Sentimentic analysis on Amazon Fine Food Reviews Dataset using NLTK methodology, naive Bayes, Logistic Regression and metrics like roc_curve and auc value, confusion_matrix and and classification_report

```
# Amazon Fine Food Reviews Analysis
Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-
foods-visualization/

The Amazon Fine Food Reviews dataset consists of reviews of fine
foods from Amazon.

Number of reviews: 568,454
Number of users: 256,059
Number of products: 74,258
Timespan: Oct 1999 - Oct 2012
Number of Attributes/Columns in data: 10

Attribute Information:

Id
ProductId - unique identifier for the product
UserId - unqiue identifier for the user
ProfileName
HelpfulnessNumerator - number of users who found the review helpful
HelpfulnessDenominator - number of users who indicated whether they
found the review helpful or not
Score - rating between 1 and 5
Time - timestamp for the review
Summary - brief summary of the review
Text - text of the review
Objective:
Given a review, determine whether the review is positive (Rating of
4 or 5) or negative (rating of 1 or 2).


[Q] How to determine if a review is positive or negative?

[Ans] We could use the Score/Rating. A rating of 4 or 5 could be
cosnidered a positive review. A review of 1 or 2 could be considere
negative. A review of 3 is nuetral and ignored. This is an
approximate and proxy way of determining the polarity
(positivity/negativity) of a review
```

In [5]:
```
# Loading the data
# The dataset is available in two forms

# .csv file
# SQLite Database
```

```
# In order to load the data, We have used the SQLITE dataset as it
# Here as we only want to get the global sentiment of the recommend
```

In [6]:
```
#The purpose of this analysis is to make up a prediction model wher
# In order to load the data, I will make use of sqlite3 package whe
```

In [7]:
```
%matplotlib inline
# import sqlite3
import pandas as pd
import numpy as np
import string
import nltk
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
# from sklearn.cross_validation import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

con=sqlite3.connect('database.sqlite')
messages=pd.read_sql_query(""" SELECT Score, Summary from Reviews w
messages.head()
```

Out[7]:

	Score	Summary
0	5	Good Quality Dog Food
1	1	Not as Advertised
2	4	"Delight" says it all
3	2	Cough Medicine
4	5	Great taffy

Changing the score into positive and negative recommendation with a score < 3 goes in the negative sentiment list and score >= 3 goes in the positive sentiment list.

In [8]:
```
def partition(x):
    if x < 3:
        return 'negative'
    return 'positive'

Score=messages['Score']
Score= Score.map(partition)
Summary=messages['Summary']
X_train, X_test, y_train, y_test = train_test_split(Summary, Score
```

In [9]:
```
tmp=messages
tmp['Score']=tmp['Score'].map(partition)
tmp.head()
```

Out[9]:

	Score	Summary
0	positive	Good Quality Dog Food
1	negative	Not as Advertised
2	positive	"Delight" says it all

	Score	Summary
3	negative	Cough Medicine

```
Defining and making use of some nltk packages which include
PorterStemmer(), word_tokenize().
Making use of sting package to remove punctuation from text data
```

In [10]:
```
stemmer=PorterStemmer()
def stem_tokens(token, stemmer):
    stemmed=[]
    for item in token:
        stemmed.append(stemmer.stem(item))
    return stemmed
```

In [11]:
```
def tokenize(text):
    token=nltk.word_tokenize(text)
    stems=stem_tokens(token,stemmer)
    return ' '.join(stems)
```

In [12]:
```
intab = string.punctuation
outtab = "                                "
trantab = str.maketrans(intab, outtab)
```

```
Different processes are applied on text in X_train and X_test data
Changing the upper case letter to the lower case.
Making use of maketrans to remove punctuation.
Tokenizing the text data with nltk's word_tokenize package.
Finally, when the text data is ready, applying CountVectorizer() an
Tfidf transformer() on them
```

In [13]:
```
import nltk
nltk.download('punkt')
corpus=[]
for text in X_train:
    text=text.lower()
    text=text.translate(trantab)
    text=tokenize(text)
    corpus.append(text)

count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(corpus)

tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
```

```
[nltk_data] Downloading package punkt to /home/sunil/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

Same procedure is applied on the test data.

In [14]:
```
test_set=[]
for text in X_test:
    text=text.lower()
    text=text.translate(trantab)
    text=tokenize(text)
    test_set.append(text)

X_new_counts = count_vect.transform(test_set)
X_test_tfidf = tfidf_transformer.transform(X_new_counts)
```

The changes before and after all the preprocess and nltk techniques.

In [15]:
```
df=pd.DataFrame({'Before': X_train, 'After': corpus})
df.head()
```

Out[15]:

	Before	After
496497	ALMONDS GREAT BUY	almond great buy
225396	I never thought i'd have to say no to more fru...	i never thought i d have to say no to more fru...
288197	We love this Stuff	we love thi stuff
88450	Fan-friggen-tastic	fan friggen tastic
354669	Great for office	great for offic

```
Defining a dictionary predictors which will contain the predicted
score for all the rows and for all the machine learning models.
Applying MultinomialNB from sklearn.naive_bayes:
How MultinomialNB works: "it counts how often word occurs in the
data"
```

In [16]:
```
predictors={}
from sklearn.naive_bayes import MultinomialNB
model=MultinomialNB().fit(X_train_tfidf, y_train)
predictors['Multinomial']= model.predict(X_test_tfidf)
```

```
Applying BernoulliNB from sklearn.naive_bayes:
Here Bernoulli would be applied as a text classification with bag o
words model where this model checks where that "specific word
occurred in the document" or not
```

In [17]:
```
from sklearn.naive_bayes import BernoulliNB
model=BernoulliNB().fit(X_train_tfidf, y_train)
predictors['Bernoulli']= model.predict(X_test_tfidf)
```

```
Applying LogisticRegression:
```

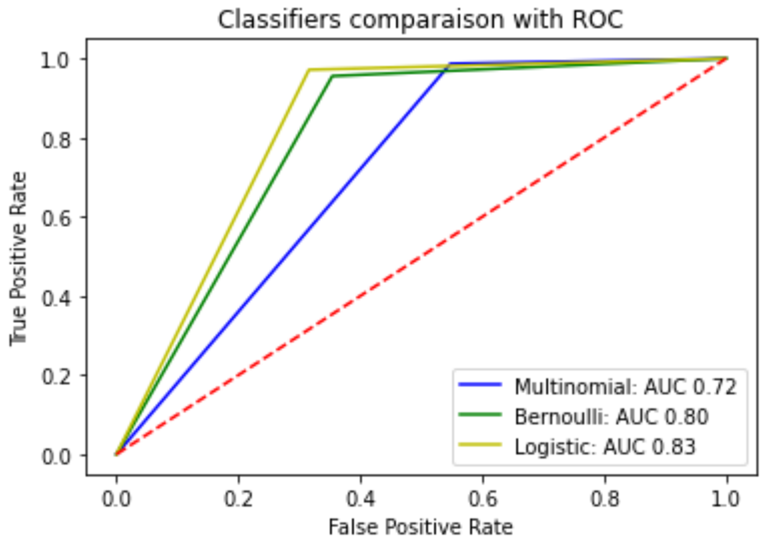
In [18]:
```
from sklearn import linear_model
logreg = linear_model.LogisticRegression(C=1e6)
logreg.fit(X_train_tfidf, y_train)
predictors['Logistic'] = logreg.predict(X_test_tfidf)
```

```
/home/sunil/anaconda3/lib/python3.8/site-packages/sklearn/linear_m
odel/_logistic.py:762: ConvergenceWarning: lbfgs failed to converg
e (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (ht
tps://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver opti
```

```
To evaluate the performance of above models, I'm making use of
roc_curve and auc from sklearn.metrics package.
With some exciting use of matplotlib library, I can actually
represent this metrics model with a nice plot as shown in the outpu
down.
The curve with highest AUC value will show our best algorithm
```

In [19]:

```
def formatt(x):
    if x == 'negative':
        return 0
    return 1
vfunc = np.vectorize(formatt)

cmp = 0
colors = ['b', 'g', 'y', 'm', 'k']
for model, predicted in predictors.items():
    false_positive_rate, true_positive_rate, thresholds = roc_curve
    roc_auc = auc(false_positive_rate, true_positive_rate)
    plt.plot(false_positive_rate, true_positive_rate, colors[cmp],
    cmp += 1

plt.title('Classifiers comparaison with ROC')
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Understanding the model result with classification_report and confusion_matrix

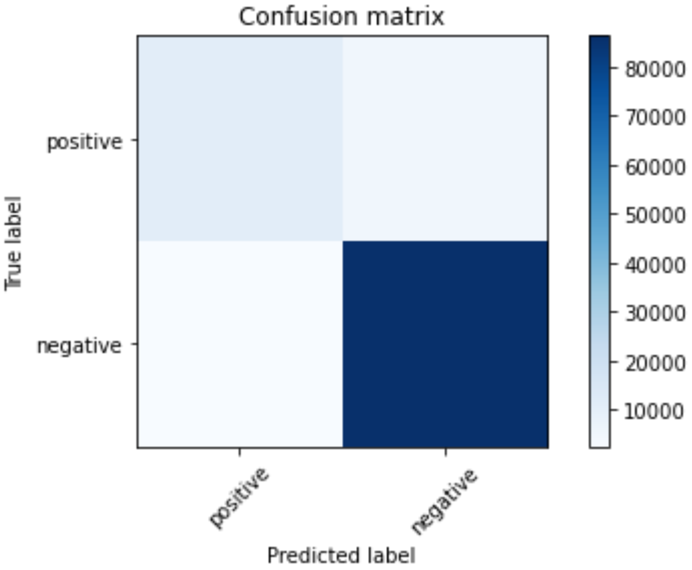
In [20]:

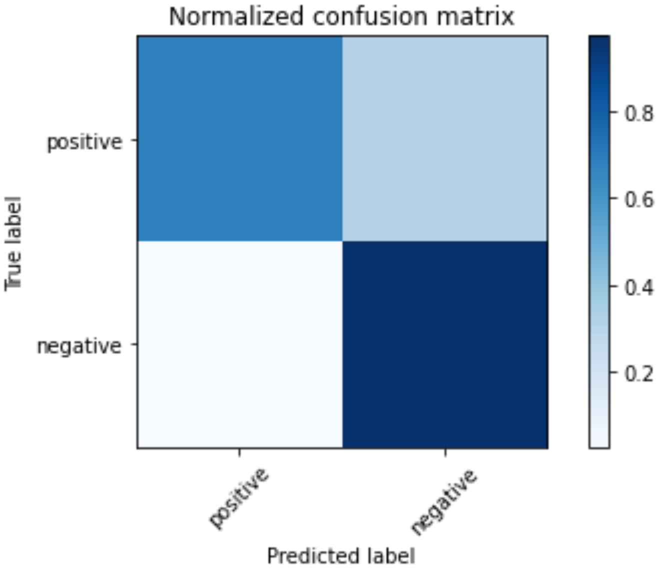
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [21]:
```
print(classification_report(y_test, predictors['Logistic']))
```

```
              precision    recall  f1-score   support

    negative       0.82      0.68      0.74     16379
    positive       0.94      0.97      0.96     88784

    accuracy                           0.93    105163
   macro avg       0.88      0.83      0.85    105163
weighted avg       0.92      0.93      0.92    105163
```

In [22]:
```
def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(set(Score)))
    plt.xticks(tick_marks, set(Score), rotation=45)
    plt.yticks(tick_marks, set(Score))
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Compute confusion matrix
cm = confusion_matrix(y_test, predictors['Logistic'])
np.set_printoptions(precision=2)
plt.figure()
plot_confusion_matrix(cm)

cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
plt.figure()
plot_confusion_matrix(cm_normalized, title='Normalized confusion ma

plt.show()
print(y_test)
```





```
126221    positive
481339    positive
202590    positive
435819    positive
            ...
251254    positive
438335    positive
14154     positive
203200    negative
260344    positive
Name: Score, Length: 105163, dtype: object
```

In [ ]: