Amazon Fine Food Reviews- Support Vector Machines

In [2]:

```
#importing necessary packages
import sqlite3
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
import pickle
import sklearn.cross_validation
from sklearn.model_selection import train_test_split
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import cross_validation
from sklearn.metrics import precision_score,recall_score,f1_score,confusion_matrix,roc_auc_score,roc_curve
```

Reading already Cleaned, Preprocessed data from database

After removing stopwords, punctuations, meaningless characters, HTML tags from Text and done stemming. Using it directly as it was alredy done in prevoius assignment

In [383]:

```
#Reading
conn= sqlite3.connect('cleanedTextData.sqlite')
data= pd.read_sql_query('''
SELECT * FROM Reviews
''',conn)
data=data.drop('index',axis=1)
data.shape
```

```
Out[383]: (364171, 11)
```

```
In [3]: data.columns
```

```
Out[3]: Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerato
        r',
               'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text',
               'CleanedText'],
              dtype='object')
```

```
In [4]: data['CleanedText'].head(3)
```

```
Out[4]: 0    witti littl book make son laugh loud recit car...
        1    rememb see show air televis year ago child sis...
        2    beetlejuic well written movi everyth act speci...
        Name: CleanedText, dtype: object
```

Linear Kernel (100k datapoints)

Sorting on the basis of 'Time' and taking top 100k pts

This data has time attribute so it will be reasonable to do time based splitting instead of random splitting.

So, before splitting we have to sort our data according to time and here we are taking 100k points from our dataset(population)

```
In [384]: data["Time"] = pd.to_datetime(data["Time"], unit = "ms")
```

```
data = data.sort_values(by = "Time")
```

In [385]:

```
#latest 100k points according to time
data= data[:100000]
len(data)
```

Out[385]:

```
100000
```

Splitting data into train60% cv20 test20%

Splitting our data into train and test data.

- train data will train our ML model
- cross validataion data will be for determining our hyperparameter
- test data will tell how Generalized our model is
- dataframes after splitting:- traindata, cvdata, testdata

In [386]:

```
traindata, testdata= train_test_split(data, test_size= 0.2, shuffle= False,stratify= None)
traindata, cvdata= train_test_split(traindata, test_size= 0.25, shuffle= False,stratify= None)
print(len(traindata),len(cvdata),len(testdata))
```

```
60000 20000 20000
```

In [387]:

```
Xtrain,Xcv,Xtest= traindata['CleanedText'],cvdata['CleanedText'],testdata['CleanedText']
Ytrain,Ycv,Ytest= traindata['Score'],cvdata['Score'],testdata['Score']
```

In [388]:

```
# converting positive to 1 and negative to -1
Ytrain=Ytrain.map(lambda x:1 if x=='Positive' else -1)
Ycv=Ycv.map(lambda x:1 if x=='Positive' else -1)
Ytest=Ytest.map(lambda x:1 if x=='Positive' else -1)
```

Taking Text and score(class) as sequences

- traindata -> Xtrain, Ytrain
- cvdata -> Xcv, Ycv
- testdata -> Xtest, Ytest

In [254]:

```
from sklearn.svm import SVC
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV,calibration_curve
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import log_loss
import warnings
warnings.simplefilter(action='ignore', category=(FutureWarning,DeprecationWarning))
```

In [264]:

```
def LinearSVCsearchCalib(Xtr,Xcv):
    '''
    Returns optimum alpha and penalty.
    We take use of CalibratedClassifierCV to find log-probabilities and further find log
    errors. Those parameters with least log_error will be our best parameters.
    We use cv data here
    It first compares C's with l2 reg then C's with l1 reg and result will be shown in two sets
    best out on two will be best hyperparameter
    '''
    alpha= [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
    log_errors= []
    for i in alpha:
        sgdC= SGDClassifier(loss='hinge',penalty='l2',alpha=i)
        calibSGD= CalibratedClassifierCV(sgdC,cv=5,method='isotonic')
        calibSGD.fit(Xtr,Ytrain)
        ypreds = calibSGD.predict_proba(Xcv)
        log_errors.append(log_loss(Ycv, ypreds, eps=1e-15, normalize=True))
    #getting score out of log error
    scores = 1 - np.array(log_errors)
    #plotting score vs parameters
    parameters= alpha
    print('Using L2 regularization:')
    plt.plot(parameters,scores)
    plt.xlabel('Hyperparameter alpha')
    plt.ylabel('log_loss')
    plt.title('log_loss vs alpha')
    plt.show()
    #showing best outcomes
    print('best log-loss - ',max(scores))
    print('best parameters using l2 is- ',alpha[log_errors.index(min(log_errors))])

    log_errors= []
    for i in alpha:
        sgdC= SGDClassifier(loss='hinge',penalty='l1',alpha=i)
        calibSGD= CalibratedClassifierCV(sgdC,cv=5,method='isotonic')
        calibSGD.fit(Xtr,Ytrain)
        ypreds = calibSGD.predict_proba(Xcv)
        log_errors.append(log_loss(Ycv, ypreds, eps=1e-15, normalize=True))
    #getting score out of log error
    scores = 1 - np.array(log_errors)
    #plotting score vs parameters
    parameters= alpha
    print('Using L1 regularization:')
    plt.plot(parameters,scores)
    plt.xlabel('Hyperparameter alpha')
    plt.ylabel('log_loss')
    plt.title('log_loss vs alpha')
    plt.show()
    #sowing best outcomes
    print('best log-loss - ',max(scores))
    print('best parameters using l1 is- ',alpha[log_errors.index(min(log_errors))])
```

In [382]:

```
def performance(y_train_pred,y_pred,clf):
    '''
    This will predict some of model's performance metrics
    '''
    print("Accuracy on test set: %0.3f%%"%(accuracy_score(Ytest, y_pred
)*100))
    print("Precision on test set: %0.3f"%(precision_score(Ytest, y_pred
)*100))
    print("Recall on test set: %0.3f"%(recall_score(Ytest, y_pred)*100
))
    print("F1-Score on test set: %0.3f"%(f1_score(Ytest, y_pred)*100))
    fpr_train,tpr_train,ts_train=roc_curve(Ytrain,y_train_pred)
    fpr,tpr,ts=roc_curve(Ytest,y_pred)
    plt.plot(fpr,tpr,label='TEST')
    plt.plot(fpr_train,tpr_train,label='TRAIN')
    plt.xlabel('False positive rate')
    plt.ylabel('True positive rate')
    plt.title('ROC curve')
    plt.legend()
    plt.show()
    print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
    df_cm = pd.DataFrame(confusion_matrix(Ytest, y_pred), range(2),rang
e(2))
    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

BOW Vectorization

Bow vectorization is basic technique to convert a text into numerical vector.

- We will build a model on train text using fit-transform
- Then transform (test) text on model build by train text
- Transformed data will be in the form of sparse matrix

In [389]:

```
# vectorizing X and transforming
bowModel=CountVectorizer()
XtrainBOWV=bowModel.fit_transform(Xtrain.values)
```

In [390]:

```
XcvBOWV= bowModel.transform(Xcv)
XtestBOWV= bowModel.transform(Xtest)
XtestBOWV.shape
```

Out[390]:

```
(20000, 36270)
```

In [ ]:

```
#Standardizing vectors
std = StandardScaler(with_mean=False).fit(XtrainBOWV)
XtrainBOWV = std.transform(XtrainBOWV)
XcvBOWV = std.transform(XcvBOWV)
XtestBOWV = std.transform(XtestBOWV)
```

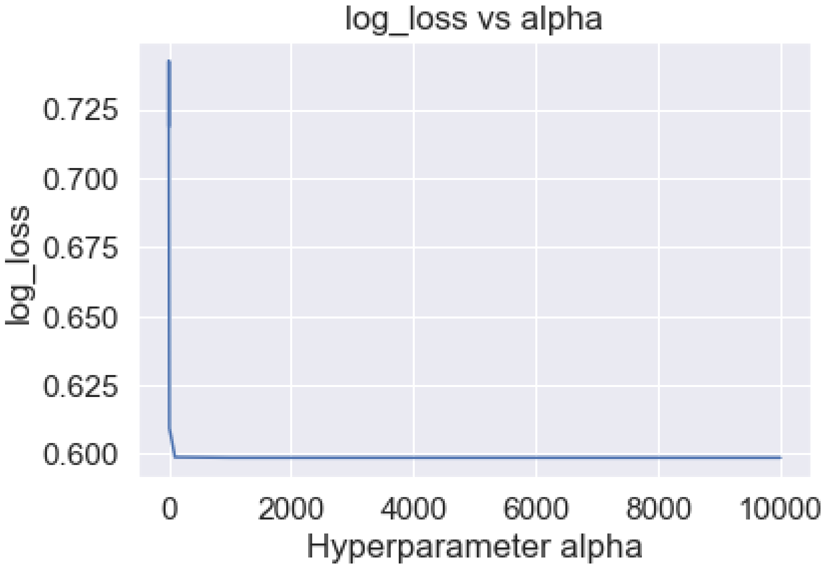
Below function will be called for getting best hyperparameters. There are two result sets:-

- One with C and l2 regularization
- other with C and l1 regulization
- best out of both will be chosen

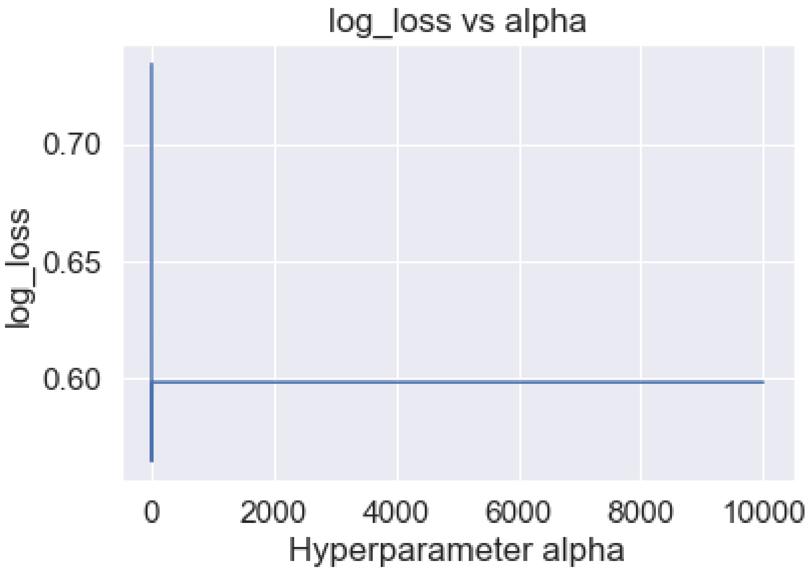
In [265]:

```
LinearSVCsearchCalib(XtrainBOWV,XcvBOWV)
```

```
Using L2 regularization:
```



```
best log-loss -  0.7423950273107458
best parameters using l2 is-  0.1
Using L1 regularization:
```



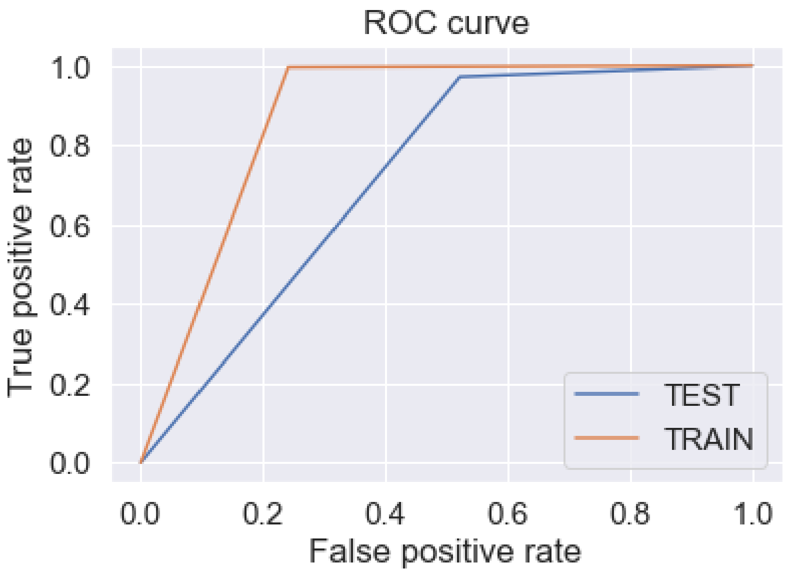
```
best log-loss -  0.734399158683549
best parameters using l1 is-  0.0001
```

It looks like l2 with alpha = 0.1 is the best parameter according to score

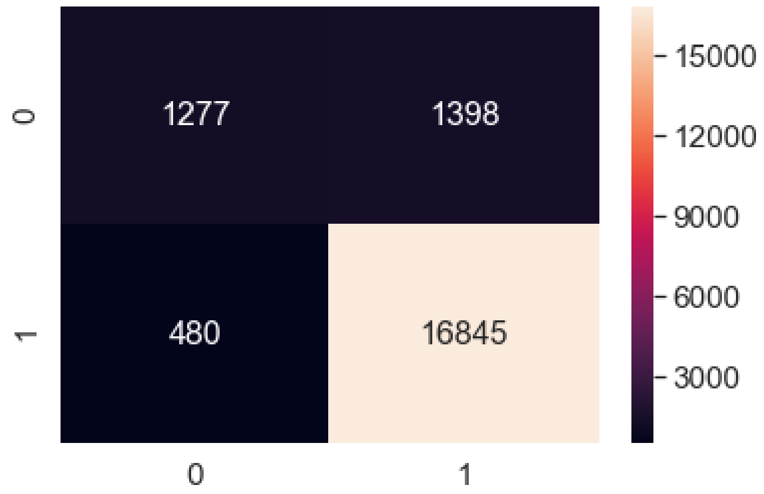
In [392]:
```
#trianing finally
sgdBOW= SGDClassifier(loss='hinge', alpha=0.1, penalty='l2')
sgdBOW.fit(XtrainBOWV,Ytrain)
train_pred= sgdBOW.predict(XtrainBOWV)
pred= sgdBOW.predict(XtestBOWV)
```

In [393]:
```
performance(train_pred,pred,sgdBOW)
```

```
Accuracy on test set: 90.610%
Precision on test set: 92.337
Recall on test set: 97.229
F1-Score on test set: 94.720
```



```
Confusion Matrix of test set:
 [ [TN  FP]
[FN TP] ]
```



Feature importance

In [269]:

```
def show_most_informative_features(vectorizer, clf, n=15):
    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\tNegative\t\t\t\t\t\tPositive")
    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))

#Code Reference:https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-
#for-scikit-learn-classifiers
```

In [272]:

```
show_most_informative_features(bowModel,sgdBOW)
```

```
                Negative
        Positive
________________________________________________________________________
_________________________
        -0.1174 disappoint                               0.2018   great
        -0.0859 worst                                    0.1836   love
        -0.0699 terribl                                  0.1497   good
        -0.0664 unfortun                                 0.1459   best
        -0.0644 return                                   0.1040   delici
        -0.0636 aw                                       0.0990   excel
        -0.0626 horribl                                  0.0866   nice
        -0.0603 wast                                     0.0866   favorit
        -0.0556 thought                                  0.0794   tasti
        -0.0554 mayb                                     0.0793   perfect
        -0.0542 bland                                    0.0774   wonder
        -0.0540 sorri                                    0.0675   find
        -0.0523 unpleas                                  0.0665   use
        -0.0510 threw                                    0.0599   easi
        -0.0481 stale                                    0.0586   enjoy
```

TFIDF vectorization

- We will build a model on train text using fit-transform
- Then transform (test) text on model build by train text
- Transformed data will be in the form of sparse matrix
- Then Standardize our data

In [394]:

```
# generating vetor out of text using tfidf
tfidfModel=TfidfVectorizer()
XtrainTFIDFV= tfidfModel.fit_transform(Xtrain)
XcvTFIDFV= tfidfModel.transform(Xcv)
XtestTFIDFV= tfidfModel.transform(Xtest)
```

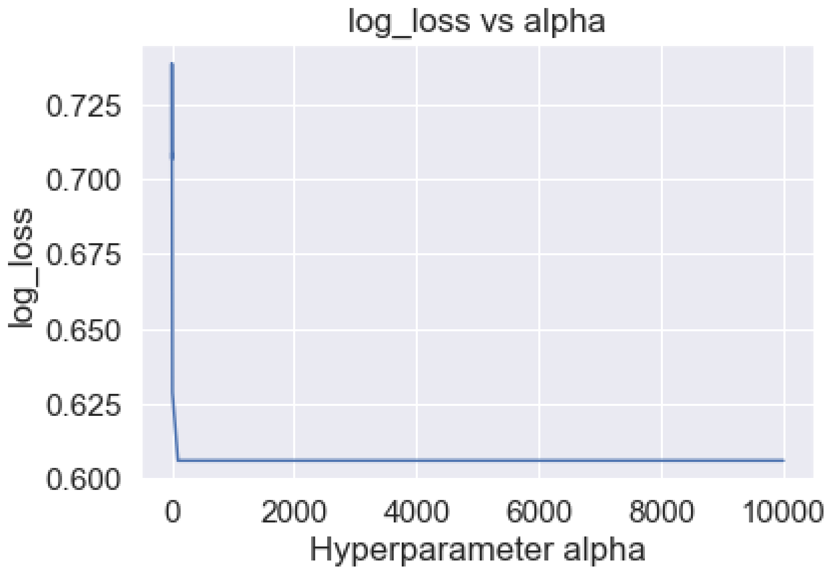
In [395]:

```
std= StandardScaler(with_mean=False)
XtrainTFIDFV = std.fit_transform(XtrainTFIDFV)
XcvTFIDFV = std.transform(XcvTFIDFV)
XtestTFIDFV = std.transform(XtestTFIDFV)
```

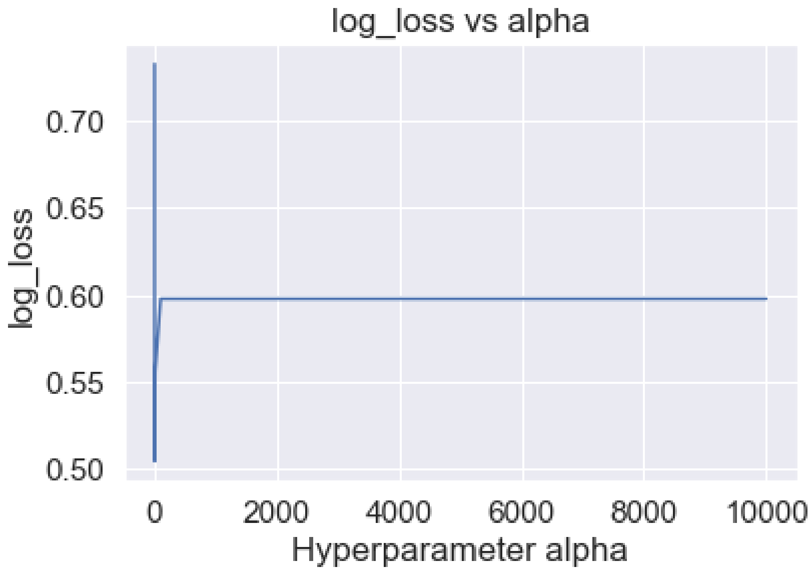
In [273]:

```
LinearSVCsearchCalib(XtrainTFIDFV,XcvTFIDFV)
```

```
Using L2 regularization:
```



```
best log-loss -  0.7386173441056143
best parameters using l2 is-  0.1
Using L1 regularization:
```



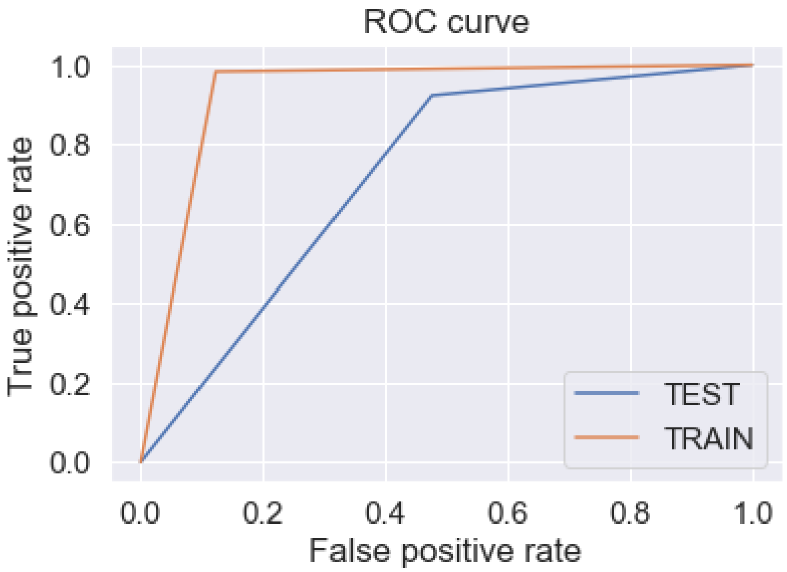
```
best log-loss -  0.732288124951558
best parameters using l1 is-  0.0001
```

It looks like l2 regularizer with alpha of 0.0001 will be good according to score

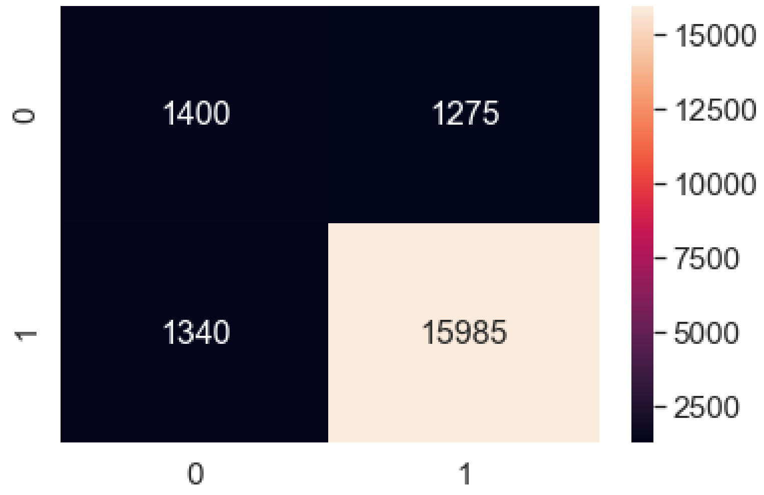
```
In [396]: sgdTFIDF= SGDClassifier(loss='hinge', alpha=0.0001, penalty='l2')
          sgdTFIDF.fit(XtrainTFIDFV,Ytrain)
          train_pred= sgdTFIDF.predict(XtrainTFIDFV)
          pred= sgdTFIDF.predict(XtestTFIDFV)
```

```
In [397]: performance(train_pred,pred,sgdTFIDF)
```

```
Accuracy on test set: 86.925%
Precision on test set: 92.613
Recall on test set: 92.266
F1-Score on test set: 92.439
```



```
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```



Feature importance

In [282]:

```
def show_most_informative_features(vectorizer, clf, n=15):
    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\tNegative\t\t\t\t\t\tPositive")
    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))

#Code Reference:https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-
#for-scikit-learn-classifiers
```

In [283]:

```
show_most_informative_features(tfidfModel, sgdTFIDF, n=15)
```

```
                        Negative
```

```
                Positive
________________________________________________________________________
_________________________
        -24.4379        worst                                    49.6235
great
        -19.8545        disappoint                               40.1818
love
        -13.6392        unpleas                                  36.1969
best
        -12.8970        aw                                       29.3371
delici
        -11.6628        thought                                  27.7191
perfect
        -11.4945        bland                                    25.1403
good
        -11.4907        peapod                                   23.9224
favorit
        -11.3042        cartooni                                 23.7174
excel
        -11.2877        tribun                                   23.4098
nice
        -11.2876        unfortun                                 19.7236
find
        -11.1819        welllll                                  19.2820
wonder
        -11.1614        threw                                    18.6928
amaz
        -10.9367        delud                                    17.4770
addict
        -10.8164        squirmi                                  17.3669
awesom
        -10.7524        terribl                                  16.6705
keep
```

Avg W2V vectorization

In [15]:

```python
import gensim
```

```
# training our gensim model on our train text
import re
import string
def cleanhtml(sentance): #substitute expression contained in <> with '
 '
    cleaned= re.sub(re.compile('<.*?>'),' ',sentance)
    return cleaned
#function for removing punctuations chars
def cleanpunc(sentance):
    cleaned= re.sub(r'[?|!|\'|"|#]',r'',sentance)
    cleaned= re.sub(r'[.|,|)|(|\|/]',r'',sentance)
    return cleaned
i=0
lists=[]

for sent in Xtrain.values:
    filtered_sentence=[]
    sent=cleanhtml(sent)
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if(cleaned_words.isalpha()):
                filtered_sentence.append(cleaned_words.lower())
            else:
                continue
    lists.append(filtered_sentence)


w2v_model= gensim.models.Word2Vec(lists,min_count=5,size=50,workers=4)
print(len(list(w2v_model.wv.vocab)))
```

```
C:\Anaconda3\lib\site-packages\gensim\utils.py:1209: UserWarning: detec
ted Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_seria
l")
```

```
9613
```

In [16]:

```
w2v_words = list(w2v_model.wv.vocab)
```

In [17]:

```
# converting list of sentance into list of list of words
# then to vector using avg w2v
# function to convert list of list of words to vect using avg w2v
def w2vVect(X):
    '''
    This function takes list of sentance as input (X) and convert it in
to
    list of list of words and then feed it into our gensim model to get
 vector
    and then take its average, finally returns sent_vectors(vector of s
entance)
    *************GENSIM MODEL WAS TRAINED ON TRAINDATA***************
    '''

    lists=[]
    for sent in X.values:
        filtered_sentence=[]
        sent=cleanhtml(sent)
        for w in sent.split():
            for cleaned_words in cleanpunc(w).split():
                if(cleaned_words.isalpha()):
                    filtered_sentence.append(cleaned_words.lower())
                else:
                    continue
        lists.append(filtered_sentence)

    sent_vectors = [];
    for sent in lists:
        sent_vec = np.zeros(50)
        cnt_words =0;
        for word in sent:
            if word in w2v_words:
                vec = w2v_model.wv[word]
                sent_vec += vec
                cnt_words += 1
        if cnt_words != 0:
            sent_vec /= cnt_words
        sent_vectors.append(sent_vec)
    return sent_vectors
```

In [398]:

```
# Vectorizing our data
XtrainW2VV= w2vVect(Xtrain)
XcvW2VV= w2vVect(Xcv)
XtestW2VV= w2vVect(Xtest)
```

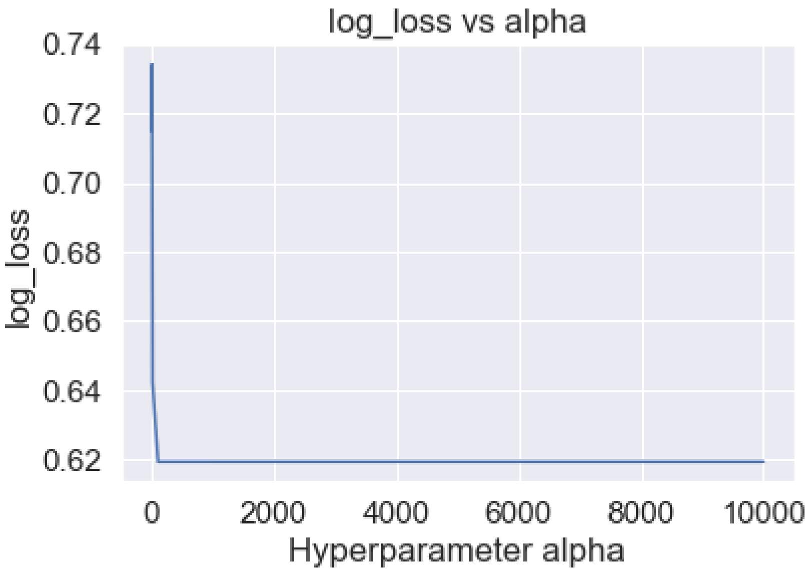
In [399]:

```
#Standardizing vectors
std = StandardScaler(with_mean=False).fit(XtrainW2VV)
XtrainW2VV = std.transform(XtrainW2VV)
XcvW2VV = std.transform(XcvW2VV)
XtestW2VV = std.transform(XtestW2VV)
```

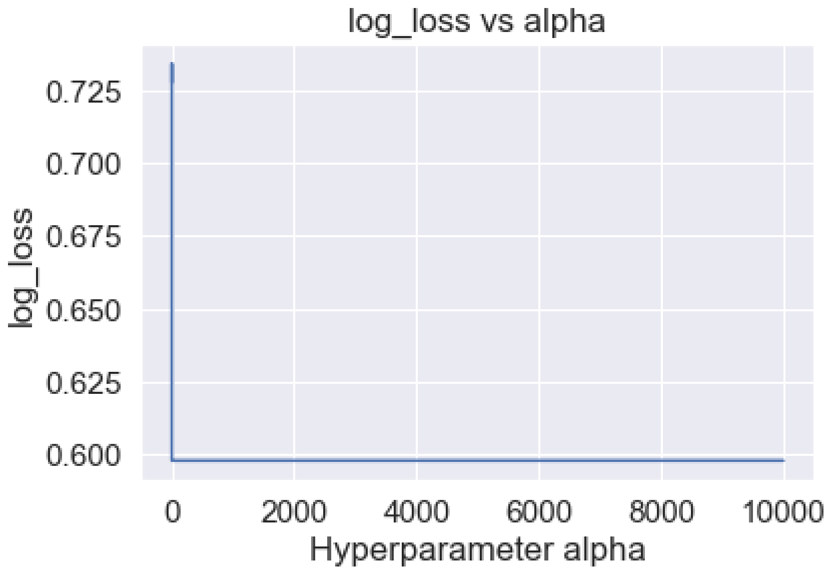
In [284]:

```
LinearSVCsearchCalib(XtrainW2VV,XcvW2VV)
```

```
Using L2 regularization:
```



```
best log-loss -  0.7341467234573728
best parameters using l2 is-  0.01
Using L1 regularization:
```



```
best log-loss -  0.7342504035162503
best parameters using l1 is-  0.001
```

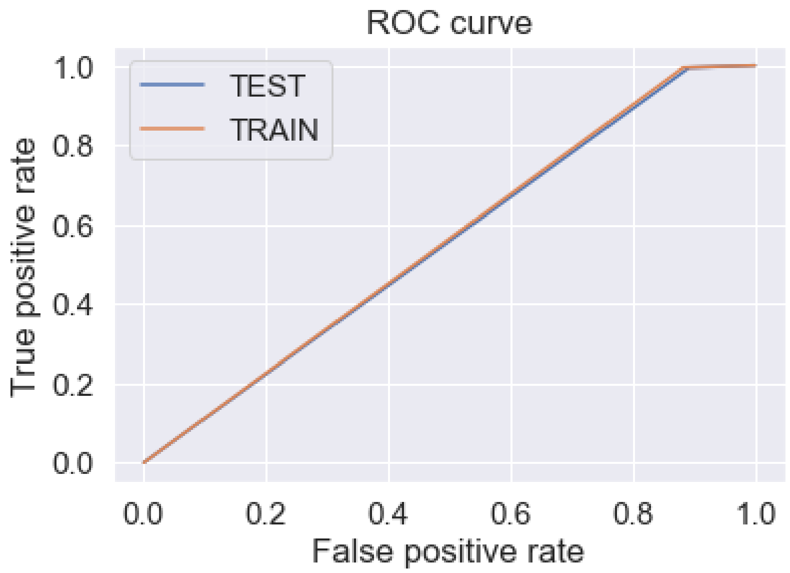
In [400]:

```
sgdW2V= SGDClassifier(loss='hinge', alpha=0.001, penalty='l1')
sgdW2V.fit(XtrainW2VV,Ytrain)
train_pred= sgdW2V.predict(XtrainW2VV)
pred= sgdW2V.predict(XtestW2VV)
```

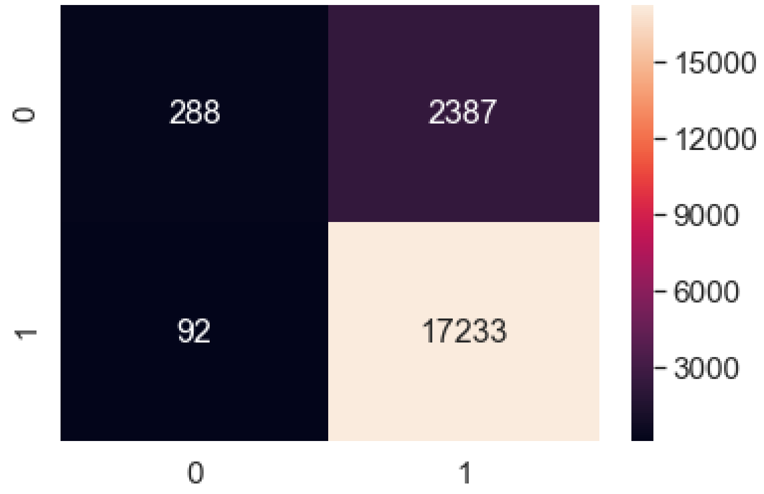
In [401]:

```
performance(train_pred,pred,sgdW2V)
```

```
Accuracy on test set: 87.605%
Precision on test set: 87.834
Recall on test set: 99.469
F1-Score on test set: 93.290
```



```
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```



tfidf-weighted avg w2v vectorization

In [20]:

```
tfmodel=TfidfVectorizer(max_features=2000)
tf_idf_matrix = tfmodel.fit_transform(Xtrain.values)
tfidf_feat=tfmodel.get_feature_names()
dictionary = {k:v for (k,v) in zip(tfmodel.get_feature_names(), list(tf
model.idf_))}
```

In [21]:

```
def tfidfw2vVect(X):
    '''
    This function converts list of sentance into list of list of words
 and then
    finally applies average-tfidf-w2w to get final sentance vector
    w2v model and w2v words already made during w2v vectorization part
    '''
    lists=[]
    for sent in X.values:
        filtered_sentence=[]
        sent=cleanhtml(sent)
        for w in sent.split():
            for cleaned_words in cleanpunc(w).split():
                if(cleaned_words.isalpha()):
                    filtered_sentence.append(cleaned_words.lower())
                else:
                    continue
        lists.append(filtered_sentence)

    tfidfw2v_sent_vectors = []; # the tfidf-w2v for each sentence/revie
w is stored in this list
    row=0;
    for sent in lists: # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length
        weight_sum =0; # num of words with a valid vector in the senten
ce/review
        for word in sent: # for each word in a review/sentence
            try:
                if word in w2v_words:
                    vec = w2v_model.wv[word]
                    #tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
                    #to reduce the computation we are
                    #dictionary[word] = idf value of word in whole courpus
                    #sent.count(word) = tf valeus of word in this review
                    tf_idf = (dictionary[word])*((sent.count(word))/len(sent))
                    sent_vec += (vec * tf_idf)
                    weight_sum += tf_idf
            except:
                pass
        if weight_sum != 0:
            sent_vec /= weight_sum
        tfidfw2v_sent_vectors.append(sent_vec)
        row += 1
    # converting nan and infinte values in vect to digit
    tfidfw2v_sent_vectors= np.nan_to_num(tfidfw2v_sent_vectors)
    return tfidfw2v_sent_vectors
```

In [402]:

```
# feeding text data and recieving vectorized data
XtrainTFIDFW2VV= tfidfw2vVect(Xtrain)
XcvTFIDFW2VV= tfidfw2vVect(Xcv)
XtestTFIDFW2VV= tfidfw2vVect(Xtest)
```

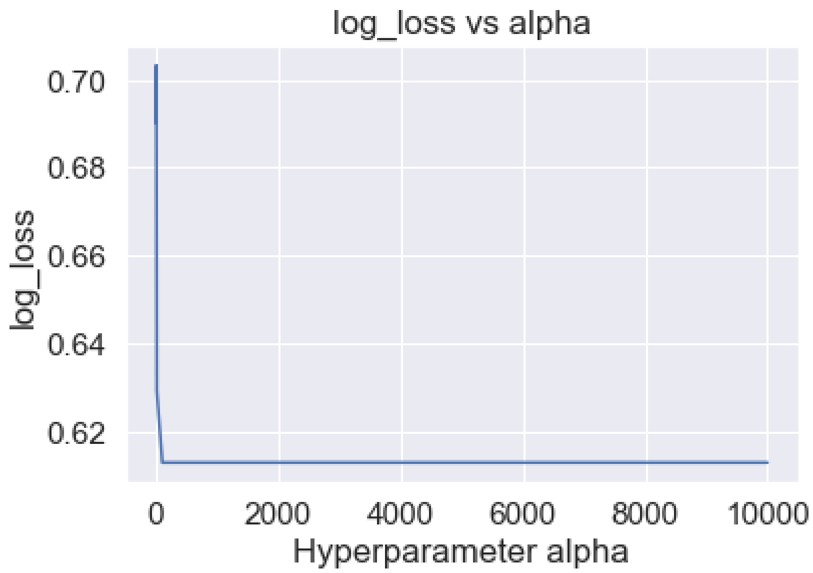
In [403]:

```
#Standardizing vectors
std = StandardScaler(with_mean=False).fit(XtrainTFIDFW2VV)
XtrainTFIDFW2VV = std.transform(XtrainTFIDFW2VV)
XcvTFIDFW2VV = std.transform(XcvTFIDFW2VV)
XtestTFIDFW2VV = std.transform(XtestTFIDFW2VV)
```

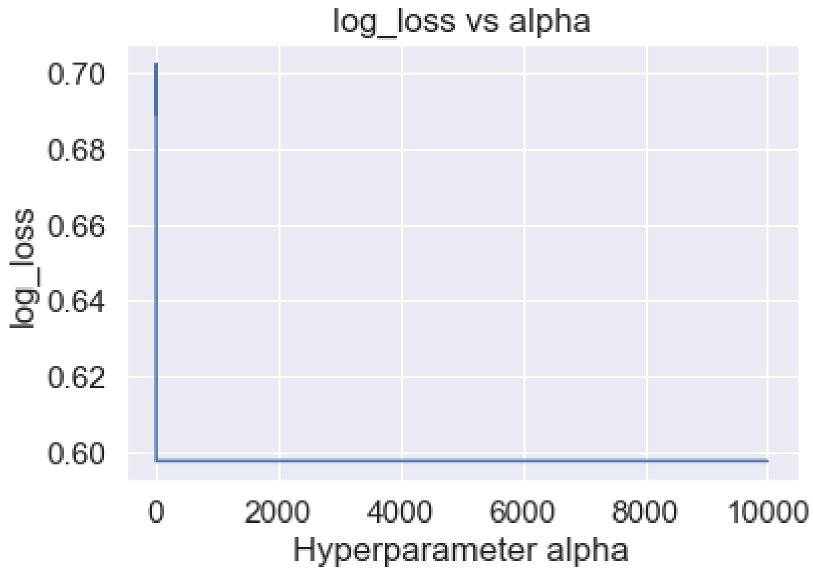
In [295]:

```
#calibratedSVC
LinearSVCsearchCalib(XtrainTFIDFW2VV,XcvTFIDFW2VV)
```

```
Using L2 regularization:
```



```
best log-loss -  0.7031723155084051
best parameters using l2 is-  0.001
Using L1 regularization:
```

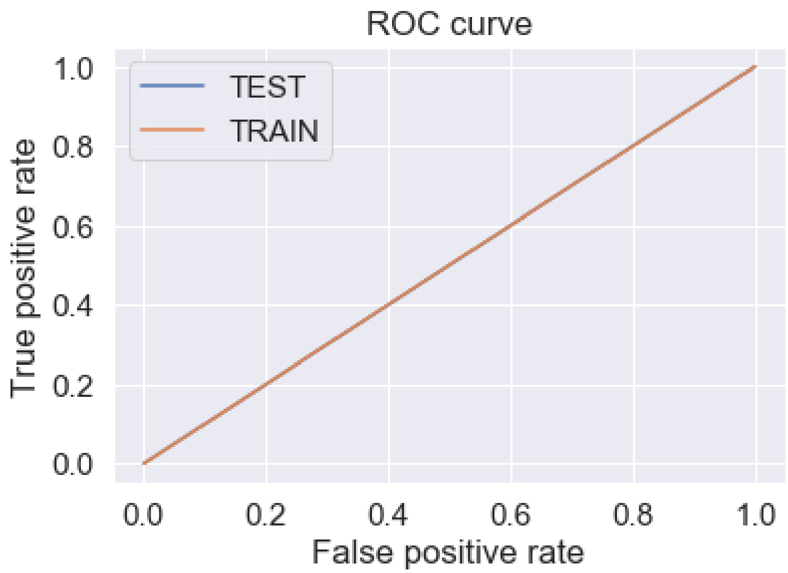


```
best log-loss -  0.7022328300348195
best parameters using l1 is-  0.001
```

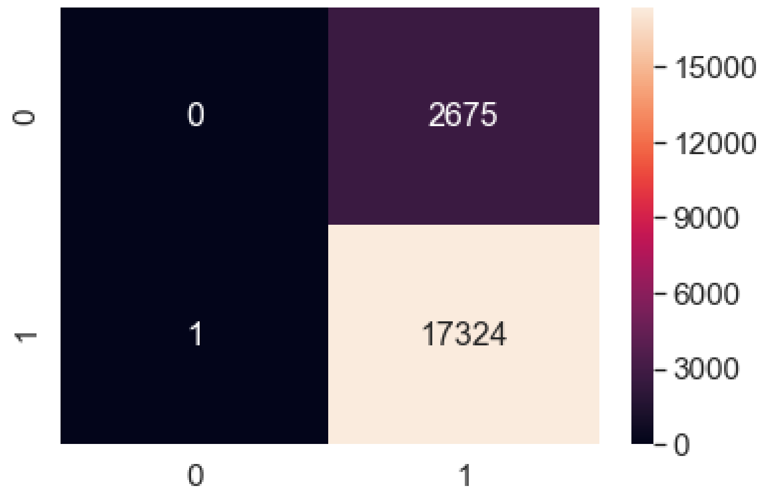
```
In [404]: sgdTFIDFW2V= SGDClassifier(loss='hinge', alpha=0.001, penalty='l1')
          sgdTFIDFW2V.fit(XtrainTFIDFW2VV,Ytrain)
          train_pred= sgdTFIDFW2V.predict(XtrainTFIDFW2VV)
          pred= sgdTFIDFW2V.predict(XtestTFIDFW2VV)
```

```
In [405]: performance(train_pred,pred,sgdTFIDFW2V)
```

```
Accuracy on test set: 86.620%
Precision on test set: 86.624
Recall on test set: 99.994
F1-Score on test set: 92.830
```



```
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```



RBF SVM (with 20k datapoints)

```
In [310]: #latest 20k points according to time
          data= data[:20000]
          len(data)

Out[310]: 20000
```

Splitting data into train60% cv20 test20%

Splitting our data into train and test data.

- train data will train our ML model
- cross validataion data will be for determining our hyperparameter
- test data will tell how Generalized our model is
- dataframes after splitting:- traindata, testdata

```
In [ ]: traindata, testdata= train_test_split(data, test_size= 0.2, shuffle= Fa
        lse,stratify= None)
```

```
traindata, cvdata= train_test_split(traindata, test_size= 0.25, shuffle
= False,stratify= None)
```

In [407]:

```
Xtrain,Xcv,Xtest= traindata['CleanedText'],cvdata['CleanedText'],testda
ta['CleanedText']
Ytrain,Ycv,Ytest= traindata['Score'],cvdata['Score'],testdata['Score']
```

In [408]:

```
# converting positive to 1 and negative to -1
Ytrain=Ytrain.map(lambda x:1 if x=='Positive' else -1)
Ycv=Ycv.map(lambda x:1 if x=='Positive' else -1)
Ytest=Ytest.map(lambda x:1 if x=='Positive' else -1)
```

Taking Text and score(class) as sequences

- traindata -> Xtrain, Ytrain
- cvdata -> Xcv, Ycv
- testdata -> Xtest, Ytest

In [369]:

```
from sklearn.svm import SVC
def SVCsearch(X,Y):
    '''
    We have two hyperparameters in rbf SVC C and gamma
    this function will find best C and gamma using Gridsearch
    '''
    parameters= {'C':[0.0001,0.001,0.01,0.1,1,10,100,1000,10000], 'gamm
a':[0.0001,0.001,0.01,0.1,1,10,100,1000,10000]}
    svc= SVC(kernel='rbf',max_iter=1000)
    gridcv= GridSearchCV(svc,parameters,scoring='roc_auc',cv=10,n_jobs=
-1)
    gridcv.fit(X,Y)
    #plotting
    plt.xlim([0.0001, 0.1])
    scores = [x[1] for x in gridcv.grid_scores_]
    scores= np.array(scores).reshape(9,9)
    for ind,i in enumerate(parameters['C']):
        plt.plot(parameters['gamma'], scores[ind], label='C: ' + str(i
))
```

```
    plt.legend()
    plt.xlabel('gamma')
    plt.ylabel('roc score')
    plt.show()
    print('best auc- ',gridcv.best_score_)
    print('best parameters- ',gridcv.best_params_)
```

BOW Vectorization

In [409]:

```
# vectorizing X and transforming
bowModel=CountVectorizer(min_df = 10, max_features = 500)
XtrainBOWV=bowModel.fit_transform(Xtrain.values)
```

In [410]:

```
XcvBOWV= bowModel.transform(Xcv)
XtestBOWV= bowModel.transform(Xtest)
XtestBOWV.shape
```

Out[410]:

```
(20000, 500)
```

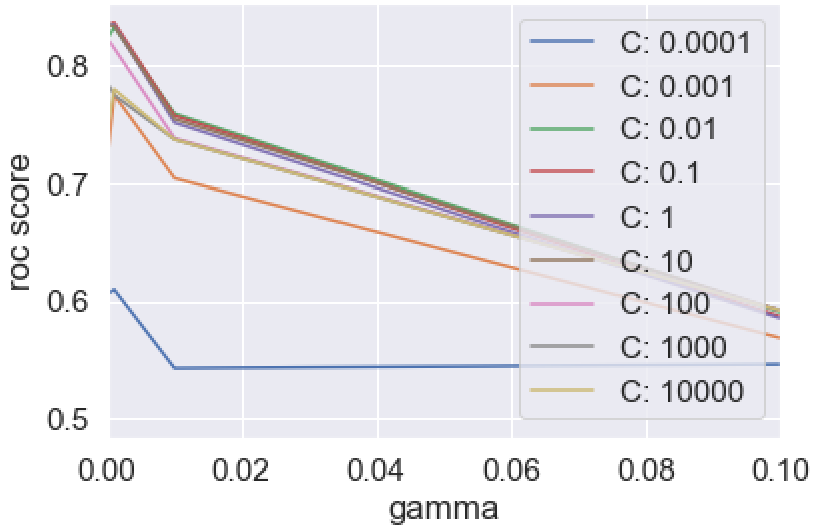
In [ ]:

```
#Standardizing vectors
std = StandardScaler(with_mean=False).fit(XtrainBOWV)
XtrainBOWV = std.transform(XtrainBOWV)
XcvBOWV = std.transform(XcvBOWV)
XtestBOWV = std.transform(XtestBOWV)
```

In [370]:

```
SVCsearch(XcvBOWV,Ycv)
```

```
C:\Anaconda3\lib\site-packages\sklearn\svm\base.py:218: ConvergenceWarn
ing: Solver terminated early (max_iter=1000).  Consider pre-processing
your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```



```
best auc-  0.8367880655690559
best parameters-  {'C': 0.1, 'gamma': 0.001}
```

In [412]:

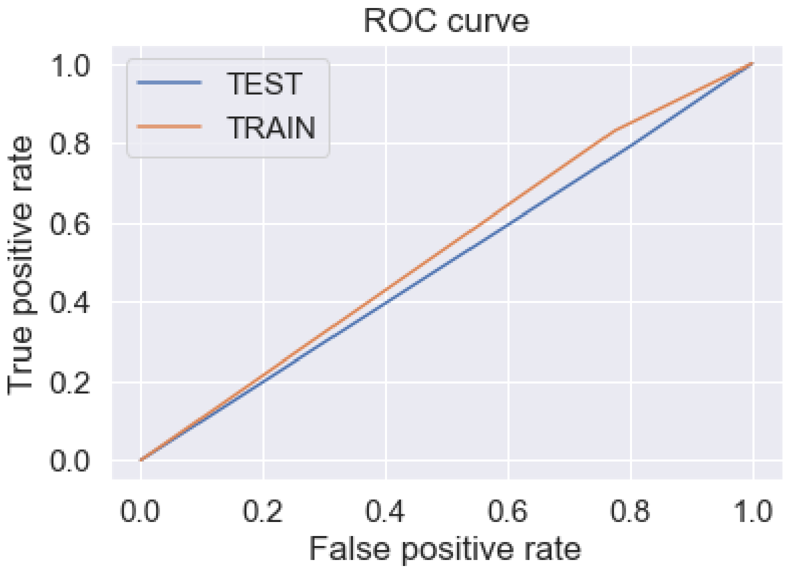
```
#finally training
svcBOW= SVC(kernel='rbf',max_iter=1000,C=0.1,gamma=0.001)
svcBOW.fit(XtrainBOWV,Ytrain)
train_pred= svcBOW.predict(XtrainBOWV)
pred= svcBOW.predict(XcvBOWV)
```

```
C:\Anaconda3\lib\site-packages\sklearn\svm\base.py:218: ConvergenceWarn
ing: Solver terminated early (max_iter=1000).  Consider pre-processing
your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```

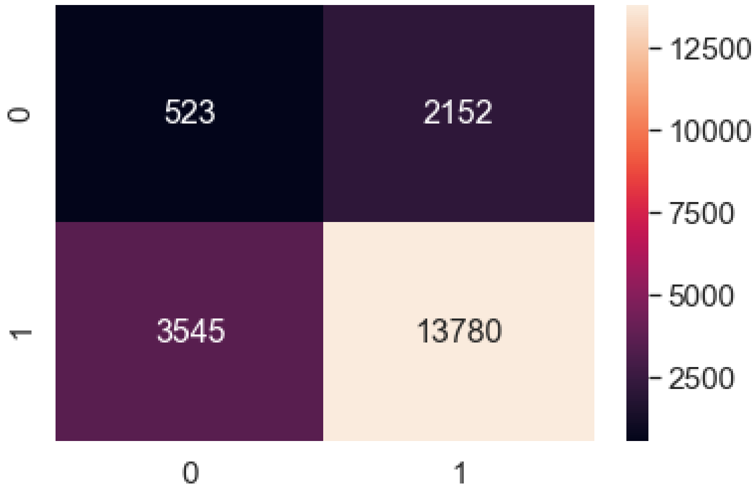
In [413]:

```
performance(train_pred,pred,svcBOW)
```

```
Accuracy on test set: 71.515%
Precision on test set: 86.493
Recall on test set: 79.538
F1-Score on test set: 82.870
```



```
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```



TFIDF vectorization

In [414]:

```
# generating vetor out of text using tfidf
tfidfModel=TfidfVectorizer(min_df = 10, max_features = 500)
XtrainTFIDFV= tfidfModel.fit_transform(Xtrain)
XcvTFIDFV= tfidfModel.transform(Xcv)
XtestTFIDFV= tfidfModel.transform(Xtest)
```

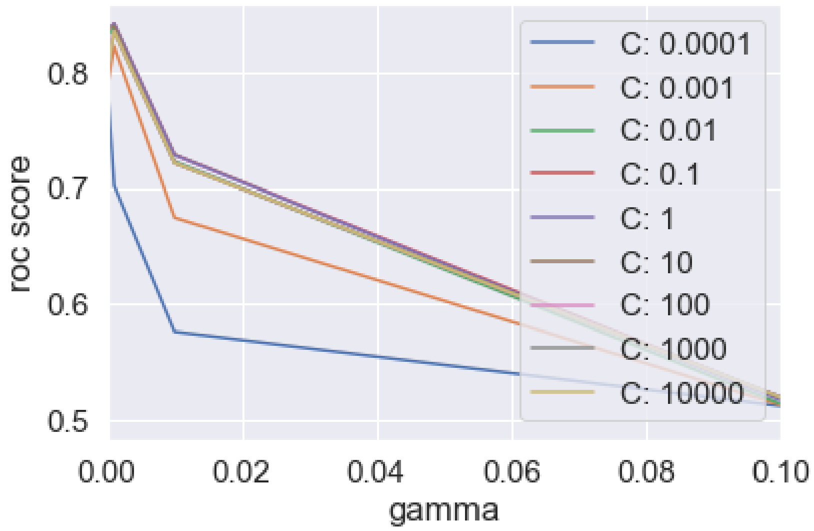
In [415]:

```
std= StandardScaler(with_mean=False)
XtrainTFIDFV = std.fit_transform(XtrainTFIDFV)
XcvTFIDFV = std.transform(XcvTFIDFV)
XtestTFIDFV = std.transform(XtestTFIDFV)
```

In [373]:

```
SVCsearch(XcvTFIDFV,Ycv)
```

```
C:\Anaconda3\lib\site-packages\sklearn\svm\base.py:218: ConvergenceWarn
ing: Solver terminated early (max_iter=1000).  Consider pre-processing
your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```



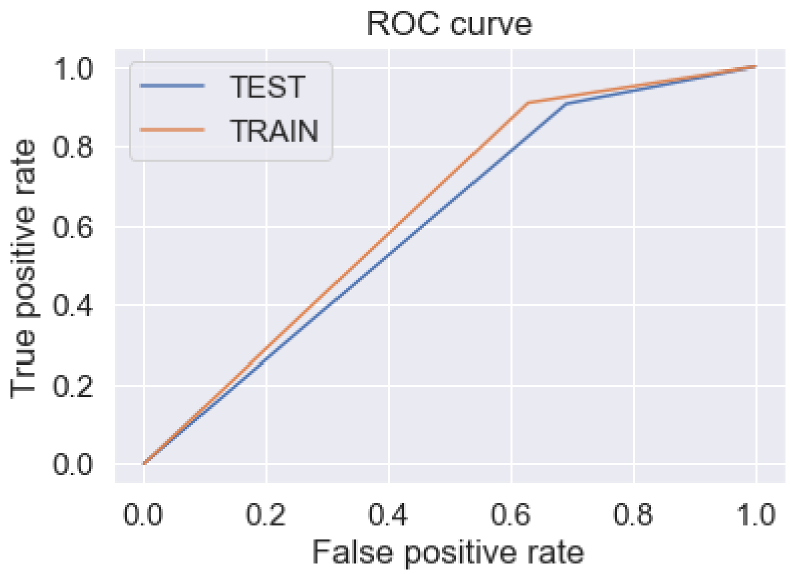
```
best auc-  0.841353375051359
best parameters-  {'C': 1, 'gamma': 0.001}
```

In [416]:
```
svcTFIDF= SVC(kernel='rbf',max_iter=1000,C=1,gamma=0.001)
svcTFIDF.fit(XtrainTFIDFV,Ytrain)
train_pred= svcTFIDF.predict(XtrainTFIDFV)
pred= svcTFIDF.predict(XtestTFIDFV)
```

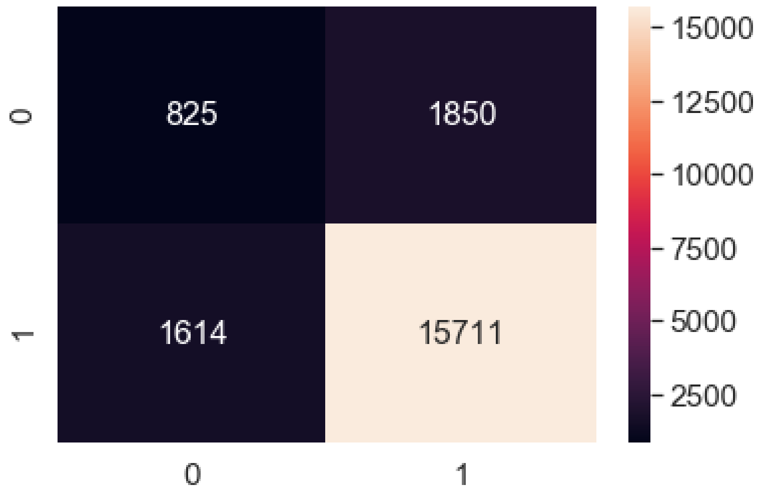
```
C:\Anaconda3\lib\site-packages\sklearn\svm\base.py:218: ConvergenceWarn
ing: Solver terminated early (max_iter=1000).  Consider pre-processing
your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```

In [417]:
```
performance(train_pred,pred,svcTFIDF)
```

```
Accuracy on test set: 82.680%
Precision on test set: 89.465
Recall on test set: 90.684
F1-Score on test set: 90.071
```



```
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```



W2V vectorization

In [418]:

```
# Vectorizing our data
XtrainW2VV= w2vVect(Xtrain)
XcvW2VV= w2vVect(Xcv)
XtestW2VV= w2vVect(Xtest)
```

In [419]:

```
#Standardizing vectors
std = StandardScaler(with_mean=False).fit(XtrainW2VV)
XtrainW2VV = std.transform(XtrainW2VV)
XcvW2VV = std.transform(XcvW2VV)
XtestW2VV = std.transform(XtestW2VV)
```

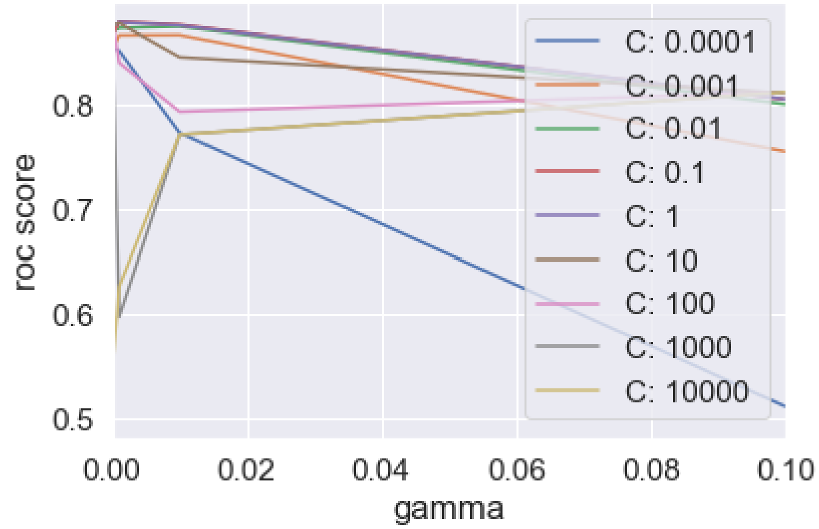
In [327]:

```
XtrainW2VV.shape
```

Out[327]:

```
(12000, 50)
```

In [376]:
```
SVCsearch(XcvW2VV,Ycv)
```



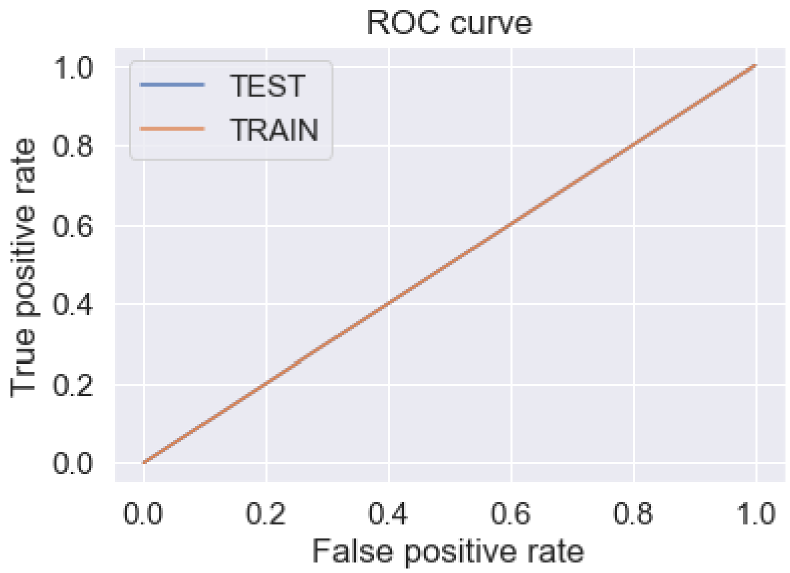
```
best auc-  0.8792813940114392
best parameters-  {'C': 0.1, 'gamma': 0.001}
```

In [420]:
```
svcW2V= SVC(kernel='rbf',max_iter=1000,C=0.1,gamma=0.001)
svcW2V.fit(XtrainW2VV,Ytrain)
train_pred= svcW2V.predict(XtrainW2VV)
pred= svcW2V.predict(XtestW2VV)
```

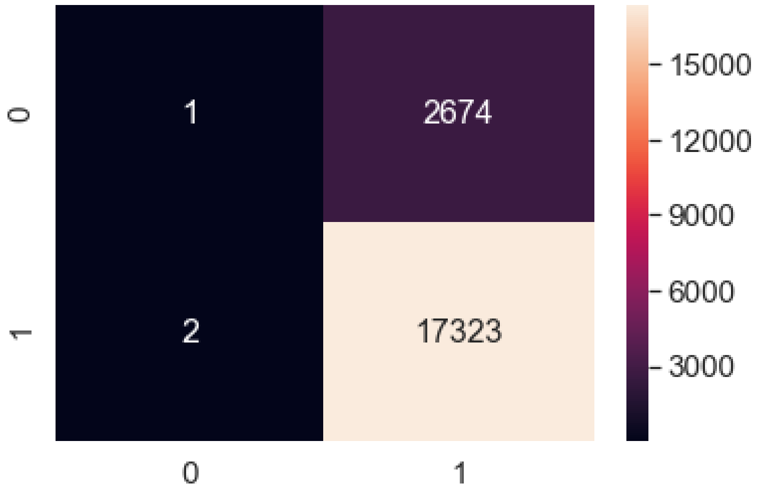
```
C:\Anaconda3\lib\site-packages\sklearn\svm\base.py:218: ConvergenceWarn
ing: Solver terminated early (max_iter=1000).  Consider pre-processing
your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```

In [421]:
```
performance(train_pred,pred,svcW2V)
```

```
Accuracy on test set: 86.620%
Precision on test set: 86.628
Recall on test set: 99.988
F1-Score on test set: 92.830
```



```
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```



tfidfw2v vectorization

In [422]:

```
tfmodel=TfidfVectorizer(max_features=500)
tf_idf_matrix = tfmodel.fit_transform(Xtrain.values)
tfidf_feat=tfmodel.get_feature_names()
dictionary = {k:v for (k,v) in zip(tfmodel.get_feature_names(), list(tf
model.idf_))}
```

In [423]:

```
# feeding text data and recieving vectorized data
XtrainTFIDFW2VV= tfidfw2vVect(Xtrain)
XcvTFIDFW2VV= tfidfw2vVect(Xcv)
XtestTFIDFW2VV= tfidfw2vVect(Xtest)
```

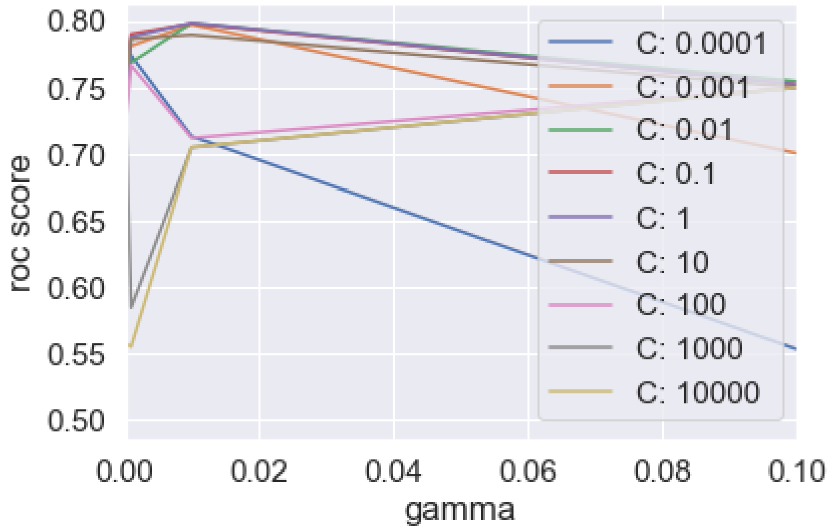
In [424]:

```
#Standardizing vectors
std = StandardScaler(with_mean=False).fit(XtrainTFIDFW2VV)
XtrainTFIDFW2VV = std.transform(XtrainTFIDFW2VV)
XcvTFIDFW2VV = std.transform(XcvTFIDFW2VV)
XtestTFIDFW2VV = std.transform(XtestTFIDFW2VV)
```

In [379]:

```
SVCsearch(XcvTFIDFW2VV,Ycv)
```

```
C:\Anaconda3\lib\site-packages\sklearn\svm\base.py:218: ConvergenceWarn
ing: Solver terminated early (max_iter=1000).  Consider pre-processing
your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```



```
best auc-  0.798321800125743
best parameters-  {'C': 1, 'gamma': 0.01}
```

In [425]:

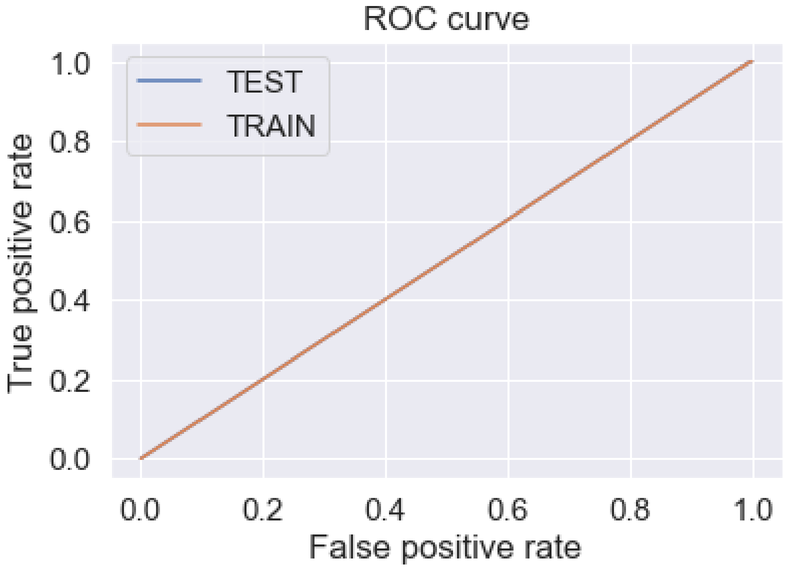
```
svcTFIDFW2V= SVC(kernel='rbf',max_iter=1000,C=0.1,gamma=0.001)
svcTFIDFW2V.fit(XtrainTFIDFW2VV,Ytrain)
train_pred= svcTFIDFW2V.predict(XtrainTFIDFW2VV)
pred= svcTFIDFW2V.predict(XtestTFIDFW2VV)
```

```
C:\Anaconda3\lib\site-packages\sklearn\svm\base.py:218: ConvergenceWarn
ing: Solver terminated early (max_iter=1000).  Consider pre-processing
your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```

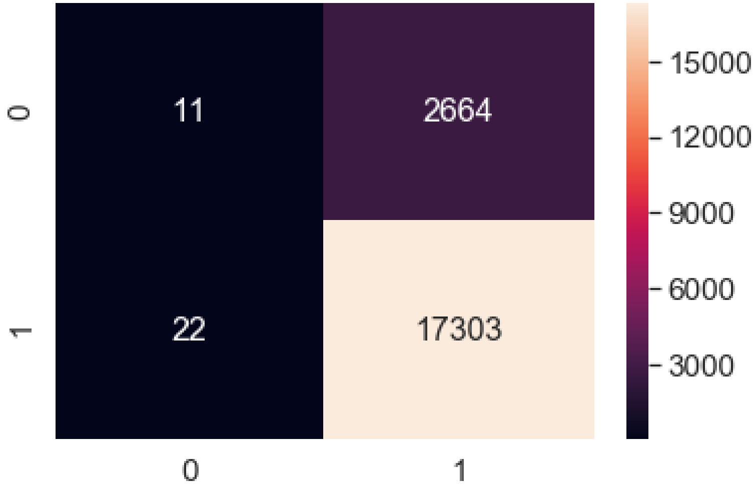
In [426]:

```
performance(train_pred,pred,svcTFIDFW2V)
```

```
Accuracy on test set: 86.570%
Precision on test set: 86.658
Recall on test set: 99.873
F1-Score on test set: 92.797
```



```
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```



Summary

LinearSGDClassifier

Vectorizer	Best Regularizer	Optimal C	Accuracy	Precision	Recall	F1 score
BOW	L2	0.1	90.610	92.337	97.229	94.720
TFIDF	L2	0.0001	86.925	92.613	92.266	92.439
W2V	L1	0.001	87.605	87.834	99.469	93.290
TFIDF-W2v	L1	0.001	86.620	86.624	99.994	92.830

RBF SVM

Vectorizer	Best gamma	Optimal C	Accuracy	Precision	Recall	F1 score
BOW	0.001	0.1	71.515	86.493	79.538	82.870
TFIDF	0.001	1	82.680	89.465	90.684	90.071
W2V	0.001	0.1	86.620	86.628	99.988	92.830
TFIDF-W2v	0.001	0.1	86.570	86.658	99.873	92.797

Observations:

- We found best results in SGDClassifier as compared to RBF SVM, the main reason is that we have taken 12k datapoints training rbf svm and 60k datapoints in SGDCassifier

- CalibratedClassifierCV helped finding calibrated probability which was further used to get best hyperparameter
- RBF kernel SVM was signinficantly slower as expected from it -> as size if n(datapoints) increases time complexity of RBF SVM increases exponentially
- Best result was from BOW featuruzed vector passed in SGDClassifier with L2 regularization and optimal C = 0.1 with F1 score of 94.7%
- Best RBF SVM model was made using W2V with gamma as 0.001 and C as 0.1 with the F1 score of 92.83%

In [1]:

```
print('END\n\n\n')
```

```
END
```