Logistic Regression on Amazon Fine FOod Reviews

In [1]:
```
#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,TfidfVector
izer
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
```

```
C:\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: Deprecat
ionWarning: This module was deprecated in version 0.18 in favor of the
model_selection module into which all the refactored classes and functi
ons are moved. Also note that the interface of the new CV iterators are
different from that of this module. This module will be removed in 0.2
0.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:
```
from sklearn.metrics import precision_score,recall_score,f1_score,confu
sion_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 [3]:
```
#Reading
conn= sqlite3.connect('cleanedTextData.sqlite')
data= pd.read_sql_query('''
SELECT * FROM Reviews
''',conn)
data=data.drop('index',axis=1)
data.shape
```

Out[3]:
```
(364171, 11)
```

In [4]:
```
data.columns
```

Out[4]:
```
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerato
r',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text',
       'CleanedText'],
      dtype='object')
```

In [5]:
```
data['CleanedText'].head(3)
```

Out[5]:
```
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
```

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 [6]:
```
data["Time"] = pd.to_datetime(data["Time"], unit = "ms")
data = data.sort_values(by = "Time")
```

In [7]:
```
#latest 100k points according to time
data= data[:100000]
len(data)
```

Out[7]: 100000

Splitting data into train70% test30%

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 [8]:
```
traindata, testdata= train_test_split(data, test_size= 0.3, shuffle= Fa
lse,stratify= None)
print(len(traindata),len(testdata))
```

```
70000 30000
```

In [9]:
```
Xtrain,Xtest= traindata['CleanedText'],testdata['CleanedText']
Ytrain,Ytest= traindata['Score'],testdata['Score']
```

In [10]:
```
# converting positive to 1 and negative to 0
Ytrain=Ytrain.map(lambda x:1 if x=='Positive' else 0)
Ytest=Ytest.map(lambda x:1 if x=='Positive' else 0)
```

Taking Text and score(class) as sequences

- traindata -> Xtrain, Ytrain

- testdata -> Xtest, Ytest

Function for testing performance when different C and penalty passed as parameters with train and test data

In [11]:
```
# this will take c, penalty, test, train data as parameters and compute
 different accuracies
# this function was made because this code will be used many times
from sklearn.linear_model import LogisticRegression
def myLogReg(c,penalti,xtrn,xtst):

    #Testing performance on Test data
    clf = LogisticRegression(C= c, penalty= penalti)
    clf.fit(xtrn,Ytrain)
    y_train_pred= clf.predict(xtrn)
    y_pred = clf.predict(xtst)
    print("AUC score on test set: %0.3f%%"%(roc_auc_score(Ytest, y_pred
)*100))
    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))
    print("Non Zero weights:",np.count_nonzero(clf.coef_))
    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 [12]:
```
# vectorizing X and transforming
bowModel=CountVectorizer()
XtrainBOWV=bowModel.fit_transform(Xtrain.values)
```

In [13]:
```
XtestBOWV= bowModel.transform(Xtest)
XtestBOWV.shape
```

Out[13]: (30000, 39730)

In [ ]:
```
#Standardizing vectors
std = StandardScaler(with_mean=False).fit(XtrainBOWV)
XtrainBOWV = std.transform(XtrainBOWV)
XtestBOWV = std.transform(XtestBOWV)
```

Gridsearch CV on BOW vector for optimal C= 1/lambda

It will be having two cases :-

1. L2 regularization
2. L1 regularization

In [15]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
```

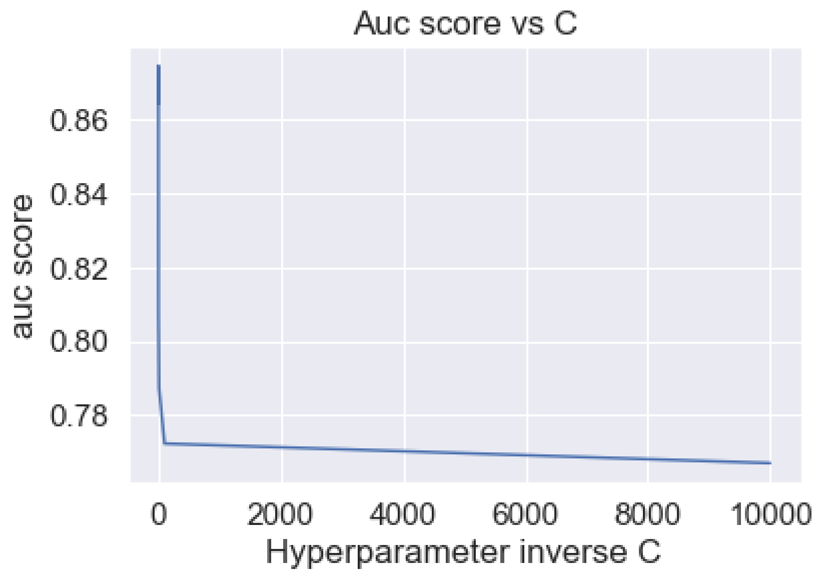
In [30]:
```
logreg=LogisticRegression()
gridSearchParam= {'C':[10**4,10**2,10,1,10**-1,10**-2,10**-4]}
gridSearch= GridSearchCV(logreg,gridSearchParam,cv=10,scoring='roc_auc'
,n_jobs=-1)
gridSearch.fit(XtrainBOWV,Ytrain)
print(gridSearch.best_estimator_)
print('Best Hyperparameter is ',gridSearch.best_params_)
print('Best auc score is ',gridSearch.best_score_)
```

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept
=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=
1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.00
01,
          verbose=0, warm_start=False)
Best Hyperparameter is  {'C': 0.01}
Best auc score is  0.8743194688817856
```

In [72]:
```
scores = [x[1] for x in gridSearch.grid_scores_]
parameters= gridSearch.param_grid['C']
plt.plot(parameters,scores)
plt.xlabel('Hyperparameter inverse C')
plt.ylabel('auc score')
plt.title('Auc score vs C')
```

```
C:\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:761:
DeprecationWarning: The grid_scores_ attribute was deprecated in versio
n 0.18 in favor of the more elaborate cv_results_ attribute. The grid_s
cores_ attribute will not be available from 0.20
  DeprecationWarning)
```

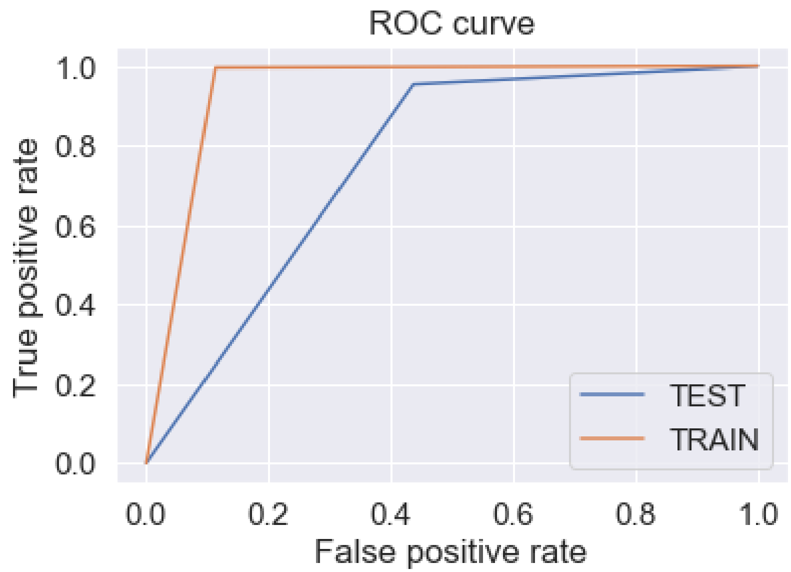
Out[72]:
```
Text(0.5,1,'Auc score vs C')
```



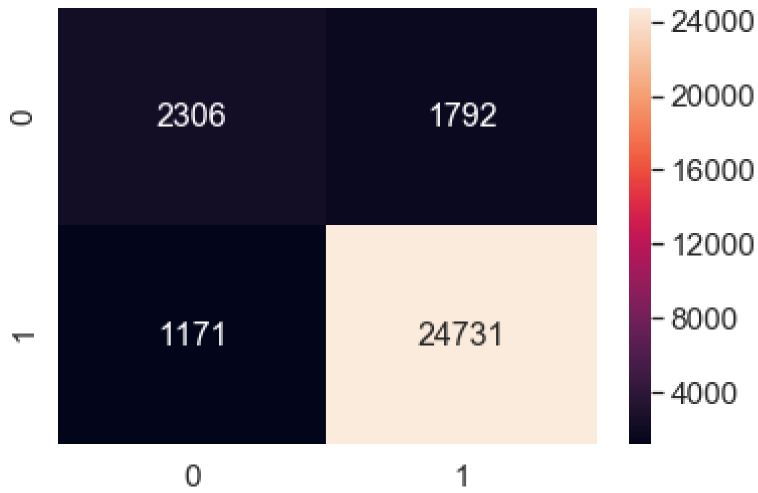
Training and testing with L2 Regularization

In [80]:
```
# calling myLogReg function. it was made because it will be used many t
imes
myLogReg(c=0.01,penalti='l2',xtrn=XtrainBOWV,xtst=XtestBOWV)
```

```
AUC score on test set: 75.875%
Accuracy on test set: 90.123%
Precision on test set: 93.244
Recall on test set: 95.479
F1-Score on test set: 94.348
Non Zero weights: 39730
```



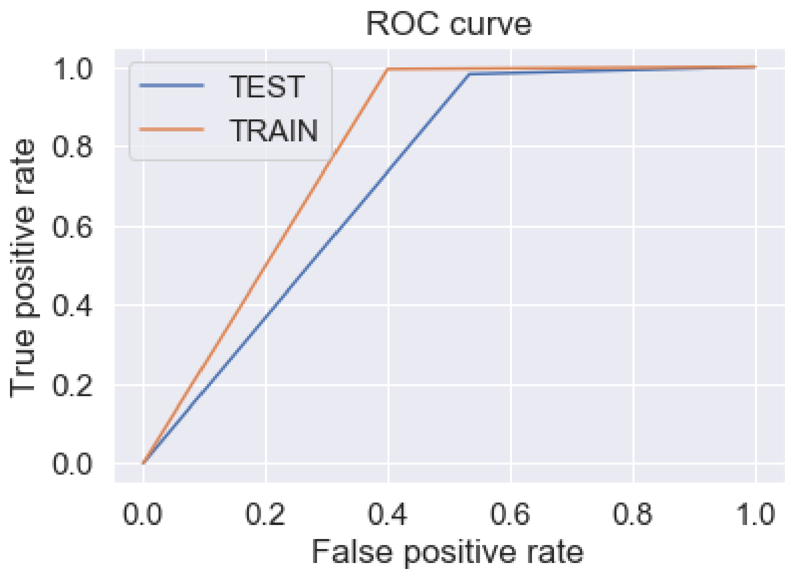

```
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```



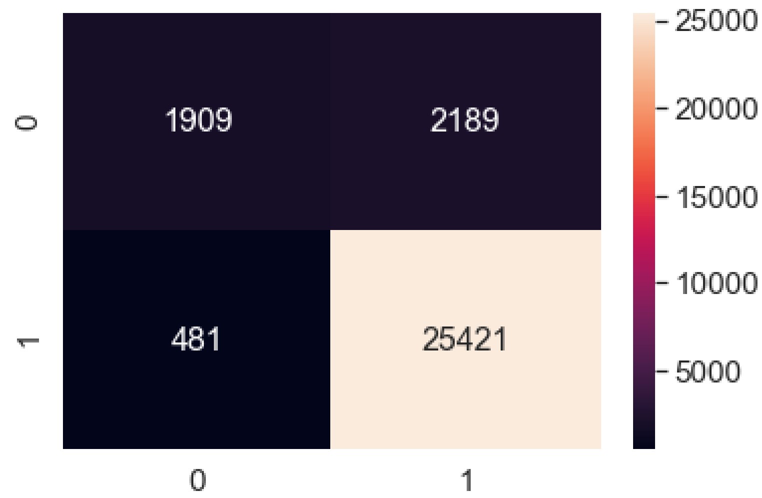
Training and testing with L1 regularization

In [81]:
```
myLogReg(c=0.01,penalti='l1',xtrn=XtrainBOWV,xtst=XtestBOWV)
```

```
AUC score on test set: 72.363%
Accuracy on test set: 91.100%
Precision on test set: 92.072
Recall on test set: 98.143
F1-Score on test set: 95.010
Non Zero weights: 3894
```




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



Observation

We can see when we regularize with L1 we get lesser auc score and we notice significant drop in nonzero weight count from 39730 in l2 to 3800 in l1

RandomizedSearch CV on BOW vector for optimal C= 1/lambda

It will be having two cases :-

1. L2 regularization
2. L1 regularization

In [84]:
```
logreg= LogisticRegression()
randSearchParam= {'C':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,
0.001,0.0005,0.0001]}
randsearch= RandomizedSearchCV(logreg,randSearchParam,cv=10,scoring='ro
c_auc',n_jobs=-1)
randsearch.fit(XtrainBOWV,Ytrain)
```

```python
print(randsearch.best_estimator_)
print('Best Hyperparameter is ',randsearch.best_params_)
print('Best auc score is ',gridSearch.best_score_)
```

```
LogisticRegression(C=0.001, class_weight=None, dual=False, fit_intercep
t=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=
1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.00
01,
          verbose=0, warm_start=False)
Best Hyperparameter is  {'C': 0.001}
Best auc score is  0.8743194688817856
```

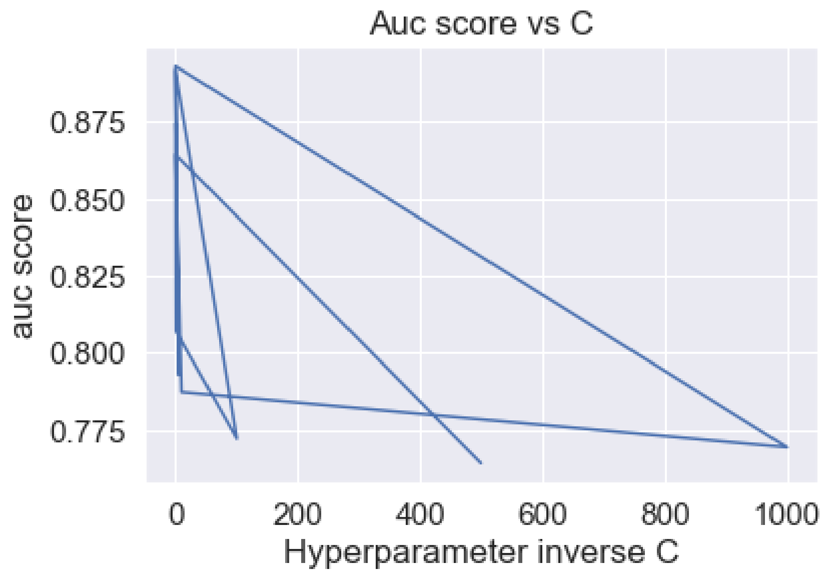
In [96]:

```python
scores = [x[1] for x in randsearch.grid_scores_]
parameters= [x[0]['C'] for x in randsearch.grid_scores_]
plt.plot(parameters,scores)
plt.xlabel('Hyperparameter inverse C')
plt.ylabel('auc score')
plt.title('Auc score vs C')
```

```
C:\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:761:
DeprecationWarning: The grid_scores_ attribute was deprecated in versio
n 0.18 in favor of the more elaborate cv_results_ attribute. The grid_s
cores_ attribute will not be available from 0.20
  DeprecationWarning)
C:\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:761:
DeprecationWarning: The grid_scores_ attribute was deprecated in versio
n 0.18 in favor of the more elaborate cv_results_ attribute. The grid_s
cores_ attribute will not be available from 0.20
  DeprecationWarning)
```

Out[96]:

```
Text(0.5,1,'Auc score vs C')
```



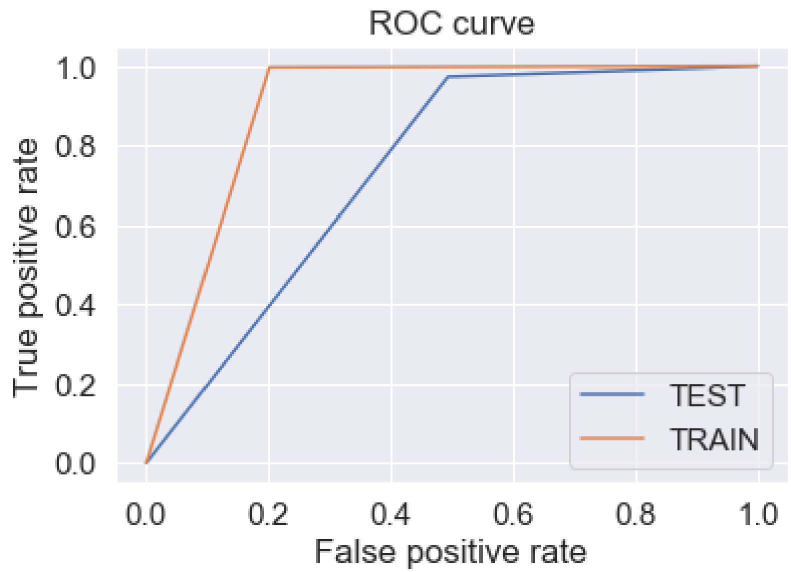
Observation

As this is randomize search it will choose hyperparameters ramdomly from given sample space thats why we get zig-zaged lines

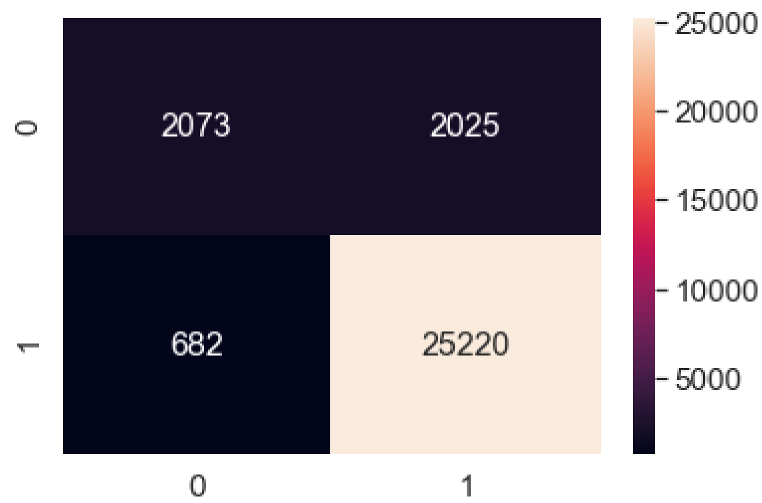
Training and testing with L2 regularization

In [98]:
```
myLogReg(c=0.001,penalti='l2',xtrn=XtrainBOWV,xtst=XtestBOWV)
```

```
AUC score on test set: 73.976%
Accuracy on test set: 90.977%
Precision on test set: 92.567
Recall on test set: 97.367
F1-Score on test set: 94.907
Non Zero weights: 39730
```




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

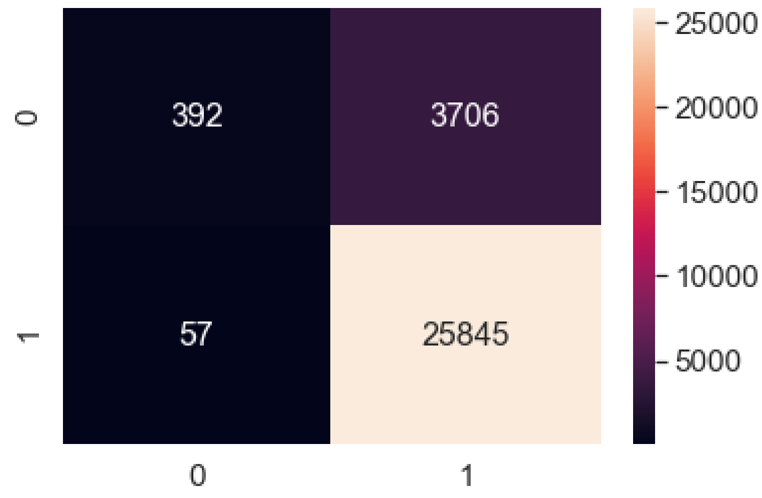


Training and testing with L1 regularization

In [99]:
```
myLogReg(c=0.001,penalti='l1',xtrn=XtrainBOWV,xtst=XtestBOWV)
```

```
AUC score on test set: 54.673%
Accuracy on test set: 87.457%
Precision on test set: 87.459
Recall on test set: 99.780
F1-Score on test set: 93.214
Non Zero weights: 50
```


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



Observation

We can see the non-Zero weights droped drastically from around 40k in L2 to 50 in L1 which obviously affect our model hence we got auc score dropped from 73% L2 to 54% in L1

Change in sparsity with changing 'C' when L1 regularized (taking Bow vect)

This is a test of how sparsity is affected when C is changed when L1 regulairzed

In [16]:
```
# loop with different c's
for c in [1000,100,10,1,0.1,0.01,0.001]:
    clf = LogisticRegression(C= c, penalty= 'l1',n_jobs=1)
    clf.fit(XtrainBOWV,Ytrain)
    y_pred = clf.predict(XtestBOWV)
    print('\nWhen C is : ',c,'-----------------------------------------
------------',sep=' ')
    print("AUC score on test set: %0.3f%%"%(roc_auc_score(Ytest, y_pred
)*100))
```

```
    print("Accuracy on test set: %0.3f%%"%(accuracy_score(Ytest, y_pred
)*100))
    print("F1-Score on test set: %0.3f"%(f1_score(Ytest, y_pred)*100))
    print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
When C is :  1000 --------------------------------------------------------
AUC score on test set: 71.940%
Accuracy on test set: 86.413%
F1-Score on test set: 92.110
Non Zero weights: 23555

When C is :  100 --------------------------------------------------------
AUC score on test set: 72.478%
Accuracy on test set: 86.457%
F1-Score on test set: 92.122
Non Zero weights: 17738

When C is :  10 --------------------------------------------------------
AUC score on test set: 73.142%
Accuracy on test set: 86.787%
F1-Score on test set: 92.315
Non Zero weights: 15296

When C is :  1 --------------------------------------------------------
AUC score on test set: 75.383%
Accuracy on test set: 88.440%
F1-Score on test set: 93.308
Non Zero weights: 14242

When C is :  0.1 --------------------------------------------------------
AUC score on test set: 77.850%
Accuracy on test set: 90.927%
F1-Score on test set: 94.803
Non Zero weights: 11234

When C is :  0.01 --------------------------------------------------------
AUC score on test set: 72.343%
Accuracy on test set: 91.100%
F1-Score on test set: 95.011
Non Zero weights: 3886
```

```
When C is :  0.001 -----------------------------------------------------
-
AUC score on test set: 54.673%
Accuracy on test set: 87.457%
F1-Score on test set: 93.214
Non Zero weights: 50
```

Observation

We can clearly see that :-

- C decreased -> lambda(c=1/lambda) increased -> nonzero weights decreased -> sparsity increased
- so, as C decreases sparsity increases

Perturbation test (Multicollinearity)

- First we will train a model on optimal C with l2 regularization
- find and save the weight vector we got from above
- then we will introduce nose to our vectorized data
- then again we will find new weight vector
- then finally we will compare the both and save their corresponding percentage changes
- Finally we will plot the percentiles to check how much multicollinear our data is

In [82]:
```
#training our model
clf = LogisticRegression(C= 0.01, penalty= 'l2')
clf.fit(XtrainBOWV,Ytrain)
y_pred = clf.predict(XtestBOWV)
print("F1 score on test set: %0.3f%%"%(f1_score(Ytest, y_pred)*100))
print("AUC score on test set: %0.3f%%"%(roc_auc_score(Ytest, y_pred)*10
0))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("Total weights:",len(clf.coef_[0]))
```

```
F1 score on test set: 94.346%
```

```
F1 score on test set: 94.346%
AUC score on test set: 75.863%
Non Zero weights: 39730
Total weights: 39730
```

In [83]:

```python
#Weights before adding random noise
from scipy.sparse import find
weights1 = find(clf.coef_[0])[2]
print(weights1[:20])
```

```
[ 1.00092941e-09  5.42176440e-03  1.41171662e-06  5.29485361e-03
 -8.94430645e-03  2.28240842e-07  1.00489425e-03 -1.26511292e-03
  3.34873056e-03  5.20346943e-04  8.35412507e-03  3.92977137e-02
  1.67122729e-03  2.14251515e-02 -4.59131503e-02  4.51357434e-03
  1.10295558e-04  1.57678030e-02  3.46710592e-03  5.36605929e-03]
```

In [84]:

```python
# adding random noise
XtrainBOWV_t = XtrainBOWV
#Random noise of size same as weights
epsilon = np.random.uniform(low=-0.0001, high=0.0001, size=(find(Xtrain
BOWV_t)[0].size,))
#Getting the postions(row and column) and value of non-zero datapoints
a,b,c = find(XtrainBOWV_t)

#Introducing random noise to non-zero datapoints
XtrainBOWV_t[a,b] = epsilon + XtrainBOWV_t[a,b]
```

In [86]:

```python
#Training on train data having random noise
clf = LogisticRegression(C= 0.01, penalty= 'l2')
clf.fit(XtrainBOWV_t,Ytrain)
y_pred = clf.predict(XtestBOWV)
print("F1 score on test set: %0.3f%%"%(f1_score(Ytest, y_pred)*100))
print("AUC score on test set: %0.3f%%"%(roc_auc_score(Ytest, y_pred)*10
0))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("Total weights:",len(clf.coef_[0]))
```

```
F1 score on test set: 94.346%

AUC score on test set: 75.863%
```

```
Non Zero weights: 39730
Total weights: 39730
```

In [87]:

```
#Weights after adding random noise
weights2 = find(clf.coef_[0])[2]
print(weights2[:20])
```

```
[ 1.01205074e-09  5.42160746e-03  1.41195863e-06  5.29469810e-03
 -8.94369993e-03  2.28302120e-07  1.00487227e-03 -1.26517450e-03
  3.34873413e-03  5.20298586e-04  8.35425288e-03  3.92983616e-02
  1.67116403e-03  2.14256791e-02 -4.59127558e-02  4.51351041e-03
  1.10286334e-04  1.57678967e-02  3.46711807e-03  5.36596911e-03]
```

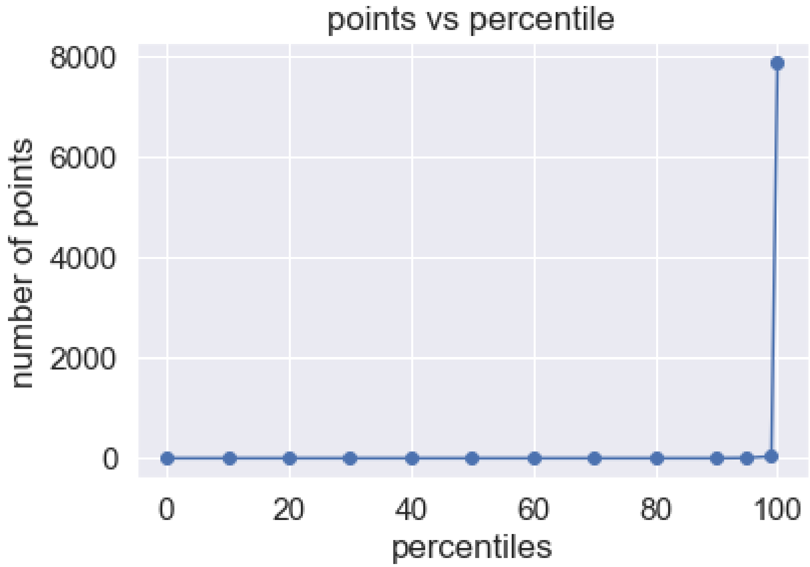
In [88]:

```
#percentage difference in weights Wi
percent_weight_diff = 100*abs(weights2 - weights1)/abs(weights1)
```

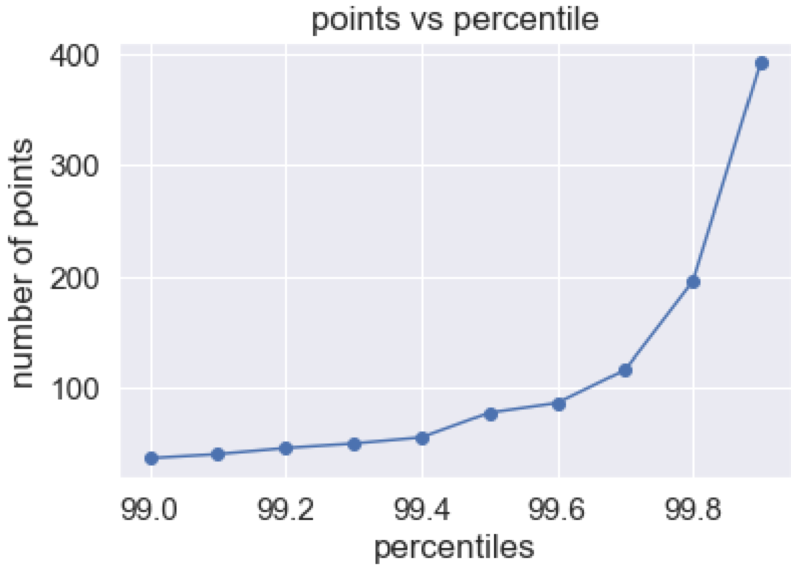
In [89]:

```
perc=[0,10,20,30,40,50,60,70,80,90,95,99,100]
plt.plot(perc, np.percentile(percent_weight_diff,perc),'bo-')
print(np.percentile(percent_weight_diff, perc))
plt.xlabel('percentiles')
plt.ylabel('number of points')
plt.title('points vs percentile')
plt.show()
```

```
[7.11883104e-09 3.24746368e-04 6.72732432e-04 1.08145584e-03
 1.62470888e-03 2.35323398e-03 3.49759690e-03 5.52308194e-03
 9.91111297e-03 4.81095895e-02 5.62355578e+00 3.60139027e+01
 7.87561358e+03]
```



In [90]:
```
perc= np.arange(99.0,100,0.1)
plt.plot(perc, np.percentile(percent_weight_diff,perc),'bo-')
plt.xlabel('percentiles')
plt.ylabel('number of points')
plt.title('points vs percentile')
plt.show()
```



In [91]:
```
# print number of weights which changed by 30%
print(percent_weight_diff[np.where(percent_weight_diff > 30)].size)
```

```
659
```

Observation

- We can see that as we introduce noise in data the some weights changes significantly. there are 659 Wi's which changed by minimum 30% (taking 30% as threshold)
- Between 99.6 to 100 percentile of points there are large amount of points(we can see it in plot)
- So finally by looking at these ovservation we can say that our L2 regularized BOW vectorized logistic regression model is showing multicollinearity as far as perturbation test is concerned

Feature importance

```
In [92]: clf = LogisticRegression(C= 0.01, penalty= 'l2')
clf.fit(XtrainBOWV,Ytrain)
y_pred = clf.predict(XtestBOWV)
print("F1 score on test set: %0.3f%%"%(f1_score(Ytest, y_pred)*100))
print("AUC score on test set: %0.3f%%"%(roc_auc_score(Ytest, y_pred)*10
0))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("Total weights:",len(clf.coef_[0]))
```

```
F1 score on test set: 94.346%
AUC score on test set: 75.863%
Non Zero weights: 39730
Total weights: 39730
```

```
In [94]: 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\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))

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

```
                        Negative
                Positive
________________________________________________________________________
_________________________
        -0.3722 disappoint                                 0.8316  great

        -0.3075 worst                                      0.6557  love

        -0.2317 unfortun                                   0.6289  best
```

```
-0.2197 stale                                  0.4527   perfect
-0.2184 aw                                     0.4267   delici
-0.2139 thought                                0.4120   good
-0.2111 wast                                   0.3943   nice
-0.2093 horribl                                0.3866   excel
-0.2080 tast                                   0.3396   favorit
-0.2058 terribl                                0.3186   find
-0.2052 return                                 0.3052   wonder
-0.2020 stick                                  0.3019   amaz
-0.1956 bad                                    0.2949   addict
-0.1954 bland                                  0.2500   keep
-0.1937 would                                  0.2460   refresh
```

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 [17]:
```
# generating vetor out of text using tfidf
tfidfModel=TfidfVectorizer()
```

```
XtrainTFIDFV= tfidfModel.fit_transform(Xtrain)
XtestTFIDFV= tfidfModel.transform(Xtest)
```

In [18]:

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

Gridsearch CV

In [100]:

```
logreg=LogisticRegression()
gridSearchParam= {'C':[10**4,10**2,10,1,10**-1,10**-2,10**-4]}
gridSearch= GridSearchCV(logreg,gridSearchParam,cv=10,scoring='roc_auc'
,n_jobs=-1)
gridSearch.fit(XtrainTFIDFV,Ytrain)
print(gridSearch.best_estimator_)
print('Best Hyperparameter is ',gridSearch.best_params_)
print('Best auc score is ',gridSearch.best_score_)
```

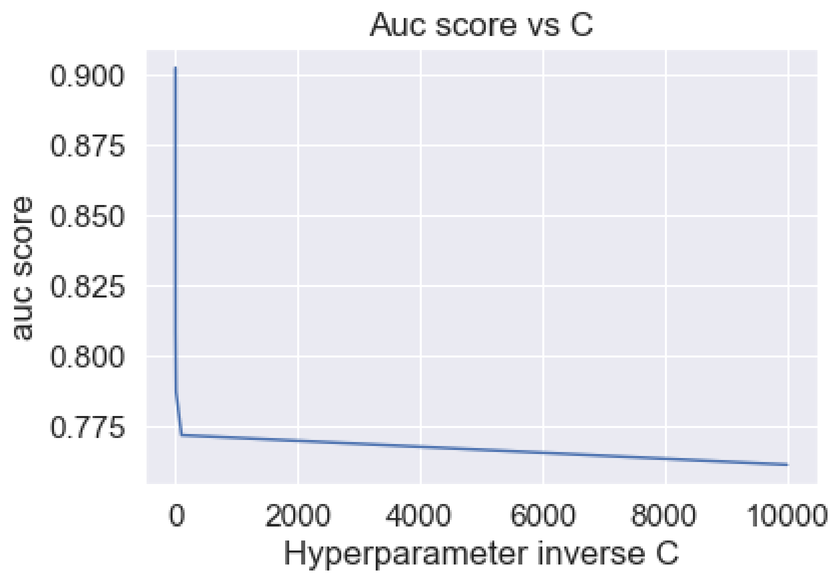
```
LogisticRegression(C=0.0001, class_weight=None, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
Best Hyperparameter is  {'C': 0.0001}
Best auc score is  0.9023941144816782
```

In [101]:

```
scores = [x[1] for x in gridSearch.grid_scores_]
parameters= gridSearch.param_grid['C']
plt.plot(parameters,scores)
plt.xlabel('Hyperparameter inverse C')
plt.ylabel('auc score')
plt.title('Auc score vs C')
```

```
C:\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:761:
DeprecationWarning: The grid_scores_ attribute was deprecated in versio
n 0.18 in favor of the more elaborate cv_results_ attribute. The grid_s
cores_ attribute will not be available from 0.20
  DeprecationWarning)
```

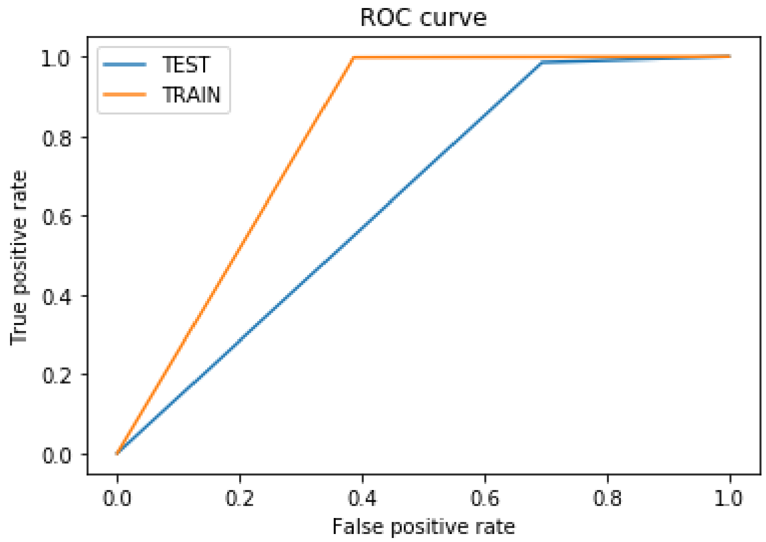
Out[101]: Text(0.5,1,'Auc score vs C')



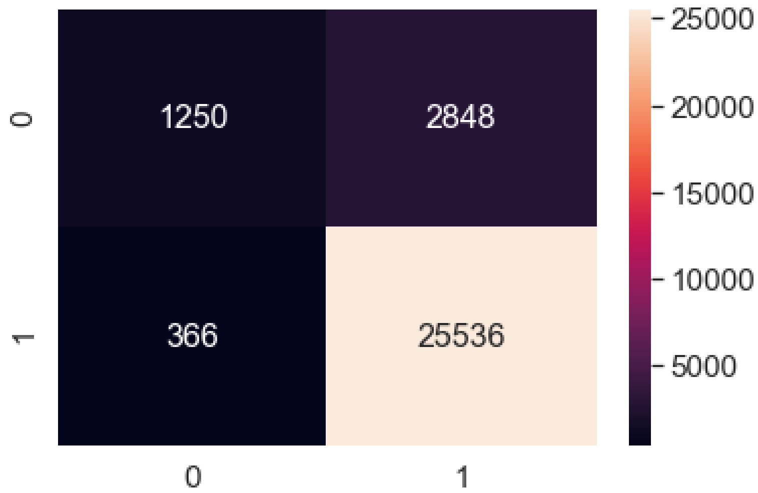
Training and testing with L2 regularization

In [19]:
```
myLogReg(c=0.0001,penalti='l2',xtrn=XtrainTFIDFV,xtst=XtestTFIDFV)
```

```
AUC score on test set: 64.545%
Accuracy on test set: 89.287%
Precision on test set: 89.966
Recall on test set: 98.587
F1-Score on test set: 94.080
Non Zero weights: 39730
```




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

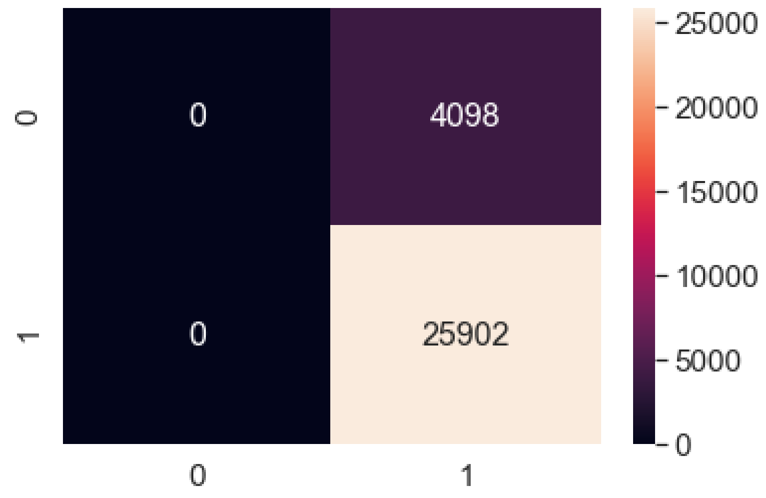


Training and testing with L1 regularization

In [20]:
```
myLogReg(c=0.0001,penalti='l1',xtrn=XtrainTFIDFV,xtst=XtestTFIDFV)
```

```
AUC score on test set: 50.000%
Accuracy on test set: 86.340%
Precision on test set: 86.340
Recall on test set: 100.000
F1-Score on test set: 92.669
Non Zero weights: 0
```


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



RandomSearch CV

In [102]:
```
logreg= LogisticRegression()
randSearchParam= {'C':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,
0.001,0.0005,0.0001]}
randsearch= RandomizedSearchCV(logreg,randSearchParam,cv=10,scoring='ro
c_auc',n_jobs=-1)
randsearch.fit(XtrainTFIDFV,Ytrain)
print(randsearch.best_estimator_)
print('Best Hyperparameter is ',randsearch.best_params_)
print('Best auc score is ',gridSearch.best_score_)
```

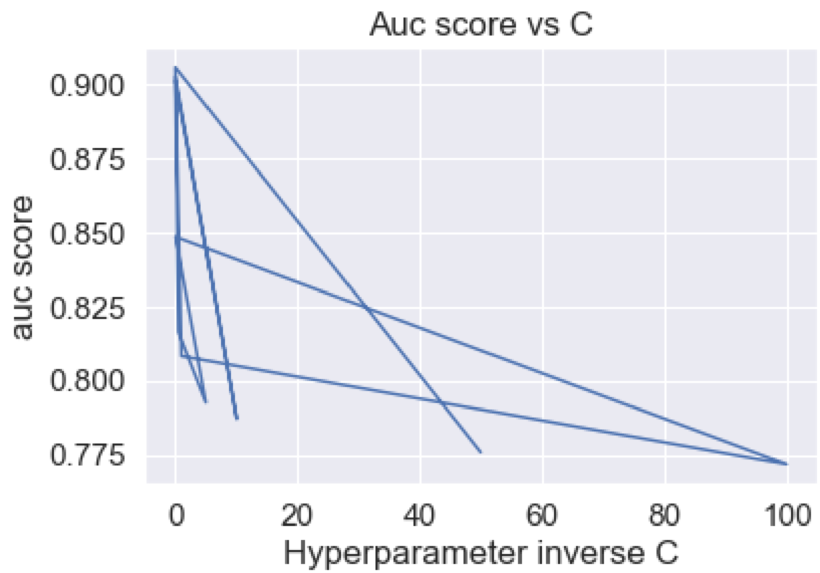
```
LogisticRegression(C=0.0005, class_weight=None, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
Best Hyperparameter is  {'C': 0.0005}
Best auc score is  0.9023941144816782
```

In [103]:
```
scores = [x[1] for x in randsearch.grid_scores_]
parameters= [x[0]['C'] for x in randsearch.grid_scores_]
```

```
plt.plot(parameters,scores)
plt.xlabel('Hyperparameter inverse C')
plt.ylabel('auc score')
plt.title('Auc score vs C')
```

```
C:\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:761:
DeprecationWarning: The grid_scores_ attribute was deprecated in versio
n 0.18 in favor of the more elaborate cv_results_ attribute. The grid_s
cores_ attribute will not be available from 0.20
  DeprecationWarning)
C:\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:761:
DeprecationWarning: The grid_scores_ attribute was deprecated in versio
n 0.18 in favor of the more elaborate cv_results_ attribute. The grid_s
cores_ attribute will not be available from 0.20
  DeprecationWarning)
```

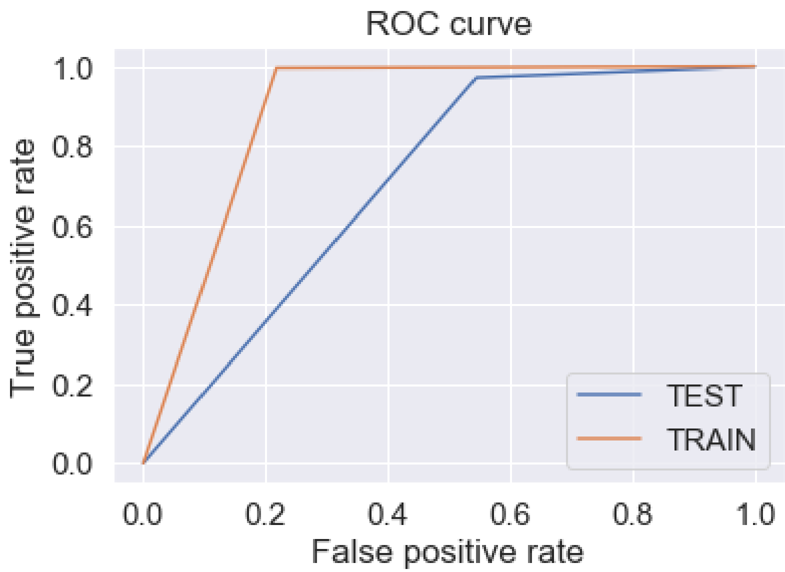
Out[103]: Text(0.5,1,'Auc score vs C')



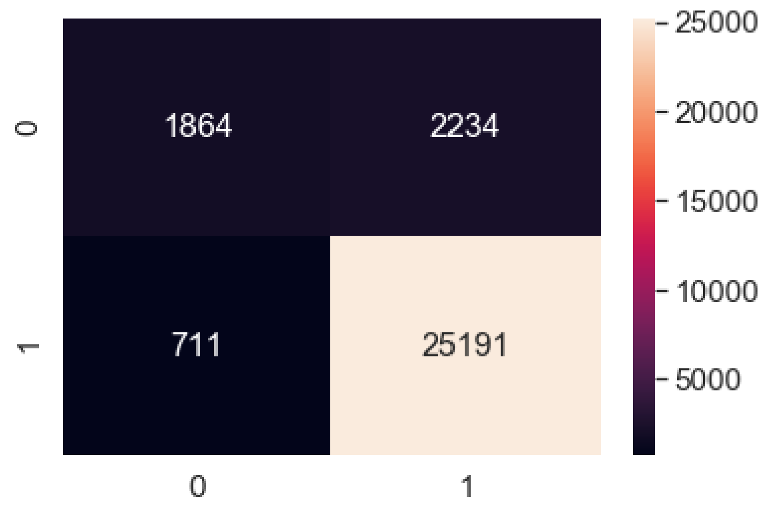
Training and testing with L2 regularization

In [21]:
```
myLogReg(c=0.0005,penalti='l2',xtrn=XtrainTFIDFV,xtst=XtestTFIDFV)
```

```
AUC score on test set: 71.370%
Accuracy on test set: 90.183%
Precision on test set: 91.854
Recall on test set: 97.255
F1-Score on test set: 94.477
Non Zero weights: 39730
```




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

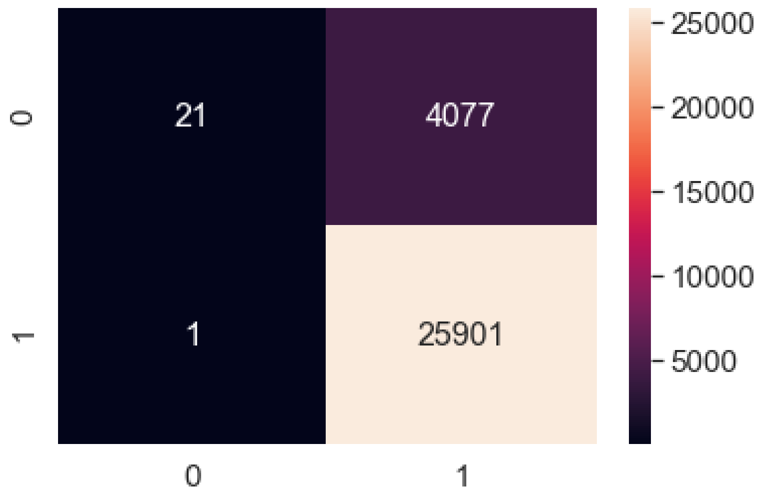


Training and testing with L1 regularization

In [22]:
```
myLogReg(c=0.0005,penalti='l1',xtrn=XtrainTFIDFV,xtst=XtestTFIDFV)
```

```
AUC score on test set: 50.254%
Accuracy on test set: 86.407%
Precision on test set: 86.400
Recall on test set: 99.996
F1-Score on test set: 92.702
Non Zero weights: 14
```


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



Feature importance L2 , TFIDF

In [79]:
```
clf = LogisticRegression(C= 0.0005, penalty= 'l2')
clf.fit(XtrainTFIDFV,Ytrain)
y_pred = clf.predict(XtestTFIDFV)
print("F1 score on test set: %0.3f%%"%(f1_score(Ytest, y_pred)*100))
print("AUC score on test set: %0.3f%%"%(roc_auc_score(Ytest, y_pred)*10
0))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("Total weights:",len(clf.coef_[0]))
```

```
F1 score on test set: 94.477%
AUC score on test set: 71.370%
Non Zero weights: 39730
Total weights: 39730
```

In [81]:
```
def show_most_informative_features(vectorizer, clf, n=10):
    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))

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

```
                        Negative
                Positive
________________________________________________________________________
_________________________
        -0.1648 disappoint                              0.3088  great

        -0.1291 worst                                   0.2649  love
```

-0.0977	aw	0.2256	best
-0.0942	stale	0.1806	good
-0.0919	terribl	0.1589	delici
-0.0917	horribl	0.1467	excel
-0.0915	unfortun	0.1441	perfect
-0.0905	bland	0.1351	favorit
-0.0865	wast	0.1333	nice
-0.0852	return	0.1294	find

Avg W2V vectorization

In [23]:
```
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")
```

```
10351
```

In [24]:

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

In [25]:

```
# 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 [26]:

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

In [27]:

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

Gridsearch CV

In [104]:

```
logreg=LogisticRegression()
```

```
gridSearchParam= {'C':[10**4,10**2,10,1,10**-1,10**-2,10**-4]}
gridSearch= GridSearchCV(logreg,gridSearchParam,cv=10,scoring='roc_auc'
,n_jobs=-1)
gridSearch.fit(XtrainW2VV,Ytrain)
print(gridSearch.best_estimator_)
print('Best Hyperparameter is ',gridSearch.best_params_)
print('Best auc score is ',gridSearch.best_score_)
```

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=Tr
ue,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=
1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.00
01,
          verbose=0, warm_start=False)
Best Hyperparameter is  {'C': 1}
Best auc score is  0.8846268174383332
```

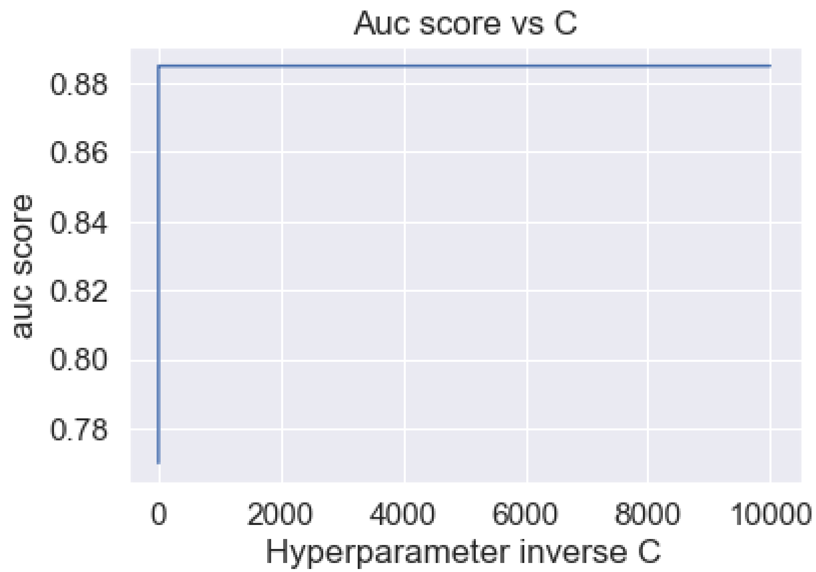
In [105]:

```
scores = [x[1] for x in gridSearch.grid_scores_]
parameters= gridSearch.param_grid['C']
plt.plot(parameters,scores)
plt.xlabel('Hyperparameter inverse C')
plt.ylabel('auc score')
plt.title('Auc score vs C')
```

```
C:\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:761:
DeprecationWarning: The grid_scores_ attribute was deprecated in versio
n 0.18 in favor of the more elaborate cv_results_ attribute. The grid_s
cores_ attribute will not be available from 0.20
  DeprecationWarning)
```

Out[105]:

```
Text(0.5,1,'Auc score vs C')
```

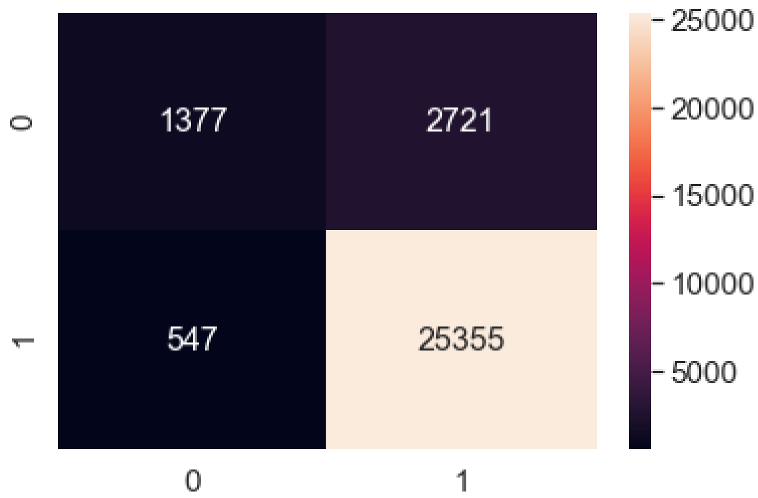


Training and testing with L2 regularization

In [28]:
```
myLogReg(c=1,penalti='l2',xtrn=XtrainW2VV,xtst=XtestW2VV)
```

```
AUC score on test set: 65.745%
Accuracy on test set: 89.107%
Precision on test set: 90.308
Recall on test set: 97.888
F1-Score on test set: 93.946
Non Zero weights: 50
```

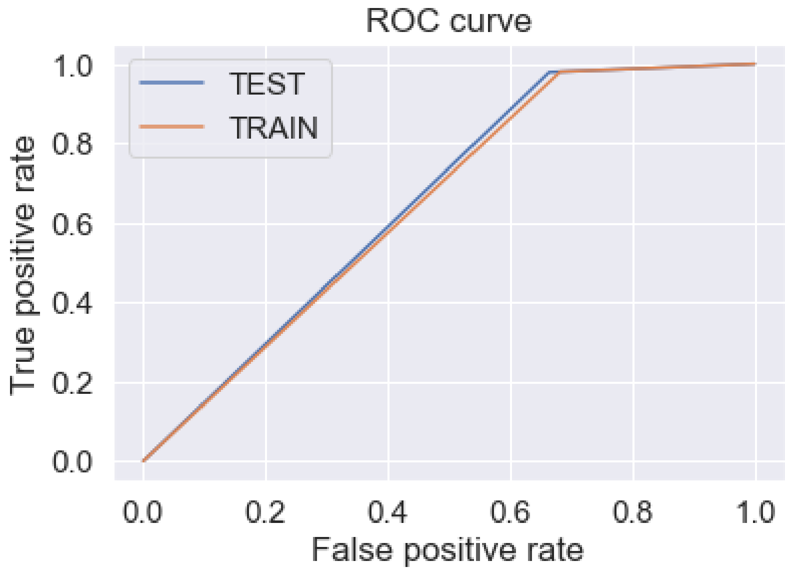

```
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```



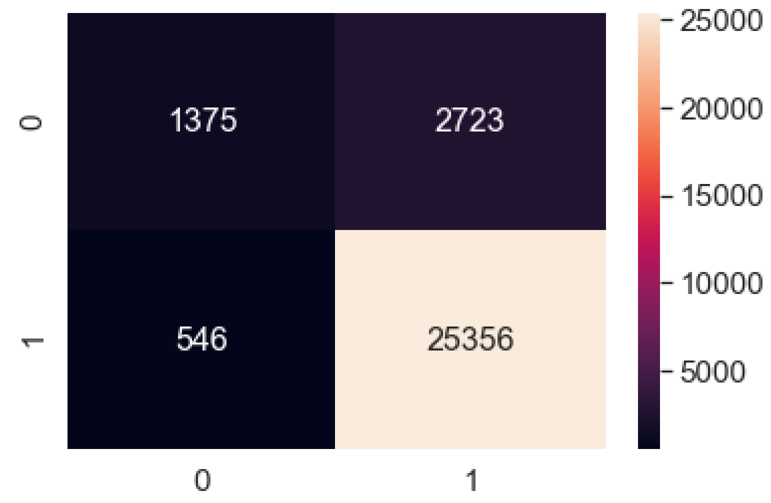
Training and testing with L1 regularization

In [29]:
```
myLogReg(c=1,penalti='l1',xtrn=XtrainW2VV,xtst=XtestW2VV)
```

```
AUC score on test set: 65.723%
Accuracy on test set: 89.103%
Precision on test set: 90.302
Recall on test set: 97.892
F1-Score on test set: 93.944
Non Zero weights: 50
```




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



RandomSearch CV

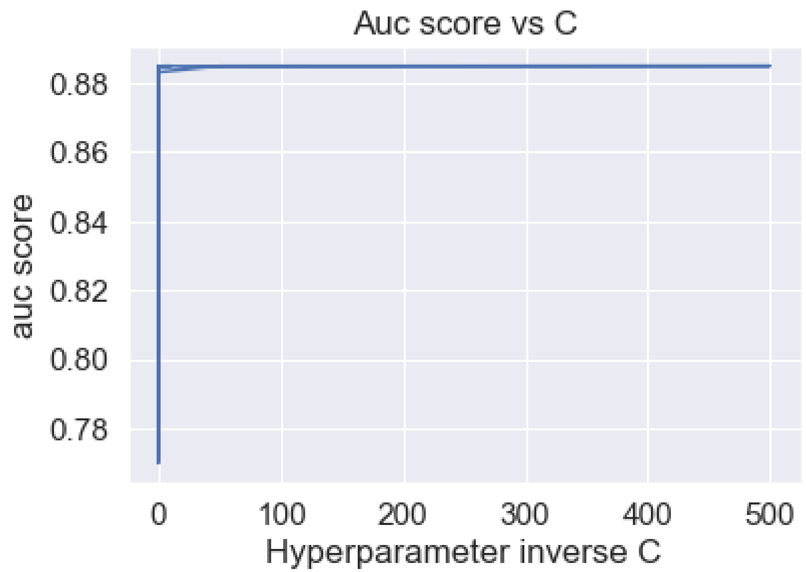
In [106]:
```
logreg= LogisticRegression()
randSearchParam= {'C':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,
0.001,0.0005,0.0001]}
randsearch= RandomizedSearchCV(logreg,randSearchParam,cv=10,scoring='ro
c_auc',n_jobs=-1)
randsearch.fit(XtrainW2VV,Ytrain)
print(randsearch.best_estimator_)
print('Best Hyperparameter is ',randsearch.best_params_)
print('Best auc score is ',gridSearch.best_score_)
```

```
LogisticRegression(C=5, class_weight=None, dual=False, fit_intercept=Tr
ue,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=
1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.00
01,
          verbose=0, warm_start=False)
Best Hyperparameter is  {'C': 5}
Best auc score is  0.8846268174383332
```

In [107]:
```
scores = [x[1] for x in randsearch.grid_scores_]
parameters= [x[0]['C'] for x in randsearch.grid_scores_]
plt.plot(parameters,scores)
plt.xlabel('Hyperparameter inverse C')
plt.ylabel('auc score')
plt.title('Auc score vs C')
```

```
C:\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:761:
DeprecationWarning: The grid_scores_ attribute was deprecated in versio
n 0.18 in favor of the more elaborate cv_results_ attribute. The grid_s
cores_ attribute will not be available from 0.20
  DeprecationWarning)
C:\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:761:
DeprecationWarning: The grid_scores_ attribute was deprecated in versio
n 0.18 in favor of the more elaborate cv_results_ attribute. The grid_s
cores_ attribute will not be available from 0.20
  DeprecationWarning)
```

Out[107]:
```
Text(0.5,1,'Auc score vs C')
```

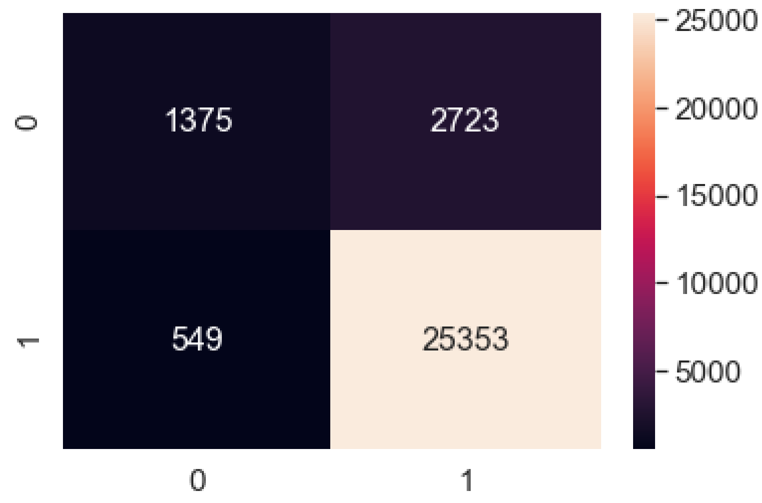


Training and testing with L2 regularization

In [30]:
```
myLogReg(c=5,penalti='l2',xtrn=XtrainW2VV,xtst=XtestW2VV)
```

```
AUC score on test set: 65.717%
Accuracy on test set: 89.093%
Precision on test set: 90.301
Recall on test set: 97.880
F1-Score on test set: 93.938
Non Zero weights: 50
```


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

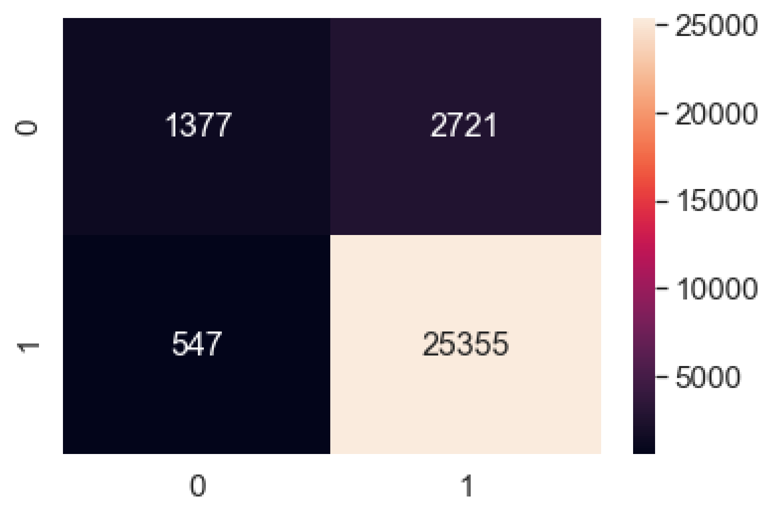


Training and testing with L1 regularization

In [31]:
```
myLogReg(c=5,penalti='l1',xtrn=XtrainW2VV,xtst=XtestW2VV)
```

```
AUC score on test set: 65.745%
Accuracy on test set: 89.107%
Precision on test set: 90.308
Recall on test set: 97.888
F1-Score on test set: 93.946
Non Zero weights: 50
```


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



tfidf-weighted avg w2v vectorization

In [32]:
```
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 [33]:
```
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(wor
d)]
                    #to reduce the computation we are
                    #dictionary[word] = idf value of word in whole cour
pus
                    #sent.count(word) = tf valeus of word in this revie
w
                    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 [34]:

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

In [35]:

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

Gridsearch CV

In [108]:

```
logreg=LogisticRegression()
gridSearchParam= {'C':[10**4,10**2,10,1,10**-1,10**-2,10**-4]}
gridSearch= GridSearchCV(logreg,gridSearchParam,cv=10,scoring='roc_auc'
,n_jobs=-1)
```

```
gridSearch.fit(XtrainTFIDFW2VV,Ytrain)
print(gridSearch.best_estimator_)
print('Best Hyperparameter is ',gridSearch.best_params_)
print('Best auc score is ',gridSearch.best_score_)
```

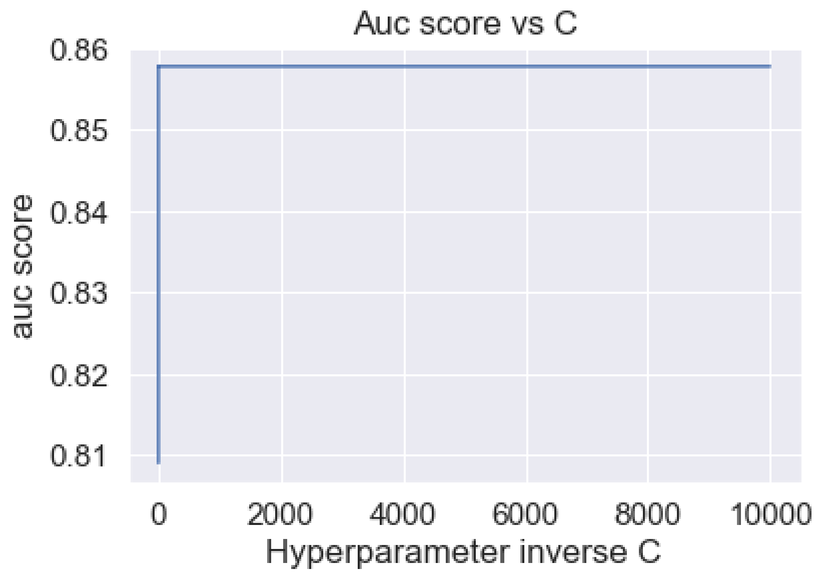
```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept
=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=
1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.00
01,
          verbose=0, warm_start=False)
Best Hyperparameter is  {'C': 0.01}
Best auc score is  0.8578684300155411
```

In [109]:

```
scores = [x[1] for x in gridSearch.grid_scores_]
parameters= gridSearch.param_grid['C']
plt.plot(parameters,scores)
plt.xlabel('Hyperparameter inverse C')
plt.ylabel('auc score')
plt.title('Auc score vs C')
```

```
C:\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:761:
DeprecationWarning: The grid_scores_ attribute was deprecated in versio
n 0.18 in favor of the more elaborate cv_results_ attribute. The grid_s
cores_ attribute will not be available from 0.20
  DeprecationWarning)
```

Out[109]: Text(0.5,1,'Auc score vs C')

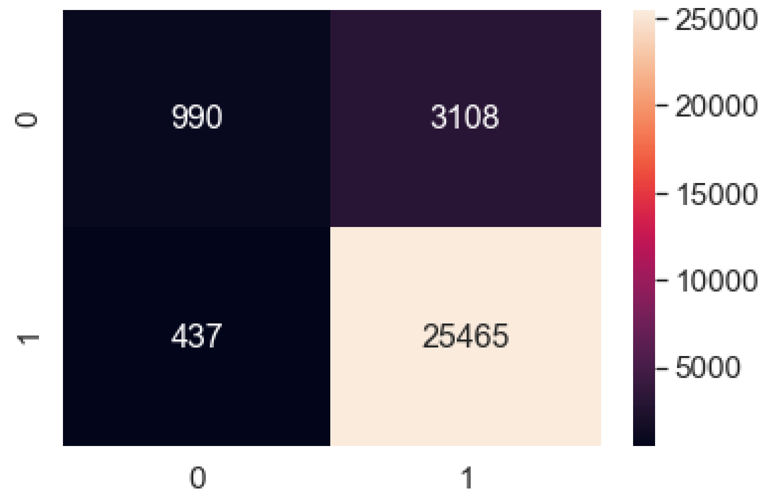


Training and testing with L2 regularization

In [36]:
```
myLogReg(c=0.01,penalti='l2',xtrn=XtrainTFIDFW2VV,xtst=XtestTFIDFW2VV)
```

```
AUC score on test set: 61.235%
Accuracy on test set: 88.183%
Precision on test set: 89.123
Recall on test set: 98.313
F1-Score on test set: 93.492
Non Zero weights: 50
```


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

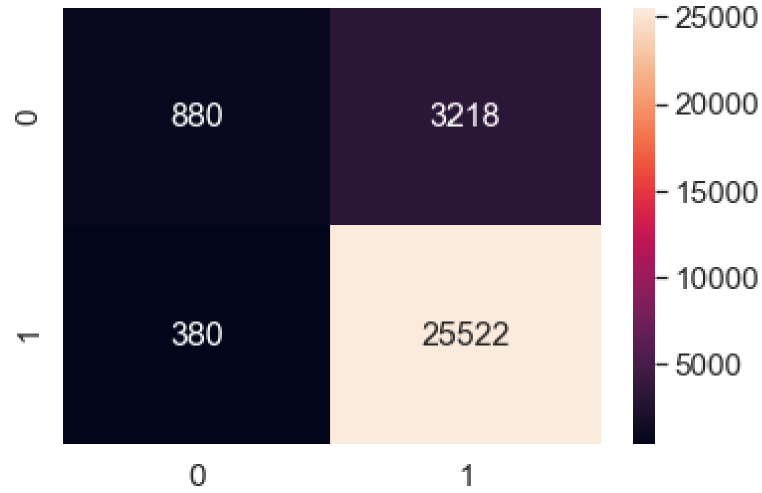


Training and testing with L1 regularization

In [37]:
```
myLogReg(c=0.01,penalti='l1',xtrn=XtrainTFIDFW2VV,xtst=XtestTFIDFW2VV)
```

```
AUC score on test set: 60.003%
Accuracy on test set: 88.007%
Precision on test set: 88.803
Recall on test set: 98.533
F1-Score on test set: 93.415
Non Zero weights: 43
```


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



RandomSearch CV

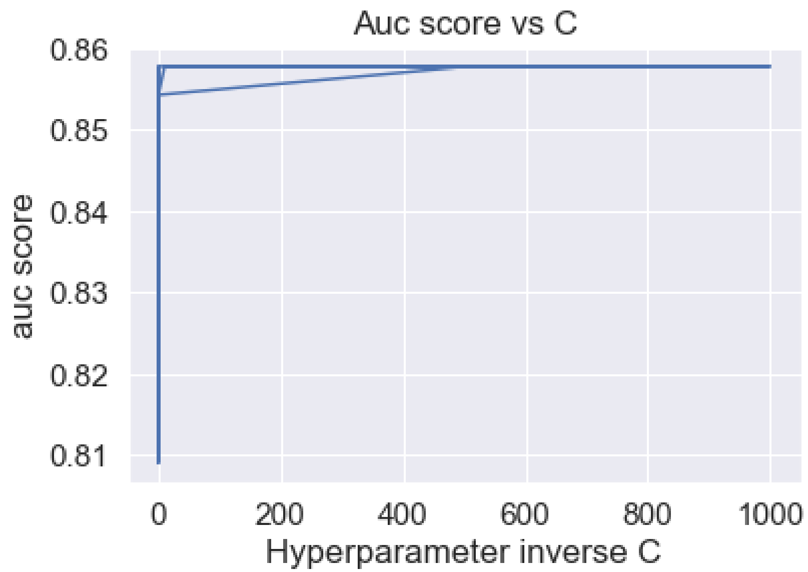
In [110]:
```
logreg= LogisticRegression()
randSearchParam= {'C':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,
0.001,0.0005,0.0001]}
randsearch= RandomizedSearchCV(logreg,randSearchParam,cv=10,scoring='ro
c_auc',n_jobs=-1)
randsearch.fit(XtrainTFIDFW2VV,Ytrain)
print(randsearch.best_estimator_)
print('Best Hyperparameter is ',randsearch.best_params_)
print('Best auc score is ',gridSearch.best_score_)
```

```
LogisticRegression(C=0.05, class_weight=None, dual=False, fit_intercept
=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=
1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.00
01,
          verbose=0, warm_start=False)
Best Hyperparameter is  {'C': 0.05}
Best auc score is  0.8578684300155411
```

In [111]:
```
scores = [x[1] for x in randsearch.grid_scores_]
parameters= [x[0]['C'] for x in randsearch.grid_scores_]
plt.plot(parameters,scores)
plt.xlabel('Hyperparameter inverse C')
plt.ylabel('auc score')
plt.title('Auc score vs C')
```

```
C:\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:761:
DeprecationWarning: The grid_scores_ attribute was deprecated in versio
n 0.18 in favor of the more elaborate cv_results_ attribute. The grid_s
cores_ attribute will not be available from 0.20
  DeprecationWarning)
C:\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:761:
DeprecationWarning: The grid_scores_ attribute was deprecated in versio
n 0.18 in favor of the more elaborate cv_results_ attribute. The grid_s
cores_ attribute will not be available from 0.20
  DeprecationWarning)
```

Out[111]: Text(0.5,1,'Auc score vs C')

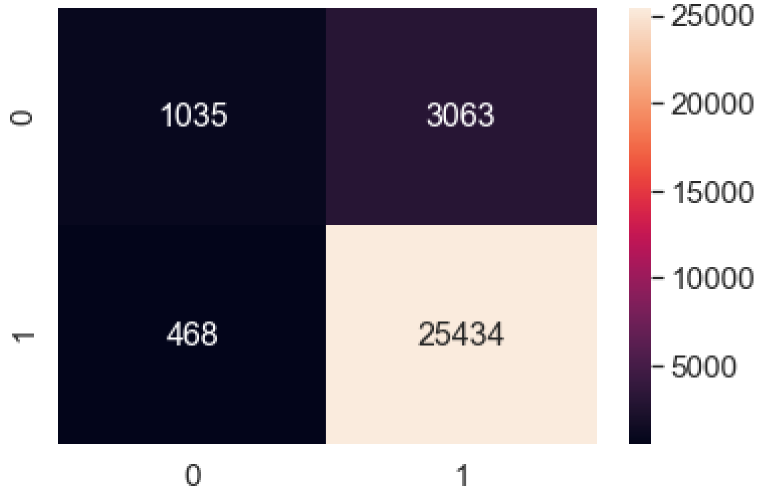


Training and testing with L2 regularization

In [38]:
```
myLogReg(c=0.05,penalti='l2',xtrn=XtrainTFIDFW2VV,xtst=XtestTFIDFW2VV)
```

```
AUC score on test set: 61.725%
Accuracy on test set: 88.230%
Precision on test set: 89.252
Recall on test set: 98.193
F1-Score on test set: 93.509
Non Zero weights: 50
```


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

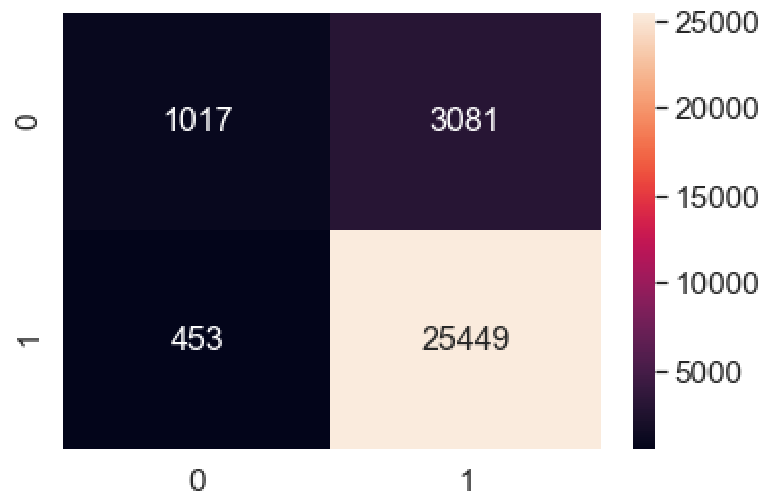


Training and testing with L1 regularization

In [39]:
```
myLogReg(c=0.05,penalti='l1',xtrn=XtrainTFIDFW2VV,xtst=XtestTFIDFW2VV)
```

```
AUC score on test set: 61.534%
Accuracy on test set: 88.220%
Precision on test set: 89.201
Recall on test set: 98.251
F1-Score on test set: 93.507
Non Zero weights: 49
```


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



Summary

GridSearchCV

Vectorizer & Regularizer	optimal C	AUC	Accuracy	Precision	recall	F1	weights
BOW L2	0.01	75.875	90.123	93.244	95.479	94.348	39730
BOW L1	0.01	72.363	91.100	92.072	98.143	95.010	3894
TFIDF L2	0.0001	64.545	89.287	89.966	98.587	94.080	39730
TFIDF L1	0.0001	50.000	86.340	86.340	100.000	92.669	0
W2V L2	1	65.745	89.107	90.308	97.888	93.946	50
W2V L1	1	65.723	89.103	90.302	97.892	93.944	50
TFIDF-W2v L2	0.01	61.235	88.183	89.123	98.313	93.492	50
TFIDF-W2V L1	0.01	60.003	88.007	88.803	98.533	93.415	43

RandomizedSearchCv

Vectorizer & Regularizer	optimal C	AUC	Accuracy	Precision	recall	F1	weights
BOW L2	0.001	73.976	90.977	92.567	97.367	94.907	39730
BOW L1	0.001	54.673	87.457	87.459	99.780	93.214	50
TFIDF L2	0.0005	71.370	90.183	91.854	97.255	94.477	39730

Vectorizer & Regularizer	optimal C	AUC	Accuracy	Precision	recall	F1	weights
TFIDF L1	0.0005	50.254	86.407	86.400	99.996	92.702	14
W2V L2	5	65.717	89.093	90.301	97.880	93.938	50
W2V L1	5	65.745	89.107	90.308	97.888	93.946	50
TFIDF-W2v L2	0.05	61.725	88.230	89.252	98.193	93.509	50
TFIDF-W2V L1	0.05	61.534	88.220	89.201	98.251	93.507	49

Conclusion

- The best model according to AUC metric is BOW with L2 regularizer and with optimal C of 0.01
- RandomizedSearchCV is as affective as GridsearchCV
- **Sparsity** is inversly proportional C and directly proportional to lambda
- We saw that L2 regularized logistic regression model from BOW vector was having **multicollinear** features
- Nonzero weights counts decreased as C decreased
- Below we can see how good our logistic regression model predicted 'Positive' and 'Negative' words

```
           Negative                             Positive
-----------------------------------------------------------------------
    -0.3722 disappoint                     0.8316   great
    -0.3075 worst                          0.6557   love
    -0.2317 unfortun                       0.6289   best
    -0.2197 stale                          0.4527   perfect
    -0.2184 aw                             0.4267   delici
    -0.2139 thought                        0.4120   good
    -0.2111 wast                           0.3943   nice
```

```
-0.2093 horribl                        0.3866  excel
-0.2080 tast                           0.3396  favorit
-0.2058 terribl                        0.3186  find
-0.2052 return                         0.3052  wonder
-0.2020 stick                          0.3019  amaz
-0.1956 bad                            0.2949  addict
-0.1954 bland                          0.2500  keep
-0.1937 would                          0.2460  refresh
```

In [95]:

```
print('end\n\n\n\n\n')
```

```
end
```