Amazon Food Reviews - [Naive Bayes]

Data Source:https://www.kaggle.com/snap/amazon-fine-food-reviews

This dataset consists of reviews of fine foods from Amazon. The data span a period of more than 10 years, including all ~500,000 reviews up to October 2012. Reviews include product and user information, ratings, and a plain text review. It also includes reviews from all other Amazon categories.

Data includes:

- Reviews from Oct 1999 - Oct 2012
- 568,454 reviews
- 256,059 users
- 74,258 products
- 260 users with > 50 reviews

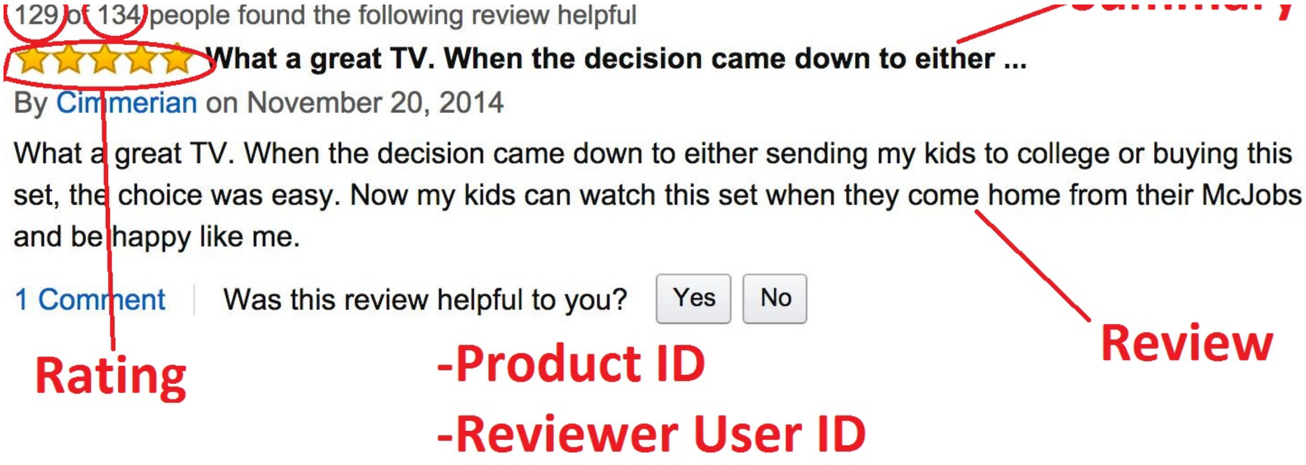
Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Number of people who found the review helpful

Number of people who indicated whether or not the review was helpful

Summary



Objective:- Review Polarity

Given a review, determine the review is positive or negative

Using text review to decide the polarity

Take the summary and text of review and analyze it using NLP whether the customer feedback/review is positive or negative

In [5]:

```
#Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sqlite3 as sql
import seaborn as sns
from time import time
import random
import gensim
import warnings
#Metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

warnings.filterwarnings("ignore")

%matplotlib inline
# sets the backend of matplotlib to the 'inline' backend:
#With this backend, the output of plotting commands is displayed inline within frontends like the
Jupyter notebook,
#directly below the code cell that produced it. The resulting plots will then also be stored in th
e notebook document.

#Functions to save objects for later use and retireve it
import pickle
def savetofile(obj,filename):
    pickle.dump(obj,open(filename+".p","wb"))
def openfromfile(filename):
    temp = pickle.load(open(filename+".p","rb"))
    return temp
```

In [28]:

```
# !wget --header="Host: e-2106e5ff6b.cognitiveclass.ai" --header="User-Agent: Mozilla/5.0 (Windows
NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/67.0.3396.99 Safari/537.36" --h
eader="Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8" --header="A
ccept-Language: en-US,en;q=0.9" --header="Cookie: _ga=GA1.2.1009651095.1527270727;
_xsrf=2|d66eb8d7|8e30b1015ec501038d0632ff567bddb6|1529904261;
session=.eJxVj9tugkAURX_FnGdi5FaBxKSgoiZKtYq0Ng0ZYIBRGBQGEY3_XjBt2r6us1f2PjdwjzhPEcWUgcbyEnOASha7Ll
oNOg3_gBoJaneSiIVh7x0UXf0wWsTyNNkZorCZKbm5Z6ZAgrAotpbZz3aC4fRn0vyqWFYujKlOp97YWF963kVdE10I-KoKzFLE
```

```
7OS9rtUzNmRnPlrOlq6a6Updnh00TspafjLEGJ18aSjtfRu4Zo0HGsD9885BgRL2GNiilX18tye70-
LNXLw4puw7vLzaWrWdxCN701OH0WAAjVQWOHcJDbPWxCkiSSPnxD_UKCCs-bLP889Ry7t-
lgIHIckL5lKU4iaoPzINTdAvnJRH1rJ_mc4PbQu_L39r2i2V55IANEWWROn-BWX4gJ4.Dh-
C7A.53fm96PBqDQvenTjy0oa1UWqE_8" --header="Connection: keep-alive" "https://e-
2106e5ff6b.cognitiveclass.ai/files/Amazon%20Fine%20Food%20Reviews%20Dataset/tfidf_w2v_vec_google.p
download=1" -O "tfidf_w2v_vec_google.p" -c
```

Loading the data

In [2]:

```
#Using sqlite3 to retrieve data from sqlite file

con = sql.connect("final.sqlite")#Loading Cleaned/ Preprocesed text that we did in Text
Preprocessing

#Using pandas functions to query from sql table
df = pd.read_sql_query("""
SELECT * FROM Reviews
""",con)

#Reviews is the name of the table given
#Taking only the data where score != 3 as score 3 will be neutral and it won't help us much
df.head()
```

Out[2]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
0	0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	Positive	13
1	1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	Negative	13
2	2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	Positive	12
3	3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	Negative	13
4	4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	Positive	13

In [3]:

```
df.describe()
```

Out[3]:

	index	Id	HelpfulnessNumerator	HelpfulnessDenominator	Time
count	364171.000000	364171.000000	364171.000000	364171.000000	3.641710e+05
mean	241825.377603	261814.561014	1.739021	2.186841	1.296135e+09
std	154519.869452	166958.768333	6.723921	7.348482	4.864772e+07
min	0.000000	1.000000	0.000000	0.000000	9.393408e+08
25%	104427.500000	113379.500000	0.000000	0.000000	1.270858e+09
50%	230033.000000	249445.000000	0.000000	1.000000	1.311379e+09
75%	376763.500000	407408.500000	2.000000	2.000000	1.332893e+09
max	525813.000000	568454.000000	866.000000	878.000000	1.351210e+09

In [4]:

```
df.shape
df['Score'].size
```

Out[4]:

```
364171
```

For EDA and Text Preprecessing Refer other ipynb notebook

In [5]:

```
#Score as positive/negative -> 0/1
def polarity(x):
    if x == "Positive":
        return 0
    else:
        return 1
df["Score"] = df["Score"].map(polarity) #Map all the scores as the function polarity i.e. positive or negative
df.head()
```

Out[5]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
0	0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	0	130
1	1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	134
2	2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	0	121
3	3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	1	130

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
4	4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	0	135

In [6]:

```
#Taking Whole Data
n_samples = 364170
df_sample = df

###Sorting as we want according to time series
df_sample.sort_values('Time',inplace=True)
df_sample.head(10)
```

Out[6]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator
117924	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0
117901	138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2
298792	417839	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	0
169281	212472	230285	B00004RYGX	A344SMIA5JECGM	Vincent P. Ross	1	2
298791	417838	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0	0
169342	212533	230348	B00004RYGX	A1048CYU0OV4O8	Judy L. Eans	2	2
169267	212458	230269	B00004RYGX	A1B2IZU1JLZA6	Wes	19	23
63317	70688	76882	B00002N8SM	A32DW342WBJ6BX	Buttersugar	0	0
169367							

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator
	212558	230376	B00004RYGX	ACJR7EQF9S6FP	Jeremy Robertson	2	3
169320	212511	230326	B00004RYGX	A2DEE7F9XKP3ZR	jerome	0	3

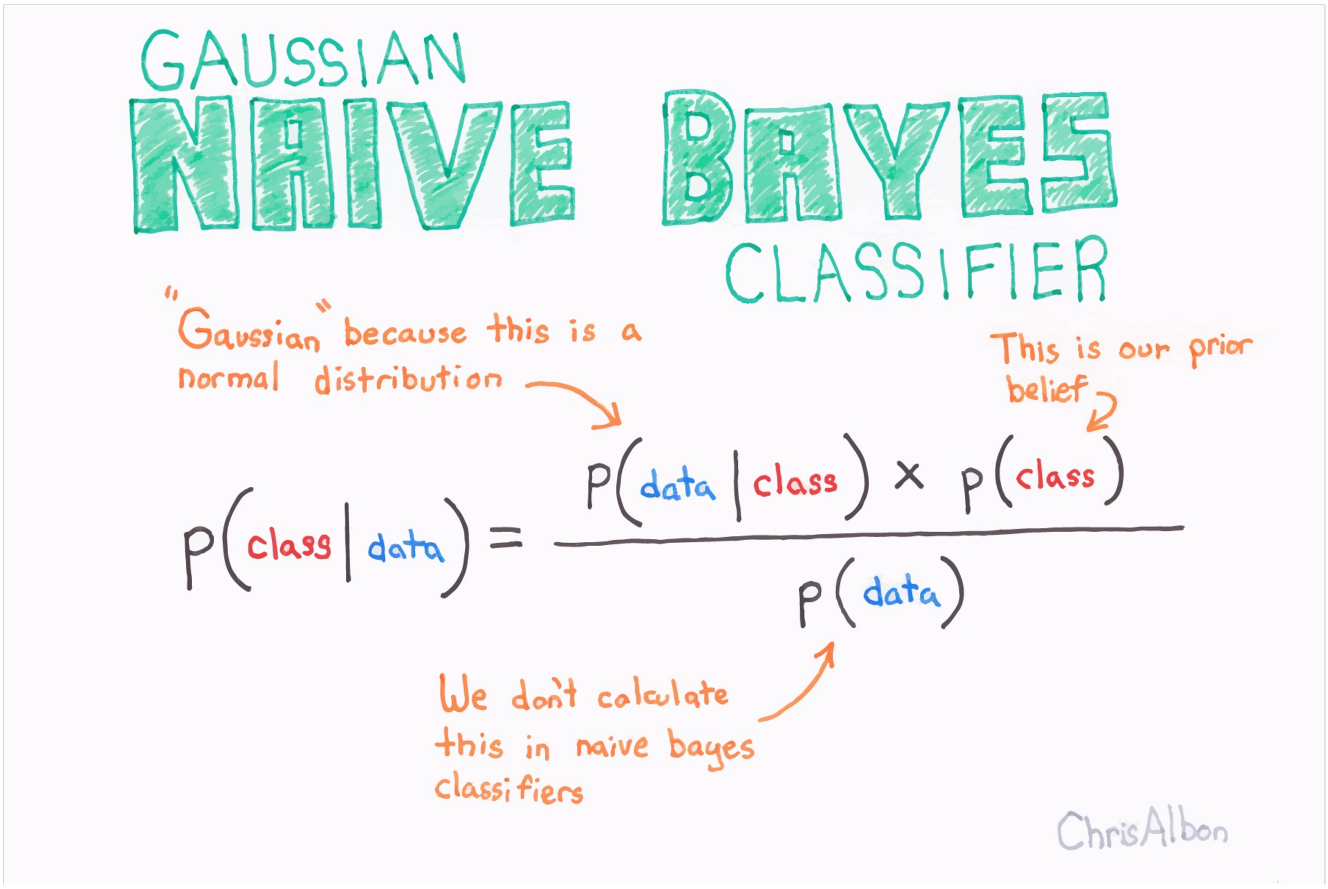
In [7]:

```
#Saving all samples in disk to as to test to test on the same sample for each of all Algo
savetofile(df_sample,"sample_nb")
```

In [8]:

```
#Opening from samples from file
df_sample = openfromfile("sample_nb")
```

Naive Bayes Model on Reviews using Different Vectorizing Techniques in NLP



Bag of Words (BoW)

A commonly used model in methods of Text Classification. As part of the BOW model, a piece of text (sentence or a document) is represented as a bag or multiset of words, disregarding grammar and even word order and the frequency or occurrence of each word is used as a feature for training a classifier.
OR
Simply,Converting a collection of text documents to a matrix of token counts

In [9]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split

#Text -> Uni gram Vectors
uni_gram = CountVectorizer() #in scikit-learn
uni_gram_vectors = uni_gram.fit_transform(df_sample['CleanedText'].values)
uni_gram_vectors.shape
```

Out[9]:

```
(364171, 209129)
```

In [10]:

```
from sklearn import preprocessing
#Normalizing the data
uni_gram_vectors_norm = preprocessing.normalize(uni_gram_vectors)
print(uni_gram_vectors_norm.min())
print(uni_gram_vectors_norm.max())
#Not shuffling the data as we want it on time basis
X_train, X_test, y_train, y_test = train_test_split(uni_gram_vectors_norm,df_sample['Score'].values
,test_size=0.3,shuffle=False)
```

```
0.0
1.0
```

In [11]:

```
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=10)
for train, cv in tscv.split(X_train):
#     print("%s %s" % (train, cv))
    print(X_train[train].shape, X_train[cv].shape)
```

```
(23179, 209129) (23174, 209129)
(46353, 209129) (23174, 209129)
(69527, 209129) (23174, 209129)
(92701, 209129) (23174, 209129)
(115875, 209129) (23174, 209129)
(139049, 209129) (23174, 209129)
(162223, 209129) (23174, 209129)
(185397, 209129) (23174, 209129)
(208571, 209129) (23174, 209129)
(231745, 209129) (23174, 209129)
```

Finding the best 'alpha' using Forward Chaining Cross Validation or Time Series CV

In [12]:

```
%time
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import BernoulliNB

bnb = BernoulliNB()
param_grid = {'alpha':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001]} #params
we need to try on classifier
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = GridSearchCV(bnb,param_grid,cv=tscv,verbose=1)
gsv.fit(X_train,y_train)
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 7.39 µs
Fitting 10 folds for each of 15 candidates, totalling 150 fits
Best HyperParameter:  {'alpha': 0.001}
Best Accuracy: 87.45%
```

```
[Parallel(n_jobs=1)]: Done 150 out of 150 | elapsed:   52.3s finished
```

With alpha=0.001 uni_gram has the highest accuracy of 87.45% in Cross Validation

In [13]:

```
#Testing Accuracy on Test data
from sklearn.naive_bayes import BernoulliNB

bnb = BernoulliNB(alpha=0.001)
bnb.fit(X_train,y_train)
y_pred = bnb.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy on test set: 87.002%
Precision on test set: 0.623
Recall on test set: 0.648
F1-Score on test set: 0.635
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[82692  7485]
 [ 6716 12359]]
```

Feature Importance[Top 25]

The coef_ attribute of MultinomialNB is a re-parameterization of the naive Bayes model as a linear classifier model. For a binary classification problems this is basically the log of the estimated probability of a feature given the positive class. It means that higher values mean more important features for the positive class.

In [14]:

```
def show_most_informative_features(vectorizer, clf, n=25):
    feature_names = vectorizer.get_feature_names()
    coefs_with_fns = sorted(zip(clf.coef_[0], feature_names))
    top = zip(coefs_with_fns[:n], coefs_with_fns[:-(n + 1):-1])
    print("\t\t\tPositive\t\t\t\t\t\tNegative")

print("________________________________________________________________________________________________
")
    for (coef_1, fn_1), (coef_2, fn_2) in top:
        print("\t%.4f\t%-15s\t\t\t\t%.4f\t%-15s" % (coef_1, fn_1, coef_2, fn_2))

show_most_informative_features(uni_gram,bnb)
#Code Reference:https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-for-scikit-learn-classifiers
```

```
   Positive      Negative
________________________________________________________________________________________________
 -17.4540 aaaa                  -0.6000 not
 -17.4540 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaseri    -0.9927 tast
 -17.4540 aaaaaaaaaaaaaaaaaaaargh    -1.0046 like
 -17.4540 aaaaaaaaaagghh        -1.2246 product
 -17.4540 aaaaaaaah             -1.3641 one
 -17.4540 aaaaaaahhhhhh         -1.4248 would
 -17.4540 aaaaaah               -1.4770 tri
 -17.4540 aaaaaahhh             -1.5611 flavor
 -17.4540 aaaaaahhhhh           -1.5682 good
 -17.4540 aaaaaahhhhten         -1.6188 buy
 -17.4540 aaaaaawwwwwwwwww       -1.6710 get
 -17.4540 aaaaah                -1.7154 use
 -17.4540 aaaaahhhhhhhhhhhhhhhhth    -1.7735 dont
 -17.4540 aaaaallll             -1.8378 even
```

```
 -17.4540 aaaaawher           -1.8560 order
 -17.4540 aaaaawsom           -1.9746 much
 -17.4540 aaaah               -1.9772 make
 -17.4540 aaaahhhhhh          -2.0297 realli
 -17.4540 aaaallll            -2.0442 time
 -17.4540 aaaand              -2.0579 love
 -17.4540 aaaannnndddgolazo     -2.0946 look
 -17.4540 aaaarrrrghh         -2.1057 amazon
 -17.4540 aaagh               -2.1092 eat
 -17.4540 aaah                -2.1135 bought
 -17.4540 aaahhh              -2.1186 box
```

bi-gram

In [16]:

```
from sklearn.feature_extraction.text import CountVectorizer
#taking one words and two consecutive words together
bi_gram = CountVectorizer(ngram_range=(1,2))
bi_gram_vectors = bi_gram.fit_transform(df_sample['CleanedText'].values)
bi_gram_vectors.shape
```

Out[16]:

```
(364171, 3404647)
```

In [17]:

```
from sklearn import preprocessing
bi_gram_vectors_norm = preprocessing.normalize(bi_gram_vectors)
```

In [18]:

```
from sklearn.model_selection import train_test_split
#Not shuffling the data as we want it on time basis
X_train, X_test, y_train, y_test = train_test_split(bi_gram_vectors_norm,df_sample['Score'].values,
test_size=0.3,shuffle=False)
```

In [19]:

```
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=10)
for train, cv in tscv.split(X_train):
#     print("%s %s" % (train, cv))
    print(X_train[train].shape, X_train[cv].shape)
```

```
(23179, 3404647) (23174, 3404647)
(46353, 3404647) (23174, 3404647)
(69527, 3404647) (23174, 3404647)
(92701, 3404647) (23174, 3404647)
(115875, 3404647) (23174, 3404647)
(139049, 3404647) (23174, 3404647)
(162223, 3404647) (23174, 3404647)
(185397, 3404647) (23174, 3404647)
(208571, 3404647) (23174, 3404647)
(231745, 3404647) (23174, 3404647)
```

In [19]:

```
%time
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import BernoulliNB

bnb = BernoulliNB()
param_grid = {'alpha':[1000,500,100,50,10,7,6,5,4,2,1,0.5,0.1,0.05,0.01,0.005,0.001]} #params we ne
ed to try on classifier
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = GridSearchCV(bnb,param_grid,cv=tscv,verbose=1)
gsv.fit(X_train,y_train)
print("Best HyperParameter: ",gsv.best_params_)
```

```
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 7.39 µs
Fitting 10 folds for each of 17 candidates, totalling 170 fits
```

```
[Parallel(n_jobs=1)]: Done 170 out of 170 | elapsed:  6.2min finished
```

```
Best HyperParameter:  {'alpha': 0.001}
Best Accuracy: 88.96%
```

With alpha=0.001 bi_gram has the highest accuracy of 88.96% in Cross Validation

In [20]:

```
#Testing Accuracy on Test data
from sklearn.naive_bayes import BernoulliNB

bnb = BernoulliNB(alpha=0.001)
bnb.fit(X_train,y_train)
y_pred = bnb.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy on test set: 89.282%
Precision on test set: 0.741
Recall on test set: 0.594
F1-Score on test set: 0.659
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[86215  3962]
 [ 7748 11327]]
```

Feature Importance[Top 25]

In [22]:

```
def show_most_informative_features(vectorizer, clf, n=25):
    feature_names = vectorizer.get_feature_names()
    coefs_with_fns = sorted(zip(clf.coef_[0], feature_names))
    top = zip(coefs_with_fns[:n], coefs_with_fns[:-(n + 1):-1])
    print("\t\t\tPositive\t\t\t\t\t\tNegative")

print("________________________________________________________________________________________________
")
    for (coef_1, fn_1), (coef_2, fn_2) in top:
        print("\t%.4f\t%-15s\t\t\t\t%.4f\t%-15s" % (coef_1, fn_1, coef_2, fn_2))

show_most_informative_features(bi_gram,bnb)
#Code Reference:https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-for-scikit-learn-classifiers
```

```
   Positive       Negative
________________________________________________________________________________________________
 -17.4540 aa pleas          -0.6000 not
 -17.4540 aa state          -0.9927 tast
 -17.4540 aaa aaa           -1.0046 like
 -17.4540 aaa class         -1.2246 product
 -17.4540 aaa condit        -1.3641 one
 -17.4540 aaa hockey        -1.4248 would
 -17.4540 aaa job           -1.4770 tri
 -17.4540 aaa magazin       -1.5611 flavor
 -17.4540 aaa perfect       -1.5682 good
 -17.4540 aaa plus          -1.6188 buy
 -17.4540 aaa rate          -1.6710 get
```

```
 -17.4540 aaa spelt          -1.7154 use
 -17.4540 aaa tue            -1.7735 dont
 -17.4540 aaaa               -1.8378 even
 -17.4540 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaseri    -1.8560 order
 -17.4540 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaseri good    -1.9746 much
 -17.4540 aaaaaaaaaaaaaaaaaaaaaargh    -1.9772 make
 -17.4540 aaaaaaaaaaaaaaaaaaaaaargh wait    -2.0297 realli
 -17.4540 aaaaaaaaagghh      -2.0442 time
 -17.4540 aaaaaaah           -2.0579 love
 -17.4540 aaaaaaah good      -2.0946 look
 -17.4540 aaaaaaahhhhhh      -2.1057 amazon
 -17.4540 aaaaaaahhhhhh raspberri    -2.1092 eat
 -17.4540 aaaaaah            -2.1135 bought
 -17.4540 aaaaaah melt       -2.1186 box
```

tf-idf

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

In [23]:

```
%%time
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(ngram_range=(1,2)) #Using bi-grams
tfidf_vec = tfidf.fit_transform(df_sample['CleanedText'].values)
tfidf_vec.shape
```

```
CPU times: user 1min 1s, sys: 624 ms, total: 1min 2s
Wall time: 1min 2s
```

In [24]:

```
tfidf_vec.shape
```

Out[24]:

```
(364171, 3404647)
```

In [25]:

```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

tfidf_vec_norm = preprocessing.normalize(tfidf_vec)

#Not shuffling the data as we want it on time basis
X_train, X_test, y_train, y_test = train_test_split(tfidf_vec_norm,df_sample['Score'].values,test_s
ize=0.3,shuffle=False)
```

In [25]:

```
%time
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import BernoulliNB

bnb = BernoulliNB()
param_grid = {'alpha':[1000,500,100,50,10,7,6,5,4,2,1,0.5,0.1,0.05,0.01,0.005,0.001]} #params we ne
ed to try on classifier
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = GridSearchCV(bnb,param_grid,cv=tscv,verbose=1)
gsv.fit(X_train,y_train)
```

```
gsv.fit(X_train,y_train)
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 7.63 µs
Fitting 10 folds for each of 17 candidates, totalling 170 fits

[Parallel(n_jobs=1)]: Done 170 out of 170 | elapsed:  6.1min finished

Best HyperParameter:  {'alpha': 0.001}
Best Accuracy: 88.96%
```

With alpha=0.001 tf-idf has the highest accuracy of 88.96% in Cross Validation

In [26]:

```
#Testing Accuracy on Test data
from sklearn.naive_bayes import BernoulliNB

bnb = BernoulliNB(alpha=0.001)
bnb.fit(X_train,y_train)
y_pred = bnb.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy on test set: 89.282%
Precision on test set: 0.741
Recall on test set: 0.594
F1-Score on test set: 0.659
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[86215  3962]
 [ 7748 11327]]
```

Feature Importance[Top 25]

In [27]:

```
def show_most_informative_features(vectorizer, clf, n=25):
    feature_names = vectorizer.get_feature_names()
    coefs_with_fns = sorted(zip(clf.coef_[0], feature_names))
    top = zip(coefs_with_fns[:n], coefs_with_fns[:-(n + 1):-1])
    print("\t\t\tPositive\t\t\t\t\t\tNegative")
print("________________________________________________________________________________________________
")
    for (coef_1, fn_1), (coef_2, fn_2) in top:
        print("\t%.4f\t%-15s\t\t\t\t%.4f\t%-15s" % (coef_1, fn_1, coef_2, fn_2))

show_most_informative_features(tfidf,bnb)
#Code Reference:https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-for-scikit-learn-classifiers
```

```
   Positive       Negative
_______________________________________________________________________________________________
 -17.4540 aa pleas            -0.6000 not
 -17.4540 aa state            -0.9927 tast
 -17.4540 aaa aaa             -1.0046 like
 -17.4540 aaa class           -1.2246 product
 -17.4540 aaa condit          -1.3641 one
 -17.4540 aaa hockey          -1.4248 would
 -17.4540 aaa job             -1.4770 tri
 -17.4540 aaa magazin         -1.5611 flavor
 -17.4540 aaa perfect         -1.5682 good
```

```
-17.4540 aaa plus            -1.6188 buy
-17.4540 aaa rate            -1.6710 get
-17.4540 aaa spelt           -1.7154 use
-17.4540 aaa tue             -1.7735 dont
-17.4540 aaaa                -1.8378 even
-17.4540 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaseri    -1.8560 order
-17.4540 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaseri good    -1.9746 much
-17.4540 aaaaaaaaaaaaaaaaaaaaaargh    -1.9772 make
-17.4540 aaaaaaaaaaaaaaaaaaaaaargh wait    -2.0297 realli
-17.4540 aaaaaaaaagghh       -2.0442 time
-17.4540 aaaaaaah            -2.0579 love
-17.4540 aaaaaaah good       -2.0946 look
-17.4540 aaaaaaahhhhhh       -2.1057 amazon
-17.4540 aaaaaaahhhhhh raspberri    -2.1092 eat
-17.4540 aaaaaah             -2.1135 bought
-17.4540 aaaaaah melt        -2.1186 box
```

Gensim

Gensim is a robust open-source vector space modeling and topic modeling toolkit implemented in Python. It uses NumPy, SciPy and optionally Cython for performance. Gensim is specifically designed to handle large text collections, using data streaming and efficient incremental algorithms, which differentiates it from most other scientific software packages that only target batch and in-memory processing.

Word2Vec

[Refer Docs] :https://radimrehurek.com/gensim/models/word2vec.html

In [4]:

```python
from gensim.models import KeyedVectors

#Loading the model from file in the disk
w2vec_model = KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True)
```

In [25]:

```python
w2v_vocub = w2vec_model.wv.vocab
len(w2v_vocub)
```

Out[25]:

```
3000000
```

Avg Word2Vec

- **One of the most naive but good ways to convert a sentence into a vector**
- **Convert all the words to vectors and then just take the avg of the vectors the resulting vector represent the sentence**

In [30]:

```python
%%time
avg_vec_google = [] #List to store all the avg w2vec's
# no_datapoints = 364170
# sample_cols = random.sample(range(1, no_datapoints), 20001)
for sent in df_sample['CleanedText_NoStem']:
    cnt = 0 #to count no of words in each reviews
    sent_vec = np.zeros(300) #Initializing with zeroes
#     print("sent:",sent)
    sent = sent.decode("utf-8")
    for word in sent.split():
        try:
#             print(word)
            wvec = w2vec_model.wv[word] #Vector of each using w2v model
#             print("wvec:",wvec)
            sent_vec += wvec #Adding the vectors
```

```
#             print("sent_vec:",sent_vec)
            cnt += 1
        except:
            pass #When the word is not in the dictionary then do nothing
#     print(sent_vec)
    sent_vec /= cnt #Taking average of vectors sum of the particular review
#     print("avg_vec:",sent_vec)
    avg_vec_google.append(sent_vec) #Storing the avg w2vec's for each review
#     print("*******************************************************************")
# print(avg_vec_google)
avg_vec_google = np.array(avg_vec_google)
```

```
CPU times: user 3min 13s, sys: 548 ms, total: 3min 13s
Wall time: 3min 13s
```

In [31]:

```
np.isnan(avg_vec_google).any()
```

Out[31]:

```
True
```

In [32]:

```
mask = ~np.any(np.isnan(avg_vec_google), axis=1)
# print(mask)
avg_vec_google_new = avg_vec_google[mask]
df_sample_new = df_sample['Score'][mask]
print(avg_vec_google_new.shape)
print(df_sample_new.shape)
```

```
(364167, 300)
(364167,)
```

In [33]:

```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

avg_vec_norm = preprocessing.normalize(avg_vec_google_new)

#Not shuffling the data as we want it on time basis
X_train, X_test, y_train, y_test = train_test_split(avg_vec_norm,df_sample_new.values,test_size=0.3
,shuffle=False)
```

In [34]:

```
%time
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import BernoulliNB

bnb = BernoulliNB()
param_grid = {'alpha':[100000,75000,50000,25000,10000,7500,5000,2500,1000,500,100,50,10,7,6,5,4,2,1
,0.5,0.1,0.05,0.01,0.005,0.001]} #params we need to try on classifier
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = GridSearchCV(bnb,param_grid,cv=tscv,verbose=1)
gsv.fit(X_train,y_train)
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 7.87 µs
Fitting 10 folds for each of 25 candidates, totalling 250 fits
```

```
[Parallel(n_jobs=1)]: Done 250 out of 250 | elapsed:  7.1min finished
```

```
Best HyperParameter:  {'alpha': 100000}
Best Accuracy: 84.68%
```

With alpha=100000 Avg W2Vec has the highest accuracy of 84.68% in Cross Validation

In [35]:

```
#Testing Accuracy on Test data
from sklearn.naive_bayes import BernoulliNB

bnb = BernoulliNB(alpha=100000)
bnb.fit(X_train,y_train)
y_pred = bnb.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy on test set: 82.533%
Precision on test set: 0.250
Recall on test set: 0.000
F1-Score on test set: 0.000
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[90164    12]
 [19071     4]]
```

Feature Importance[Top 25]

In [36]:

```
def show_most_informative_features(vectorizer, clf, n=25):
    feature_names = vectorizer.get_feature_names()
    coefs_with_fns = sorted(zip(clf.coef_[0], feature_names))
    top = zip(coefs_with_fns[:n], coefs_with_fns[:-(n + 1):-1])
    print("\t\t\tPositive\t\t\t\t\t\tNegative")

print("_______________________________________________________________________________________________
")
    for (coef_1, fn_1), (coef_2, fn_2) in top:
        print("\t%.4f\t%-15s\t\t\t\t%.4f\t%-15s" % (coef_1, fn_1, coef_2, fn_2))

show_most_informative_features(uni_gram,bnb)
#Code Reference:https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-
for-scikit-learn-classifiers
```

```
   Positive       Negative
_______________________________________________________________________________________________
 -0.8672 abando               -0.5451 abbazabba
 -0.8671 aaaaaahhhhh          -0.5458 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaseri
 -0.8669 abomin               -0.5463 aagreen
 -0.8668 abnormalitiesmi      -0.5463 aaaaaah
 -0.8665 aaaaaawwwwwwwwww      -0.5468 aaah
 -0.8661 ablsolut             -0.5471 aaaaah
 -0.8647 abolit               -0.5472 abovea
 -0.8644 aardvark             -0.5475 abov
 -0.8641 abita                -0.5476 aaron
 -0.8631 abecaus              -0.5476 aaaaallll
 -0.8627 abouttwin            -0.5479 aboutstil
 -0.8627 aahhh                -0.5482 abor
 -0.8627 abondanza            -0.5483 aberdeen
 -0.8624 aboveanim            -0.5484 aboutpamela
 -0.8623 aborio               -0.5485 aback
 -0.8621 aaaaaaahhhhhh        -0.5488 abour
 -0.8621 aboutamazoncom       -0.5492 aboutfor
 -0.8617 aarp                 -0.5497 aafcoa
 -0.8616 aboutand             -0.5510 ableto
 -0.8615 abosout              -0.5518 abovecfh
 -0.8613 abbreviatedserv      -0.5526 aaaaaaah
 -0.8608 abnd                 -0.5527 aboutov
 -0.8600 abduct               -0.5543 abouthi
```

```
-0.8595 abovedcp            -0.5551 abilityyet
-0.8593 abovehappi          -0.5552 abouteveryth
```

Tf-idf W2Vec

- **Another way to covert sentence into vectors**
- **Take weighted sum of the vectors divided by the sum of all the tfidf's i.e. (tfidf(word) x w2v(word))/sum(tfidf's)**

In [11]:

```
%%time
#Taking Sample Data as it was taking more that 10 hours to computer this block
n_samples = 25000
df_sample_new = df_sample.sample(n_samples)

###Sorting as we want according to time series
df_sample_new.sort_values('Time',inplace=True)

###tf-idf with No Stemming
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(ngram_range=(1,2)) #Using bi-grams

tfidf_vec_new = tfidf.fit_transform(df_sample_new['CleanedText_NoStem'].values)

print(tfidf_vec_new.shape)

# tf-idf came up with 2.9 million features for the data corpus
# from sklearn.decomposition import TruncatedSVD

# tsvd_tfidf_ns = TruncatedSVD(n_components=300)#No of components as total dimensions
# tsvd_tfidf_vec_ns = tsvd_tfidf_ns.fit_transform(tfidf_vec_ns)
# print(tsvd_tfidf_ns.explained_variance_ratio_[:].sum())
features = tfidf.get_feature_names()
```

```
(25000, 589499)
CPU times: user 6.15 s, sys: 16 ms, total: 6.16 s
Wall time: 6.16 s
```

In [ ]:

```
%%time
tfidf_w2v_vec_google = []
review = 0

for sent in df_sample_new['CleanedText_NoStem'].values:
    cnt = 0
    weighted_sum  = 0
    sent_vec = np.zeros(300)
    sent = sent.decode("utf-8")
    for word in sent.split():
        try:
#             print(word)
            wvec = w2vec_model.wv[word] #Vector of each using w2v model
#             print("w2vec:",wvec)
#             print("tfidf:",tfidf_vec_ns[review,features.index(word)])
            tfidf_vec = tfidf_vec_new[review,features.index(word)]
            sent_vec += (wvec * tfidf_vec)
            weighted_sum += tfidf_vec
        except:
#             print(review)
            pass
    sent_vec /= weighted_sum
#     print(sent_vec)
    tfidf_w2v_vec_google.append(sent_vec)
    review += 1
tfidf_w2v_vec_google = np.array(tfidf_w2v_vec_google)
savetofile(tfidf_w2v_vec_google,"tfidf_w2v_vec_google")
```

In [6]:

```
#Precomputed File
```

```
tfidf_w2v_vec_google = openfromfile("tfidf_w2v_vec_google")
```

In [12]:

```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

tfidfw2v_vecs_norm = preprocessing.normalize(tfidf_w2v_vec_google)

#Not shuffling the data as we want it on time basis
X_train, X_test, y_train, y_test = train_test_split(tfidfw2v_vecs_norm,df_sample_new['Score'].value
s,test_size=0.3,shuffle=False)
```

In [14]:

```
%time
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import TimeSeriesSplit

bnb = BernoulliNB()
param_grid = {'alpha':[100000,75000,50000,25000,10000,7500,5000,2500,1000,500,100,50,10,7,6,5,4,2,1
,0.5,0.1,0.05,0.01,0.005,0.001]} #params we need to try on classifier
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = GridSearchCV(bnb,param_grid,cv=tscv,verbose=1)
gsv.fit(X_train,y_train)
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 8.58 µs
Fitting 10 folds for each of 25 candidates, totalling 250 fits
Best HyperParameter:  {'alpha': 5000}
Best Accuracy: 84.37%
```

```
[Parallel(n_jobs=1)]: Done 250 out of 250 | elapsed:   28.7s finished
```

With alpha=5000 TF-IDF W2Vec has the highest accuracy of 84.37% in Cross Validation

In [15]:

```
#Testing Accuracy on Test data
from sklearn.naive_bayes import BernoulliNB

bnb = BernoulliNB(alpha=5000)
bnb.fit(X_train,y_train)
y_pred = bnb.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy on test set: 82.320%
Precision on test set: 0.000
Recall on test set: 0.000
F1-Score on test set: 0.000
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[6174    2]
 [1324    0]]
```

Conclusions

1. **The best thing about Naive Bayes it much quicker than algorithms amazingly fast training times**
2. **Best Models are Bi-Gram and Tf-IDF Model with accuracy of 89.28% and precision of 0.741**

2. Best Models are Bi-Gram and TF-IDF Model with accuracy of [illegible]% and precision of [illegible]