Amazon review

Objective :

Given a review determine wheather the review is positive (4 or 5) or negative (1 or 2) ?

we can achieve this by using score(an atribute in dataset). A score rating of 4 or 5 indicates a positive review and a score of 1 or 2 indicates a negative review. rating of 3 is neutral and is ignored.

The dataset contains two files 1 review.csv(sample of 10 points that repliates the database) 2 database.sqliteThe database contains columns as id,productId,UserId,ProfileName,HelpfulnessNumerator,HelpfulnessDenominator,Score,Time(UNIX/epoch),Summary,Text.1. we will process the data i.e., deduplication and text processing and sort the data according to productId, Time 2. Now we will take the random sample of 10k points , and will use simle cross validation to find the optimal k- value. 3. Now , we apply different text processing techniques like BOW , TF-IDF , W2V , TF-IDF W2V . and for each of the technique we will aplly the k-nn

In [2]:

```
#......loading the data.............
import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sbn
from sklearn import metrics
from sklearn.metrics import f1_score
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.metrics import roc_curve,auc
from nltk.stem.porter import PorterStemmer
#....for string
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
import re

#.....using sqlite for connecting and reading the data.......
con = sqlite3.connect(r"F:\mlpractise\amazon-fine-food-reviews\database.sqlite")

#.....filtering only positive reviews and negative reviews and ignoring reviews with rating =
3......
filtered_data = pd.read_sql_query('''Select * From Reviews Where Score!=3''',con)

#.....labeling reviews with rating>3 a positive reviews and reivews with rating<3 as negative u
sing the method "partition".......
def partition(x):
    if x<3:
        return'negative'
    return'positive'

#.....now changing the numerical values into positive or negative ratings....
actualscore = filtered_data['Score']
positivenegative = actualscore.map(partition)
filtered_data['Score'] = positivenegative
filtered_data.shape
```

Out[2]:

```
(525814, 10)
```

DATA DEDUPLICATION:

In [3]:

```
#...sorting the data according to productId in Ascending order
sorted_data = filtered_data.sort_values('ProductId',axis=0,ascending=True,inplace=False,kind='q
uicksort',na_position='last')
```

In [4]:

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

Out[4]:

```
(364173, 10)
```

In [5]:

```
#...makin sure that HelpfulnessDenominator is greater than HelpfulnessNumerator
final=final[final.HelpfulnessNumerator <= final.HelpfulnessDenominator]
final['Score'].value_counts()
```

Out[5]:

```
positive    307061
negative     57110
Name: Score, dtype: int64
```

In [6]:

```
final.head(2)
```

Out[6]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpful
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0
138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	1

In [7]:

```
a = final["Text"]
a[1]
```

Out[7]:

```
'Product arrived labeled as Jumbo Salted Peanuts...the peanuts were actually small
sized unsalted. Not sure if this was an error or if the vendor intended to represe
nt the product as "Jumbo".'
```

In [8]:

```
from bs4 import BeautifulSoup
sent = BeautifulSoup(a[1], 'lxml').get_text()
sent
```

Out[8]:

```
'Product arrived labeled as Jumbo Salted Peanuts...the peanuts were actually small
sized unsalted. Not sure if this was an error or if the vendor intended to represe
nt the product as "Jumbo".'
```

TEXT Preprocessing :

In [9]:

```
#...for removing stopwords and initializing snowball stemmer

stopwords =set(['br', 'the', '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', 't
hem', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'hav
ing', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'unti
l', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'durin
g', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'un
der', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'bot
h', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', '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', "mi
ghtn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "w
asn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
snow = nltk.stem.SnowballStemmer('english')
```

In [10]:

```
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [11]:

```
# Combining all the above stundents
from bs4 import BeautifulSoup
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|████████████████████████████████████████████████████████████████| 364
171/364171 [03:15<00:00, 1865.53it/s]
```

In [12]:

```
final['CleanedText'] = preprocessed_reviews
```

In [13]:

```
conn = sqlite3.connect('final.sqlite')
c = conn.cursor()
conn.text_factory = str
final.to_sql('reviews',conn, schema = None, if_exists = 'replace', index = True, index_label =
None, chunksize = None,dtype = None)
```

In [14]:

```
#....dataframe "refined_data" is our refined dataset
con = sqlite3.connect('final.sqlite')
refined_data = pd.read_sql_query("select * from Reviews" ,con)
#refined_data.head(3)
```

In [15]:

```
#....sorting the refined_data according to time,first we will convrt time into normal form
refined_data["Time"] = pd.to_datetime(refined_data["Time"] , unit = "s")

#.....now, we will sort the data according to time.
refined_data = refined_data.sort_values('Time')
```

In [16]:

```
refined_data.head(2)
```

Out[16]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpf
0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0
30	138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2

In [17]:

```
# taking 17 percent of data it is giving us over 100k points.because of limitations in my pc.
new_data = refined_data.sample(frac=0.2746)
new_data.shape
```

Out[17]:

```
(100001, 12)
```

In [18]:

```
#....now , we will define our x and y values
X = new_data["CleanedText"]
Y = new_data["Score"]
```

Bag Of Words :

(Brute Force)

In [19]:

```
import numpy as np
import pandas as pd
import sklearn
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
```

In [20]:

```
# split the data set into train and test
X_1, X_test, y_1, y_test = model_selection.train_test_split(X, Y, test_size=0.3, random_state=0
)

# split the train data set into cross validation train and cross validation test
X_tr, X_cv, y_tr, y_cv = model_selection.train_test_split(X_1, y_1, test_size=0.3)
from sklearn.feature_extraction.text import CountVectorizer

bow = CountVectorizer()
train = bow.fit_transform(X_tr)

test = bow.transform(X_test)
cv = bow.transform(X_cv)
#features = bow.get_feature_names()
```

In [21]:

```
features = bow.get_feature_names()
```

In [24]:

```
#for storing f1_scores we are creating an empty list.
f=[]
g=[]
alpha = [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]


for i in alpha:

    clf = MultinomialNB(alpha=i)

    clf.fit(train,y_tr)

    #predicting probabilities on cv
    predicted_proba_cv  = clf.predict_proba(cv)

    #getting auc crossvalidation
    #auc_cv = sklearn.metrics.roc_auc_score(y_cv,predicted_proba_cv[:,1])
    fpr, tpr, thresholds = metrics.roc_curve(y_cv, predicted_proba_cv[:,1], pos_label='positiv
e')
    auc_cv = metrics.auc(fpr, tpr)

    f.append(auc_cv)

    predicted_proba_train  = clf.predict_proba(train)

    #auc_train = sklearn.metrics.roc_auc_score(y_tr,predicted_proba_train[:,1])
    fpr1, tpr1, thresholds1 = metrics.roc_curve(y_tr, predicted_proba_train[:,1], pos_label='po
sitive')
    auc_train = metrics.auc(fpr1,tpr1)

    g.append(auc_train)
print(f)
print(g)
```

```
[0.8015778362330076, 0.8238432061319957, 0.8508112160898411, 0.8823882929362407,
0.9102664207108432, 0.9131443796074497, 0.7206606781574206, 0.5580789053913515, 0.
5232173555664644, 0.4834009253402796]
[0.9832068543336487, 0.9827233438986669, 0.9817141714641906, 0.9793881421282384,
0.9731969115511047, 0.9514710245237269, 0.7515382051478068, 0.5636263172853954, 0.
5262150427611784, 0.4854834339882272]
```

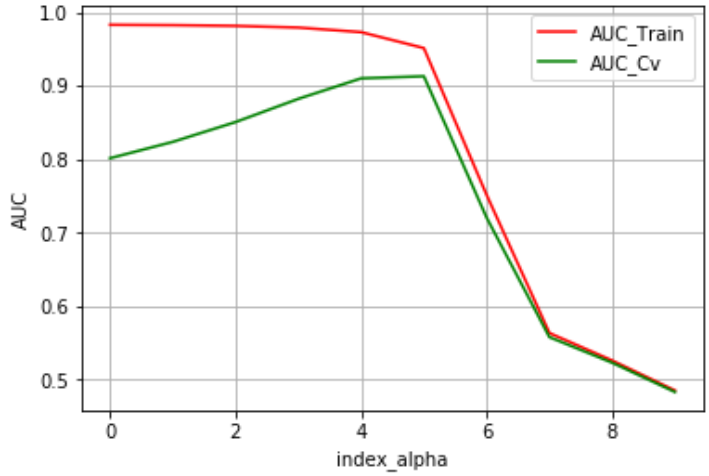
In [25]:

```
# giving indices for the alpha values so that plotting is not an issue. got this tip from ravi
 @ appliedai course.
index_alpha = [0,1,2,3,4,5,6,7,8,9]
```

Plotting the hyper perameter tuning:

In [26]:

```
map=plt.plot(index_alpha,g,index_alpha,f)
plt.setp(map[0],color='r',label='AUC_Train')
plt.setp(map[1],color='g',label='AUC_Cv')
plt.grid()
plt.legend()
plt.xlabel('index_alpha')
plt.ylabel('AUC')
plt.show()
```



from the above plot we can choose our best alpha index as 5. hence best alpha value is alpha[5] = 1.

In [28]:

```
from sklearn.metrics import confusion_matrix
best_clf = MultinomialNB(alpha = 1)
best_clf.fit(train, y_tr)
prediction = best_clf.predict(test)
predicted_test = best_clf.predict_proba(test)
#auc_test = roc_auc_score(y_test,predicted_test[:,1])
fprt, tprt, thresholdst = metrics.roc_curve(y_test, predicted_test[:,1], pos_label='positive')
auc_test = metrics.auc(fprt,tprt)
print("auc score of test is",auc_test )
```

```
auc score of test is 0.9152888370705403
```

In [29]:

```
fpr_test,tpr_test,thresholds_test = roc_curve(y_test,predicted_test[:,1],pos_label = "positive"
)
```

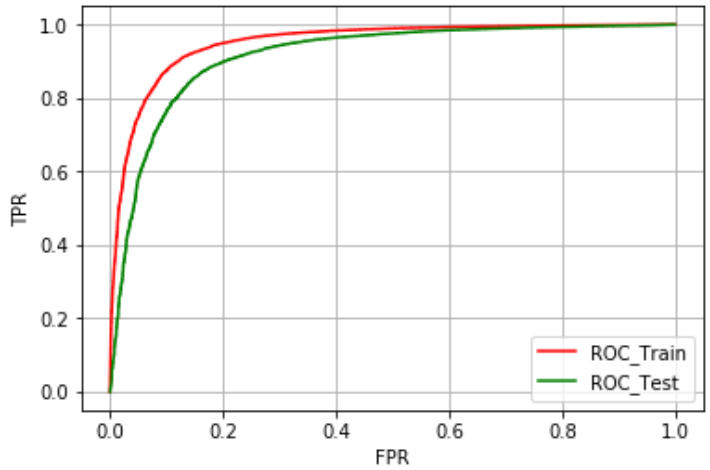
In [30]:

```
predicted_train = best_clf.predict_proba(train)
fpr_tr,tpr_tr,thresholds_tr = roc_curve(y_tr,predicted_train[:,1],pos_label = "positive")
```

Plotting Roc Curve

In [31]:

```
map=plt.plot(fpr_tr,tpr_tr,fpr_test,tpr_test)
plt.setp(map[0],color='r',label='ROC_Train')
plt.setp(map[1],color='g',label='ROC_Test')
plt.grid()
plt.legend()
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```



In [32]:

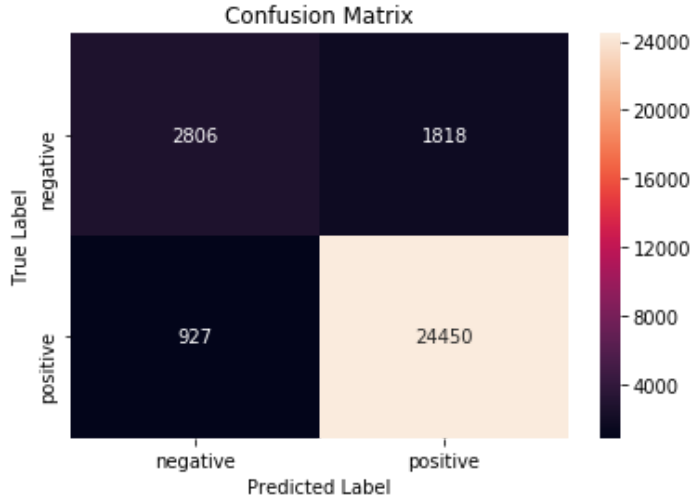
```
cm = confusion_matrix(y_test, prediction)
cm
```

Out[32]:

```
array([[ 2806,  1818],
       [  927, 24450]], dtype=int64)
```

In [33]:

```
# plot confusion matrix to describe the performance of classifier.
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



In [34]:

```
# Classification report builds a text report showing the main classification metrics
from sklearn.metrics import classification_report
print(classification_report(y_test, prediction))
```

```
             precision    recall  f1-score   support

   negative       0.75      0.61      0.67      4624
   positive       0.93      0.96      0.95     25377

avg / total       0.90      0.91      0.90     30001
```

Obtaining features from classifier:

In [35]:

```
# using feature_log_probability for finding top features in each class.
prob = abs(clf.feature_log_prob_ )
print(prob.shape)
```

```
(2, 42882)
```

In [36]:

```
#top features in class0
top_neg = np.argsort(prob[0])[-10:]
print(top_neg)
s = []
for i in top_neg:
     s.append(features[i])
print(s)
```

```
[21643 21642  4578 21640 21638 21637 21634 21633  4592 42881]
['lipswith', 'lipstick', 'britney', 'lipped', 'lipoperoxidation', 'lipoic', 'lipid
s', 'lipid', 'broadcasting', 'zzzzzzzzzzz']
```

In [37]:

```
#top features in class1
top_pos = np.argsort(prob[1])[-10:]
print(top_pos)
s = []
for i in top_pos:
     s.append(features[i])
print(s)
```

```
[36194 36192 36188 36185 20229 20232 36174 20234  2389 11988]
['strengh', 'streetlaporte', 'strech', 'streamer', 'kadoya', 'kaf', 'strawlike',
'kaffree', 'autommun', 'electrolytic']
```

Feature engineering :

Adding a new feature as no of words in the cleaned text.

In [47]:

```
refined_data.head(2)
```

Out[47]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Help
0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0
30	138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2

In [48]:

```
no_of_words=[]
for i in refined_data["CleanedText"]:
    no_of_words.append(len(i.split()))
print(no_of_words[1:10])
refined_data["Words"] = no_of_words
#print(refined_data.head(2))
```

```
[32, 13, 21, 25, 11, 32, 31, 106, 22]
```

In [49]:

```
refined_data.head(2)
```

Out[49]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Help
0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0
30	138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2

In [ ]:

```
df=pd.DataFrame()
df["CleanedText"]=refined_data["CleanedText"]
df["Words"] = refined_data["Words"]
df.head(2)
```

In [ ]:

```
X = df.sample(frac=0.2746)
Y = new_data["Score"]
```

In [ ]:

```
# split the data set into train and test
X_1, X_test, y_1, y_test = model_selection.train_test_split(X, Y, test_size=0.3, random_state=0
)

# split the train data set into cross validation train and cross validation test
X_tr, X_cv, y_tr, y_cv = model_selection.train_test_split(X_1, y_1, test_size=0.3)
from sklearn.feature_extraction.text import CountVectorizer

bow = CountVectorizer()
train = bow.fit_transform(X_tr)

test = bow.transform(X_test)
cv = bow.transform(X_cv)
#features = bow.get_feature_names()
```

In [ ]:

```
features = bow.get_feature_names()
```

In [ ]:

```
#for storing f1_scores we are creating an empty list.
f=[]
g=[]
alpha = [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]


for i in alpha:

    clf = MultinomialNB(alpha=i)

    clf.fit(train,y_tr)

    #predicting probabilities on cv
    predicted_proba_cv  = clf.predict_proba(cv)

    #getting auc crossvalidation
    auc_cv = sklearn.metrics.roc_auc_score(y_cv,predicted_proba_cv[:,1])

    f.append(auc_cv)

    predicted_proba_train  = clf.predict_proba(train)

    auc_train = sklearn.metrics.roc_auc_score(y_tr,predicted_proba_train[:,1])

    g.append(auc_train)
print(f)
print(g)
```

TF-IDF

(Brute Force)

In [38]:

```
# split the data set into train and test
X_1, X_test, y_1, y_test = model_selection.train_test_split(X, Y, test_size=0.3, random_state=0
)

# split the train data set into cross validation train and cross validation test
X_tr, X_cv, y_tr, y_cv = model_selection.train_test_split(X_1, y_1, test_size=0.3)

tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
train = tf_idf_vect.fit_transform(X_tr)
test = tf_idf_vect.transform(X_test)
cv = tf_idf_vect.transform(X_cv)
```

In [39]:

```
features = tf_idf_vect.get_feature_names()
```

In [40]:

```
#for storing f1_scores we are creating an empty list.
f=[]
g=[]
alpha = [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]


for i in alpha:

    clf = MultinomialNB(alpha=i)

    clf.fit(train,y_tr)

    #predicting probabilities on cv
    predicted_proba_cv  = clf.predict_proba(cv)

    #getting auc crossvalidation
    #auc_cv = sklearn.metrics.roc_auc_score(y_cv,predicted_proba_cv[:,1])
    fpr, tpr, thresholds = metrics.roc_curve(y_cv, predicted_proba_cv[:,1], pos_label='positiv
e')
    auc_cv = metrics.auc(fpr, tpr)


    f.append(auc_cv)

    predicted_proba_train  = clf.predict_proba(train)

    #auc_train = sklearn.metrics.roc_auc_score(y_tr,predicted_proba_train[:,1])
    fpr1, tpr1, thresholds1 = metrics.roc_curve(y_tr, predicted_proba_train[:,1], pos_label='po
sitive')
    auc_train = metrics.auc(fpr1,tpr1)

    g.append(auc_train)
print(f)
print(g)
```

```
[0.8493081798214801, 0.8706125959254234, 0.8997392368423228, 0.9307575938830455,
0.9232359531597261, 0.7924668140774869, 0.7041645717248762, 0.652086918732499, 0.6
128288043584649, 0.5980406309229502]
[0.9999954765907682, 0.9999954703234882, 0.9999953794479304, 0.9999939536417645,
0.999856766019483, 0.9381917774535555, 0.7631662834909216, 0.6807566926299493, 0.6
371119918818922, 0.6225227859571856]
```

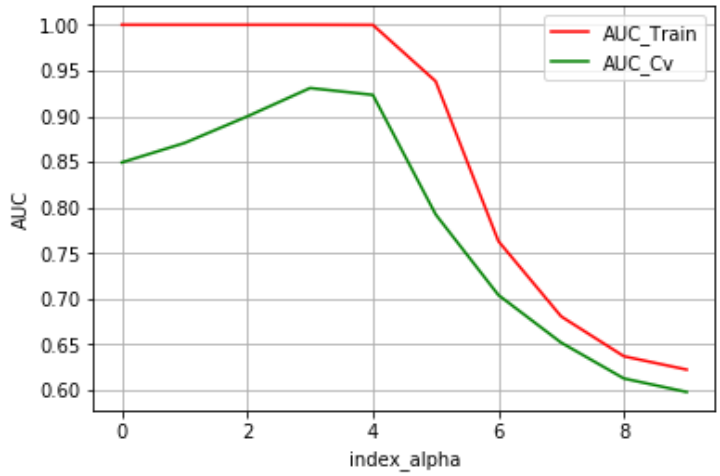
In [41]:

```
# giving indices for the alpha values so that plotting is not an issue. got this tip from ravi
 @ appliedai course.
index_alpha = [0,1,2,3,4,5,6,7,8,9]
```

Plotting the hyper perameter tuning:

In [42]:

```
map=plt.plot(index_alpha,g,index_alpha,f)
plt.setp(map[0],color='r',label='AUC_Train')
plt.setp(map[1],color='g',label='AUC_Cv')
plt.grid()
plt.legend()
plt.xlabel('index_alpha')
plt.ylabel('AUC')
plt.show()
```



from the plot , index of alpha = 4 i.e., best aplha is ,alpha[4] = 0.1

In [43]:

```
from sklearn.metrics import confusion_matrix
best_clf = MultinomialNB(alpha = 0.1)
best_clf.fit(train, y_tr)
prediction = best_clf.predict(test)
predicted_test = best_clf.predict_proba(test)
#auc_test = roc_auc_score(y_test,predicted_test[:,1])
fprt, tprt, thresholdst = metrics.roc_curve(y_test, predicted_test[:,1], pos_label='positive')
auc_test = metrics.auc(fprt,tprt)

print("auc score of test is",auc_test )
```

```
auc score of test is 0.9257921171570093
```

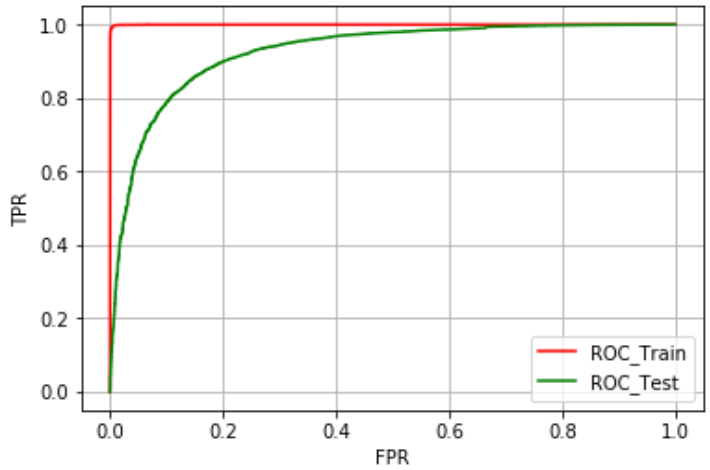
In [44]:

```
fpr_test,tpr_test,thresholds_test = roc_curve(y_test,predicted_test[:,1],pos_label = "positive"
)
predicted_train = best_clf.predict_proba(train)
fpr_tr,tpr_tr,thresholds_tr = roc_curve(y_tr,predicted_train[:,1],pos_label = "positive")
```

plotting roc curve

In [45]:

```
map=plt.plot(fpr_tr,tpr_tr,fpr_test,tpr_test)
plt.setp(map[0],color='r',label='ROC_Train')
plt.setp(map[1],color='g',label='ROC_Test')
plt.grid()
plt.legend()
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```



In [46]:

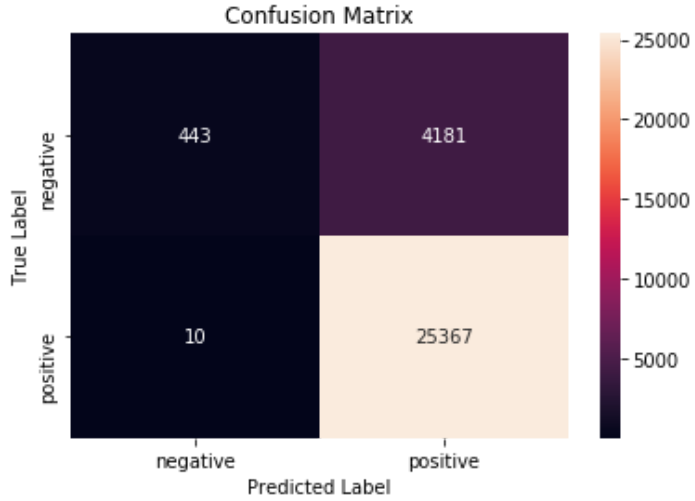
```
cm = confusion_matrix(y_test, prediction)
cm
```

Out[46]:

```
array([[  443,  4181],
       [   10, 25367]], dtype=int64)
```

In [47]:

```
# plot confusion matrix to describe the performance of classifier.
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



In [48]:

```
# Classification report builds a text report showing the main classification metrics
from sklearn.metrics import classification_report
print(classification_report(y_test, prediction))
```

```
             precision    recall  f1-score   support

   negative       0.98      0.10      0.17      4624
   positive       0.86      1.00      0.92     25377

avg / total       0.88      0.86      0.81     30001
```

Feature importance :

In [49]:

```
# using feature_log_probability for finding top features in each class.
prob = abs(clf.feature_log_prob_ )
print(prob.shape)
```

```
(2, 931415)
```

In [50]:

```
#top features in class0
top_neg = np.argsort(prob[0])[-10:]
print(top_neg)
j = []
for i in top_neg:
     j.append(features[i])
print(j)
```

```
[353844 353845 353846 353847 353848 353849 353850 353851 353816 931414]
['got healthfood', 'got healthier', 'got healthy', 'got hearty', 'got heated', 'go
t heavily', 'got heb', 'got height', 'got grassy', 'zzzzzzzzzz']
```

In [51]:

```
#top features in class1
top_pos = np.argsort(prob[1])[-10:]
print(top_pos)
s = []
for i in top_pos:
     s.append(features[i])
print(s)
```

```
[264245 264249 102721 659168 102718 827378 102717 659166 659104 462367]
['especially years', 'especially zen', 'brushing times', 'rda well', 'brushing pre
tzels', 'thick bits', 'brushing plus', 'rda supply', 'rays please', 'little nutrit
ionally']
```

Conclusion :

In [52]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Text processing Technique", "Alpha Value", "Accuracy",]
x.add_row(["Bag of Words",1, 0.9073720511189312])
x.add_row(["TF-Idf",0.1, 0.9208041502883797])
print(x)
```

```
+---------------------------+-------------+--------------------+
| Text processing Technique | Alpha Value |      Accuracy      |
+---------------------------+-------------+--------------------+
|        Bag of Words       |      1      | 0.9073720511189312 |
|           TF-Idf          |     0.1     | 0.9208041502883797 |
+---------------------------+-------------+--------------------+
```