_train.shape\nprint("final_counts_bow_train shape: ",final_counts_bow_train.shape)\n\nfea
tures_count_vect_train = count_vect_train.get_feature_names()\n\ncount_vect_test = CountVector
izer(ngram_range=(1,2), vocabulary = features_count_vect_train) \ncounts_bow_test = count_vect
_test.fit(x_train.values)\nfinal_counts_bow_test = counts_bow_test.transform(x_test.values)\n
\nprint("final_counts_bow_test shape: ",final_counts_bow_test.shape)\ntest_length=final_counts
_bow_test.shape[0]\nprint("test_length: ",test_length)\nprint("test_length type: ",type(test_l
ength))\n\ntest_length=final_counts_bow_test.shape[0]\nsplits=4\npred=[]\n\nfor i in range(0,t
est_length,splits):\n    print(i, " ",i+splits-1)\n    x_test_split = final_counts_bow_test[i:
i+splits]\n    print("x_test_split shape: ",x_test_split.shape)\n    pred_splitted=knn_optima
l.predict(x_test_split)\n    #print("pred_splitted: ",pred_splitted)\n    #list_pred_splitted=
list(pred_splitted)\n    print("list_pred_splitted : ", list_pred_splitted)\n    print("pred_s
plitted type: ",type(pred_splitted))\n    print("pred_splitted shape: ",pred_splitted.shape)\n
    pred=np.concatenate([pred,pred_splitted], axis=0)\n    print("pred: ",pred)\n    print("pr
ed type: ",type(pred))\n    print("pred shape: ",pred.shape)\nprint(\'-----------------------
-----------\')\nprint("pred: ",pred)\nprint("pred type: ",type(pred))\nprint("pred shape: ",pr
ed.shape)\n'
```

#

Top prominent number of words

In [23]:

```
top_prominent=20
```

#

BAG OF WORDS

Getting the BoW vector representation for traning data set

In [24]:
```
#BoW
count_vect_train = CountVectorizer(ngram_range=(1,2)) #in scikit-learn
final_counts_bow_train = count_vect_train.fit_transform(x_train.values)

final_counts_bow_train.shape
```

Out[24]:
```
(70000, 756453)
```

Storing vocablary of training data

In [25]:
```
features_count_vect_train = count_vect_train.get_feature_names()
features_count_vect_train[:20]
```

Out[25]:
```
['aa',
 'aa and',
 'aa cell',
 'aa coffe',
 'aa doubl',
 'aa grade',
 'aa offer',
 'aa pod',
 'aa superior',
 'aa this',
 'aa though',
 'aa to',
 'aaa',
 'aaa perfect',
 'aaaaaaaaagghh',
 'aaaaah',
 'aaaaah satisfi',
 'aaaaah they',
 'aaaaahhhhhhhhhhhhhhh',
 'aaaaahhhhhhhhhhhhhhh the']
```

Getting the BoW vector representation for test data set

- **Training vectorizer on test data using vectors obtained from train data**

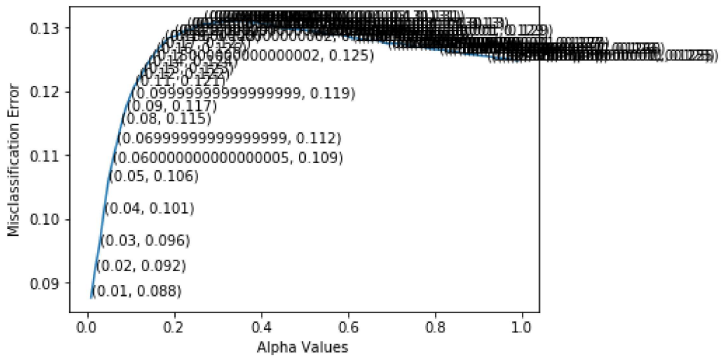
In [26]:
```
count_vect_test = CountVectorizer(ngram_range=(1,2), vocabulary = features_count_vect_train)
counts_bow_test = count_vect_test.fit(x_train.values)
final_counts_bow_test = counts_bow_test.transform(x_test.values)

final_counts_bow_test.shape
```

Out[26]:
```
(30000, 756453)
```

1. Bernaulli Naive Bayes for BoW

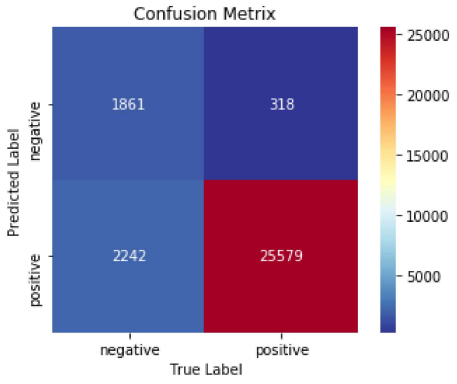
1. ### **Optimal alpha accuracy: 10 fold cross validation Bernaulli Naive Bayes for BoW**
2. - Performing **Bernaulli Naive Bayes 10 fold cross validation** over training data to get the accuracy and mis classification error
3. - Getting optimal alpha and with low misclassification error
4. - Then accuracy for test data is reported using optimal alpha

In [27]:
```
optimal_alpha,train_acc = NaiveBayes10fold_and_MSE(final_counts_bow_train, y_train, 'Bernoulli'
)
feature_log_prob = NaiveBayes_with_optimal_alpha(final_counts_bow_train, y_train, final_counts_
bow_test, y_test, optimal_alpha, 'Bernoulli')
```




```
The misclassification error for each alpha value is :  [0.088 0.092 0.096 0.101 0.106 0.109 0.
112 0.115 0.117 0.119 0.121 0.122
```

```
0.123 0.124 0.125 0.126 0.127 0.128 0.128 0.129 0.129 0.129 0.13  0.13
0.13  0.13  0.131 0.131 0.131 0.131 0.131 0.131 0.131 0.131 0.131 0.131
0.131 0.131 0.131 0.131 0.13  0.13  0.13  0.13  0.13  0.13  0.13  0.13
0.13  0.13  0.129 0.129 0.129 0.129 0.129 0.129 0.129 0.129 0.129 0.129
0.128 0.128 0.128 0.128 0.128 0.128 0.128 0.128 0.127 0.127 0.127 0.127
0.127 0.127 0.127 0.127 0.127 0.127 0.127 0.127 0.126 0.126 0.126 0.126
0.126 0.126 0.126 0.126 0.126 0.126 0.126 0.125 0.125 0.125 0.125 0.125
0.125 0.125 0.125]

The optimal number of alphas is 0 and respective accuracy over training data or cross validati
on accuracy is 91.235266%

The test accuracy or accuracy over unknown data of the  Bernoulli  Naive Bayes for optimal alp
ha = 0.010000 is 91.466667%

OBSERVATION: Able to achieve  91.47 % of test accuracy

CONFUSION METRIX :
True Negative Rate:  45.35705581281989
False Negative Rate:  1.2279414604008185
False Positive Rate:  54.642944187180106
True Positive Rate:  98.77205853959919
Precision:  0.9194133927608641
Recall:  0.9877205853959918
F1 Score:  0.9523437209129156


Model:  Bernoulli  Naive Bayes
Hyper Parameter: alpha = 0.01
Train Error:  8.76
Test Error:  8.53
```



OBSERVATIONS: Bernaulli Naive Bayes for BoW

- 1) The misclassification error for each alpha value is as above in the output
- 2) The optimal alpha is 0.01 and respective accuracy over training data or cross validation accuracy is 91.23%
- 3) The test accuracy or accuracy over unknown data of the Bernoulli Naive Bayes for optimal alpha = 0.01 is 91.466667%
- 4) **Able to achieve 91.47 % of test accuracy**
- 5) **Only 45.35% of data points are predited to be negative out of all negative points. IT LOOKS LIKE OUR MODEL IS A DUMB MODEL, THIS IS BECAUSE OUR DATASET IS IMBALANCED**
- 6) Almost 1.23% of the positive data points are predictes as negative
- 7) **Almost 54.64% of data points which are actually negative and are predicted as positive by the model. IT LOOKS LIKE OUR MODEL IS A DUMB MODEL, THIS IS BECAUSE OUR DATASET IS IMBALANCED**
- 8) Almost 98.77% of data points from the test data are predicted to be positive out of all actual positive data points
- 9) For hyper parameter alpha=0.01, the train error is 8.76% and test error is 8.53%
- Precision is 0.9194 i.e. **only 91.94% of points in the test data are actually positive out of all points predicted as positive**
- Recall is 0.9877 i.e. **98.77% of data points from the test data are predicted to be positive out of all actual positive data points**
- Good F1 score 0.9523 since its close to 1.

In [28]:
```
prob_no_class=np.array(feature_log_prob[0])
prob_yes_class=np.array(feature_log_prob[1])
```

Top 20 prominent features or words which influences class label to be predicted as NEGATIVE using Bernaulli Naive Bayes for BoW

In [29]:
```
showProminentWords(prob_no_class, prob_yes_class, features_count_vect_train, top_prominent)
```

	PROMINENT_FEATURES	LIKELIHOOD_OR_PROBABILITIES
744255	worst tast	-5.711775
744158	worst coffe	-6.442264
140947	complet wast	-6.522242
363619	lesson learn	-6.609178
662667	this crap	-6.609178
108515	cannot return	-6.704397
695247	unfortun was	-6.704397
275846	great disappoint	-6.704397
240188	follow all	-6.704397
585204	so horribl	-6.809647
634561	terribl disappoint	-6.809647
1719	absolut aw	-6.927291

526445	real disappoint	-6.927291
404142	misrepres	-6.927291
552611	same symptom	-6.927291
338688	it poor	-7.060644
663124	this garbag	-7.060644
658725	they refund	-7.060644
533185	refund but	-7.060644
724614	were moldi	-7.060644

Top 20 prominent features or words which influences class label to be predicted as POSITIVE using Bernaulli Naive Bayes for BoW

In [30]:
```
showProminentWords(prob_yes_class, prob_no_class, features_count_vect_train, top_prominent)
```

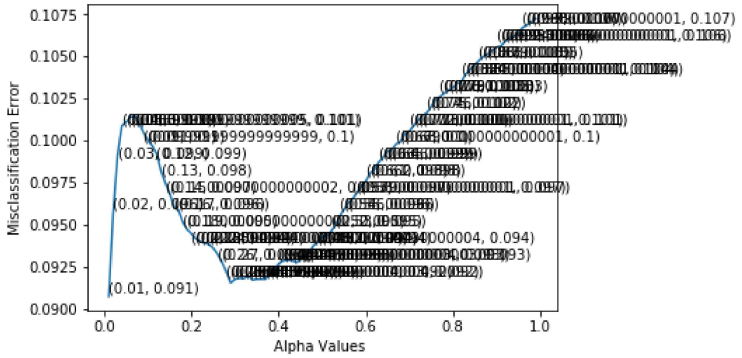
	PROMINENT_FEATURES	LIKELIHOOD_OR_PROBABILITIES
18007	also great	-5.452409
108718	cant go	-5.483062
168422	definit buy	-5.704252
212491	everyon love	-5.908153
27569	and friend	-5.969518
581990	smile	-5.969518
411709	most delici	-6.076279
275688	great choic	-6.104850
434318	not disappoint	-6.126828
487869	perfect blend	-6.156903
528730	rebecca review	-6.172287
556343	say enough	-6.211814
89964	breakfast or	-6.269917
347455	just perfect	-6.278500
521161	quick snack	-6.304700
9278	after dinner	-6.322556
505270	prefer this	-6.331605
417057	must tri	-6.349953
426896	new favorit	-6.368643
457917	onli downsid	-6.378121

2. Multinomial Naive Bayes for BoW

Optimal alpha accuracy: 10 fold cross validation Multinomial Naive Bayes for BoW

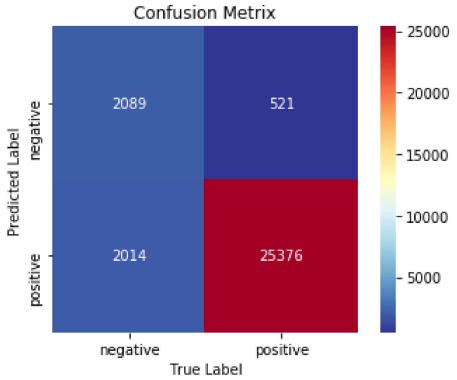
- Performing **Multinomial Naive Bayes 10 fold cross validation** over training data to get the accuracy and mis classification error
- Getting optimal alpha and with low misclassification error
- Then accuracy for test data is reported using optimal alpha

In [31]:
```
optimal_alpha,train_acc = NaiveBayes10fold_and_MSE(final_counts_bow_train, y_train, 'Multinomia
l')
feature_log_prob = NaiveBayes_with_optimal_alpha(final_counts_bow_train, y_train, final_counts_
bow_test, y_test, optimal_alpha, 'Multinomial')
```




```
The misclassification error for each alpha value is :  [0.091 0.096 0.099 0.101 0.101 0.101 0.
101 0.101 0.1   0.1   0.1   0.099
 0.098 0.097 0.097 0.096 0.096 0.095 0.095 0.094 0.094 0.094 0.094 0.094
 0.094 0.093 0.093 0.092 0.092 0.092 0.092 0.092 0.092 0.092 0.092 0.092
 0.092 0.092 0.092 0.093 0.093 0.093 0.093 0.093 0.093 0.093 0.093 0.094
 0.094 0.094 0.094 0.095 0.095 0.096 0.096 0.096 0.097 0.097 0.097 0.098
 0.098 0.098 0.099 0.099 0.099 0.099 0.1   0.1   0.1   0.101 0.101 0.101
 0.101 0.102 0.102 0.102 0.103 0.103 0.103 0.103 0.103 0.104 0.104 0.104
 0.104 0.105 0.105 0.105 0.105 0.106 0.106 0.106 0.106 0.106 0.106 0.107
 0.107 0.107 0.107]

The optimal number of alphas is 0 and respective accuracy over training data or cross validati
on accuracy is 90.925664%

The test accuracy or accuracy over unknown data of the  Multinomial  Naive Bayes for optimal a
lpha = 0.010000 is 91.550000%

OBSERVATION: Able to achieve  91.55 % of test accuracy
```

```
CONFUSION METRIX :
True Negative Rate:  50.913965391177186
False Negative Rate:  2.0118160404680077
False Positive Rate:  49.086034608822814
True Positive Rate:  97.988183959532
Precision:  0.9264695144213216
Recall:  0.9798818395953199
F1 Score:  0.9524274213222736


Model:  Multinomial  Naive Bayes
Hyper Parameter: alpha = 0.01
Train Error:  9.07
Test Error:  8.45
```



OBSERVATIONS: Multinomial Naive Bayes for BoW

- 1) The misclassification error for each alpha value is as above in the output
- 2) The optimal alpha is 0.01 and respective accuracy over training data or cross validation accuracy is 90.925664%
- 3) The test accuracy or accuracy over unknown data of the Multinomial Naive Bayes for optimal alpha = 0.01 is 91.550000%
- 4) **Able to achieve 91.55 % of test accuracy**
- 5) **Only 50.91% of data points are predited to be negative out of all negative points. IT LOOKS LIKE OUR MODEL IS A DUMB MODEL, THIS IS BECAUSE OUR DATASET IS IMBALANCED**
- 6) Almost 2.01% of the positive data points are predictes as negative
- 7) **Almost 49.08% of data points which are actually negative and are predicted as positive by the model. IT LOOKS LIKE OUR MODEL IS A DUMB MODEL, THIS IS BECAUSE OUR DATASET IS IMBALANCED**
- 8) Almost 97.98% of data points from the test data are predicted to be positive out of all actual positive data points
- 9) For hyper parameter alpha=0.01, the train error is 9.07% and test error is 8.45%

- Precision is 0.9264 i.e. **only 92.64% of points in the test data are actually positive out of all points predicted as positive**
- Recall is 0.9798 i.e. **97.98% of data points from the test data are predicted to be positive out of all actual positive data points**
- Good F1 score 0.9524 since its close to 1.

In [32]:
```
prob_no_class=np.array(feature_log_prob[0])
prob_yes_class=np.array(feature_log_prob[1])
```

Top 20 prominent features or words which influences class label to be predicted as NEGATIVE using Multinomial Naive Bayes for BoW

In [33]:
```
showProminentWords(prob_no_class, prob_yes_class, features_count_vect_train, top_prominent)
```

	PROMINENT_FEATURES	LIKELIHOOD_OR_PROBABILITIES
744255	worst tast	-10.687817
744158	worst coffe	-11.418306
209363	euro food	-11.418306
140947	complet wast	-11.498285
363619	lesson learn	-11.585220
662667	this crap	-11.585220
594752	soy jerki	-11.680440
108515	cannot return	-11.680440
240188	follow all	-11.680440
275846	great disappoint	-11.680440
695247	unfortun was	-11.680440
663124	this garbag	-11.785689
526497	real fiber	-11.785689
634561	terribl disappoint	-11.785689
585204	so horribl	-11.785689
118945	chd	-11.785689
532507	refin fructos	-11.785689
368764	like vomit	-11.903334
1719	absolut aw	-11.903334
113932	catalina dress	-11.903334

Top 20 prominent features or words which influences class label to be predicted as POSITIVE using Multinomial Naive Bayes for BoW

In [34]:

```
showProminentWords(prob_yes_class, prob_no_class, features_count_vect_train, top_prominent)
```

	PROMINENT_FEATURES	LIKELIHOOD_OR_PROBABILITIES
18007	also great	-10.330920
108718	cant go	-10.369091
168422	definit buy	-10.590281
581990	smile	-10.770653
212491	everyon love	-10.794182
27569	and friend	-10.818279
411709	most delici	-10.955291
275688	great choic	-10.955291
487869	perfect blend	-11.012857
434318	not disappoint	-11.012857
528730	rebecca review	-11.058316
556343	say enough	-11.097844
89964	breakfast or	-11.155947
347455	just perfect	-11.155947
9278	after dinner	-11.164530
521161	quick snack	-11.181920
505270	prefer this	-11.217635
417057	must tri	-11.217635
426896	new favorit	-11.245284
457917	onli downsid	-11.254673

#

TERM FREQUENCY-INVERSE DOCUMENT FREQUENCY

Getting the tf-idf vector representation for training data set

In [35]:
```
#TF-IDF
tf_idf_vect_train = TfidfVectorizer(ngram_range=(1,2))
final_count_tf_idf_train = tf_idf_vect_train.fit_transform(x_train.values)
print("final_count_tf_idf_train shape: ",final_count_tf_idf_train.shape)
```

```
final_count_tf_idf_train shape:  (70000, 756453)
```

Storing vocablary of training data

In [36]:
```
features_count_vect_train = tf_idf_vect_train.get_feature_names()
features_count_vect_train[:20]
```

Out[36]:
```
['aa',
 'aa and',
 'aa cell',
 'aa coffe',
 'aa doubl',
 'aa grade',
 'aa offer',
 'aa pod',
 'aa superior',
 'aa this',
 'aa though',
 'aa to',
 'aaa',
 'aaa perfect',
 'aaaaaaaaagghh',
 'aaaaah',
 'aaaaah satisfi',
 'aaaaah they',
 'aaaaahhhhhhhhhhhhhhh',
 'aaaaahhhhhhhhhhhhhhh the']
```

Getting the tf-idf vector representation for test data set

In [37]:
```
tf_idf_vect_test = TfidfVectorizer(ngram_range=(1,2), vocabulary = features_count_vect_train)
counts_tf_idf_test = tf_idf_vect_test.fit(x_train.values)
final_counts_tf_idf_test = counts_tf_idf_test.transform(x_test.values)
print("final_counts_tf_idf_test shape: ",final_counts_tf_idf_test.shape)
```

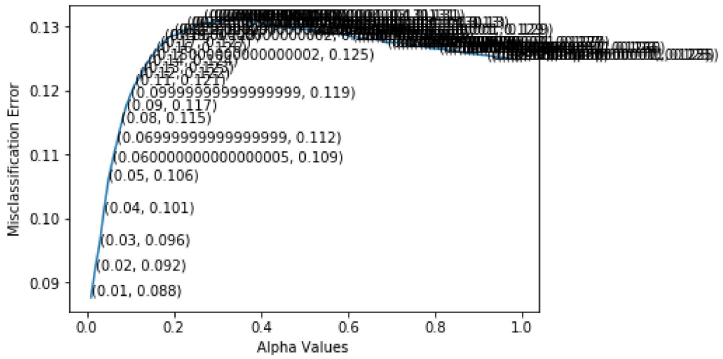
```
final_counts_tf_idf_test shape:  (30000, 756453)
```

1. Bernaulli Naive Bayes for TF-IDF

Optimal alpha accuracy: 10 fold cross validation Bernoulli Naive Bayes for TF-IDF

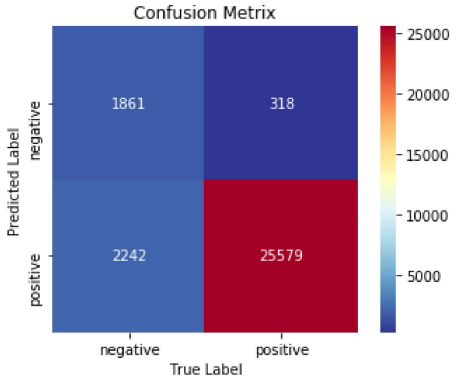
- Performing **Bernoulli Naive Bayes 10 fold cross validation** over training data to get the accuracy and mis classification error
- Getting optimal alpha and with low misclassification error
- Then accuracy for test data is reported using optimal alpha

In [38]:
```
optimal_alpha,train_acc = NaiveBayes10fold_and_MSE(final_count_tf_idf_train, y_train, 'Bernoull
i')
feature_log_prob = NaiveBayes_with_optimal_alpha(final_count_tf_idf_train, y_train, final_count
s_tf_idf_test, y_test, optimal_alpha, 'Bernoulli')
```




```
The misclassification error for each alpha value is :  [0.088 0.092 0.096 0.101 0.106 0.109 0.
112 0.115 0.117 0.119 0.121 0.122
 0.123 0.124 0.125 0.126 0.127 0.128 0.128 0.129 0.129 0.129 0.13  0.13
 0.13  0.13  0.131 0.131 0.131 0.131 0.131 0.131 0.131 0.131 0.131 0.131
 0.131 0.131 0.131 0.131 0.13  0.13  0.13  0.13  0.13  0.13  0.13  0.13
 0.13  0.13  0.129 0.129 0.129 0.129 0.129 0.129 0.129 0.129 0.129 0.129
 0.128 0.128 0.128 0.128 0.128 0.128 0.128 0.128 0.127 0.127 0.127 0.127
 0.127 0.127 0.127 0.127 0.127 0.127 0.127 0.127 0.126 0.126 0.126 0.126
 0.126 0.126 0.126 0.126 0.126 0.126 0.126 0.125 0.125 0.125 0.125 0.125
 0.125 0.125 0.125]

The optimal number of alphas is 0 and respective accuracy over training data or cross validati
on accuracy is 91.235266%

The test accuracy or accuracy over unknown data of the  Bernoulli  Naive Bayes for optimal alp
ha = 0.010000 is 91.466667%

OBSERVATION: Able to achieve  91.47 % of test accuracy

CONFUSION METRIX :
True Negative Rate:  45.35705581281989
```

```
False Negative Rate:  1.2279414604008185
False Positive Rate:  54.642944187180106
True Positive Rate:  98.77205853959919
Precision:  0.9194133927608641
Recall:  0.9877205853959918
F1 Score:  0.9523437209129156


Model:  Bernoulli  Naive Bayes
Hyper Parameter: alpha = 0.01
Train Error:  8.76
Test Error:  8.53
```



OBSERVATIONS: Bernaulli Naive Bayes for TF-IDF

- 1) The misclassification error for each alpha value is as above in the output
- 2) The optimal alpha is 0.01 and respective accuracy over training data or cross validation accuracy is 91.235266%
- 3) The test accuracy or accuracy over unknown data of the Bernaulli Naive Bayes for optimal alpha = 0.01 is 991.466667%
- 4)** Able to achieve 91.47 % of test accuracy
- 5) **Only 45.35% of data points are predited to be negative out of all negative points. IT LOOKS LIKE OUR MODEL IS A DUMB MODEL, THIS IS BECAUSE OUR DATASET IS IMBALANCED**
- 6) Almost 1.22% of the positive data points are predictes as negative
- 7) **Almost 54.64% of data points which are actually negative and are predicted as positive by the model. IT LOOKS LIKE OUR MODEL IS A DUMB MODEL, THIS IS BECAUSE OUR DATASET IS IMBALANCED**
- 8) Almost 98.77% of data points from the test data are predicted to be positive out of all actual positive data points
- 9) For hyper parameter alpha=0.01, the train error is 8.76% and test error is 8.53%
- Precision is 0.9194 i.e. **only 91.94% of points in the test data are actually positive out of all points predicted as positive**
- Recall is 0.9877 i.e. **98.77% of data points from the test data are predicted to be positive out of all actual positive data points**

- Good F1 score 0.9523 since its close to 1.

In [39]:
```
prob_no_class=np.array(feature_log_prob[0])
prob_yes_class=np.array(feature_log_prob[1])
```

Top 20 prominent features or words which influences class label to be predicted as NEGATIVE using Bernaulli Naive Bayes for TF-IDF

In [40]:
```
showProminentWords(prob_no_class, prob_yes_class, features_count_vect_train, top_prominent)
```

	PROMINENT_FEATURES	LIKELIHOOD_OR_PROBABILITIES
744255	worst tast	-5.711775
744158	worst coffe	-6.442264
140947	complet wast	-6.522242
363619	lesson learn	-6.609178
662667	this crap	-6.609178
108515	cannot return	-6.704397
695247	unfortun was	-6.704397
275846	great disappoint	-6.704397
240188	follow all	-6.704397
585204	so horribl	-6.809647
634561	terribl disappoint	-6.809647
1719	absolut aw	-6.927291
526445	real disappoint	-6.927291
404142	misrepres	-6.927291
552611	same symptom	-6.927291
338688	it poor	-7.060644
663124	this garbag	-7.060644
658725	they refund	-7.060644
533185	refund but	-7.060644
724614	were moldi	-7.060644

Top 20 prominent features or words which influences class label to be predicted as POSITIVE using Bernaulli Naive Bayes for TF-IDF

In [41]:
```
showProminentWords(prob_yes_class, prob_no_class, features_count_vect_train, top_prominent)
```

	PROMINENT_FEATURES	LIKELIHOOD_OR_PROBABILITIES

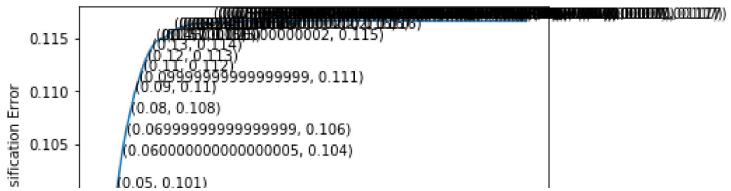
18007	also great	-5.452409
108718	cant go	-5.483062
168422	definit buy	-5.704252
212491	everyon love	-5.908153
27569	and friend	-5.969518
581990	smile	-5.969518
411709	most delici	-6.076279
275688	great choic	-6.104850
434318	not disappoint	-6.126828
487869	perfect blend	-6.156903
528730	rebecca review	-6.172287
556343	say enough	-6.211814
89964	breakfast or	-6.269917
347455	just perfect	-6.278500
521161	quick snack	-6.304700
9278	after dinner	-6.322556
505270	prefer this	-6.331605
417057	must tri	-6.349953
426896	new favorit	-6.368643
457917	onli downsid	-6.378121

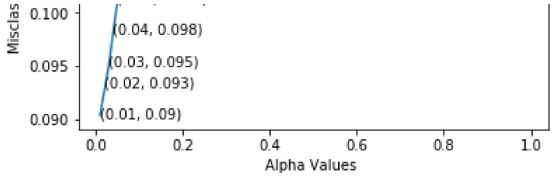
2. Multinomial Naive Bayes for TF-IDF

Optimal alpha accuracy: 10 fold cross validation Multinomial Naive Bayes for TF-IDF

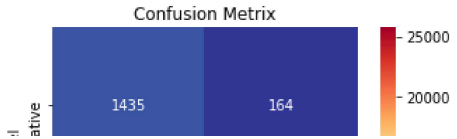
- Performing **Multinomial Naive Bayes 10 fold cross validation** over training data to get the accuracy and mis classification error
- Getting optimal alpha and with low misclassification error
- Then accuracy for test data is reported using optimal alpha

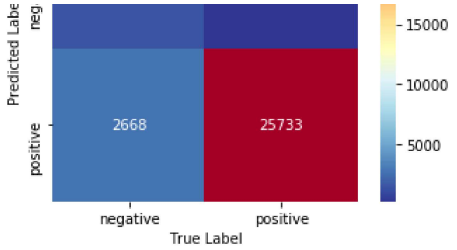
In [42]:
```
optimal_alpha,train_acc = NaiveBayes10fold_and_MSE(final_count_tf_idf_train, y_train, 'Multinom
ial')
feature_log_prob = NaiveBayes_with_optimal_alpha(final_count_tf_idf_train, y_train, final_count
s_tf_idf_test, y_test, optimal_alpha, 'Multinomial')
```






```
The misclassification error for each alpha value is :  [0.09  0.093 0.095 0.098 0.101 0.104 0.
106 0.108 0.11  0.111 0.112 0.113
 0.114 0.115 0.115 0.115 0.115 0.116 0.116 0.116 0.116 0.116 0.116 0.116
 0.116 0.116 0.117 0.117 0.117 0.117 0.117 0.117 0.117 0.117 0.117 0.117
 0.117 0.117 0.117 0.117 0.117 0.117 0.117 0.117 0.117 0.117 0.117 0.117
 0.117 0.117 0.117 0.117 0.117 0.117 0.117 0.117 0.117 0.117 0.117 0.117
 0.117 0.117 0.117 0.117 0.117 0.117 0.117 0.117 0.117 0.117 0.117 0.117
 0.117 0.117 0.117 0.117 0.117 0.117 0.117 0.117 0.117 0.117 0.117 0.117
 0.117 0.117 0.117]

The optimal number of alphas is 0 and respective accuracy over training data or cross validati
on accuracy is 90.964954%

The test accuracy or accuracy over unknown data of the  Multinomial  Naive Bayes for optimal a
lpha = 0.010000 is 90.560000%

OBSERVATION: Able to achieve  90.56 % of test accuracy

CONFUSION METRIX :
True Negative Rate:  34.97440896904704
False Negative Rate:  0.6332779858670889
False Positive Rate:  65.02559103095295
True Positive Rate:  99.36672201413292
Precision:  0.9060596457871202
Recall:  0.9936672201413291
F1 Score:  0.9478433828133633


Model:  Multinomial  Naive Bayes
Hyper Parameter: alpha = 0.01
Train Error:  9.04
Test Error:  9.44
```





OBSERVATIONS: Multinomial Naive Bayes for TF-IDF

- 1) The misclassification error for each alpha value is as above in the output
- 2) The optimal alpha is 0.01 and respective accuracy over training data or cross validation accuracy is 90.964954%
- 3) The test accuracy or accuracy over unknown data of the Bernaulli Naive Bayes for optimal alpha = 0.01 is 90.560000%
- 4)** Able to achieve 90.56% of test accuracy
- 5) **Only 34.97% of data points are predited to be negative out of all negative points. IT LOOKS LIKE OUR MODEL IS A DUMB MODEL, THIS IS BECAUSE OUR DATASET IS IMBALANCED**
- 6) Almost 0.63% of the positive data points are predictes as negative
- 7) **Almost 65.02% of data points which are actually negative and are predicted as positive by the model. IT LOOKS LIKE OUR MODEL IS A DUMB MODEL, THIS IS BECAUSE OUR DATASET IS IMBALANCED**
- 8) Almost 99.36% of data points from the test data are predicted to be positive out of all actual positive data points
- 9) For hyper parameter alpha=0.01, the train error is 9.04% and test error is 9.44%
- Precision is 0.9060 i.e. **only 90.60% of points in the test data are actually positive out of all points predicted as positive**
- Recall is 0.9936 i.e. **99.36% of data points from the test data are predicted to be positive out of all actual positive data points**
- Good F1 score 0.94 since its close to 1

In [43]:
```
prob_no_class=np.array(feature_log_prob[0])
prob_yes_class=np.array(feature_log_prob[1])
```

Top 20 prominent features or words which influences class label to be predicted as NEGATIVE using Multinomial Naive Bayes for TF-IDF

In [44]:
```
showProminentWords(prob_no_class, prob_yes_class, features_count_vect_train, top_prominent)
```

```
       PROMINENT_FEATURES  LIKELIHOOD_OR_PROBABILITIES
744255         worst tast                    -9.951882
744158        worst coffe                   -10.839117
```

363619	lesson learn	-10.939319
140947	complet wast	-11.043717
240188	follow all	-11.125058
368764	like vomit	-11.153488
275846	great disappoint	-11.171058
724614	were moldi	-11.173218
526445	real disappoint	-11.197283
108515	cannot return	-11.221167
292187	have return	-11.231375
179709	disapoint in	-11.252730
695247	unfortun was	-11.273373
634737	terribl veri	-11.274945
662667	this crap	-11.283114
585204	so horribl	-11.283648
435479	not reorder	-11.292746
426486	never reciev	-11.297643
435443	not recomend	-11.309410
663124	this garbag	-11.317149

Top 20 prominent features or words which influences class label to be predicted as POSITIVE using Multinomial Naive Bayes for TF-IDF

In [45]:
```
showProminentWords(prob_yes_class, prob_no_class, features_count_vect_train, top_prominent)
```

	PROMINENT_FEATURES	LIKELIHOOD_OR_PROBABILITIES
108718	cant go	-10.002544
18007	also great	-10.002684
168422	definit buy	-10.096145
212491	everyon love	-10.279764
411709	most delici	-10.520274
27569	and friend	-10.544394
487869	perfect blend	-10.612822
27339	and fast	-10.625620
417057	must tri	-10.633819
521161	quick snack	-10.662800
581990	smile	-10.687559
434318	not disappoint	-10.689890
347455	just perfect	-10.728525
270202	good snack	-10.748921
426896	new favorit	-10.759949
89964	breakfast or	-10.761605
556343	say enough	-10.769166
453652	on chicken	-10.791075

9278	after dinner	-10.792096
405111	mix make	-10.824099

#

Observation:

- **Its required to do upsampling or downsampling to get the acceptable model or technically the acceptable confusion metrix**

In [46]:

Did you find this Kernel useful?
Show your appreciation with an upvote

0

Comments (0)

Sort by Select...

Click here to enter a comment...