[1] Amazon Fine Food Reviews Analysis

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

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

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

Attribute Information:

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

Objective:

Given a review, determine whether the review is positive (Rating of 4 or 5) or negative (rating of 1 or 2).

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

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

[7.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score id above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
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 import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
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

from tqdm import tqdm
import os
warnings.filterwarnings("ignore")

%matplotlib inline
# sets the backend of matplotlib to the 'inline' backend:
#With this backend, the output of plotting commands is displayed inline within frontends li
#directly below the code cell that produced it. The resulting plots will then also be store

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

Exploratory Data Analysis

[7.1.2] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

In [2]:

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

con = sqlite3.connect("final.sqlite")#Loading Cleaned/ Preprocesed text that we did in Text

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

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

Out[2]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfu
0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
1	138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	
2	417839	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	
3	346055	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	1	
4	417838	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0	

In [3]:

```
final['Score'] = final['Score'].replace('positive',1)
final['Score'] = final['Score'].replace('negative',0)
```

In [4]:

```
final.duplicated(subset={"UserId","ProfileName","Time","Text"}).value_counts()
```

Out[4]:

```
False    364171
dtype: int64
```

In [5]:

```
final =  final.drop_duplicates(subset={"UserId","ProfileName","Time","Text"},keep="first")
```

In [6]:

```
final = final[final.HelpfulnessNumerator <= final.HelpfulnessDenominator]
print("Size of data",final['Id'].size," rows ")
```

```
Size of data 364171  rows
```

In [7]:

```
import re #Regex (Regualar Expr Operations)
#string = r"sdfsdfd" :- r is for raw string as Regex often uses \ backslashes(\w), so they

########Function to remove html tags from data
def striphtml(data):
    p = re.compile('<.*?>')#Find this kind of pattern
#     print(p.findall(data))#List of strings which follow the regex pattern
    return p.sub('',data) #Substitute nothing at the place of strings which matched the pat

striphtml('<a href="foo.com" class="bar">I Want This <b>text!</b></a><>')
```

Out[7]:

```
'I Want This text!'
```

In [8]:

```
def strippunc(data):
    p = re.compile(r'[?|!|\'|"|#|.|,|)|(|\|/|~|%|*]')
    return p.sub('',data)
strippunc("fsd*?~,,,( sdfsdfdsvv)#")
```

Out[8]:

```
'fsd sdfsdfdsvv'
```

In [9]:

```
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

stop = stopwords.words('english') #All the stopwords in English language
#excluding some useful words from stop words list as we doing sentiment analysis
excluding = ['against','not','don', "don't",'ain', 'aren', "aren't", 'couldn', "couldn't",
             'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'i
             'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't",'shouldn', "shoul
             "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
stop = [words for words in stop if words not in excluding]
print(stop)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you'r
e", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves',
'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'i
t', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselve
s', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'tho
se', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has',
'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'bu
t', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for',
'with', 'about', 'between', 'into', 'through', 'during', 'before', 'after',
'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'ove
r', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'w
here', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'o
ther', 'some', 'such', 'no', 'nor', 'only', 'own', 'same', 'so', 'than', 'to
o', 'very', 's', 't', 'can', 'will', 'just', 'should', "should've", 'now',
'd', 'll', 'm', 'o', 're', 've', 'y', 'ma', 'shan', "shan't"]
```

In [10]:

```
from nltk.stem import SnowballStemmer
snow = SnowballStemmer('english') #initialising the snowball stemmer
print("Stem/Root words of the some of the words using SnowBall Stemmer:")
print(snow.stem('tasty'))
print(snow.stem('tasteful'))
print(snow.stem('tastiest'))
print(snow.stem('delicious'))
print(snow.stem('amazing'))
print(snow.stem('amaze'))
print(snow.stem('initialize'))
print(snow.stem('fabulous'))
print(snow.stem('Honda City'))
print(snow.stem('unpleasant'

))
```

```
Stem/Root words of the some of the words using SnowBall Stemmer:
tasti
tast
tastiest
delici
amaz
amaz
initi
fabul
honda c
unpleas
```

In [11]:

```
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
for sent in final['Text'][2:3].values: #Running only for 2nd review
    filtered_sentence=[]
    print(sent) #Each review
    sent=striphtml(sent)# remove HTML tags
    sent=strippunc(sent)# remove Punctuation Symbols
    print(sent.split())
    for w in sent.split():
        print("================================>",w)
        if((w.isalpha()) and (len(w)>2)):#If it is a numerical value or character of lenght
            if(w.lower() not in stop):# If it is a stopword
                s=(snow.stem(w.lower())).encode('utf8') #Stemming the word using SnowBall S
                print("Selected: Stem Word->",s)
                filtered_sentence.append(s)
            else:
                print("Eliminated as it is a stopword")
                continue
        else:
            print("Eliminated as it is a numerical value or character of lenght less than 2
            continue
#     print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words

    final_string.append(str1)
    print("*************************************************************************")
    print("Finally selected words from the review:\n",final_string)
```

```
Beetlejuice is a well written movie ..... everything about it is excellen
t! From the acting to the special effects you will be delighted you chose
to view this movie.
['Beetlejuice', 'is', 'a', 'well', 'written', 'movie', 'everything', 'abou
t', 'it', 'is', 'excellent', 'From', 'the', 'acting', 'to', 'the', 'specia
l', 'effects', 'you', 'will', 'be', 'delighted', 'you', 'chose', 'to', 'vi
ew', 'this', 'movie']
================================> Beetlejuice
Selected: Stem Word-> b'beetlejuic'
================================> is
Eliminated as it is a numerical value or character of lenght less than 2
================================> a
Eliminated as it is a numerical value or character of lenght less than 2
================================> well
Selected: Stem Word-> b'well'
================================> written
Selected: Stem Word-> b'written'
================================> movie
Selected: Stem Word-> b'movi'
```

In [12]:

```
%%time
# Code takes a while to run as it needs to run on around 500k sentences.
i=0
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
for sent in final['Text'].values:
    filtered_sentence=[]
#     print(sent) #Each review
    sent=striphtml(sent)# remove HTML tags
    sent=strippunc(sent)# remove Punctuation Symbols
#     print(sent.split())
    for w in sent.split():
#         print("===============================>",w)
        if((w.isalpha()) and (len(w)>2)):#If it is a numerical value or character of lenght
            if(w.lower() not in stop):# If it is a stopword
                s=(snow.stem(w.lower())).encode('utf8') #Stemming the word using SnowBall S
                                                        #encoding as byte-string/utf-8
#                 print("Selected: Stem Word->",s)
                filtered_sentence.append(s)
                if (final['Score'].values)[i] == 'Positive':
                    all_positive_words.append(s) #list of all words used to describe positi
                if(final['Score'].values)[i] == 'Negative':
                    all_negative_words.append(s) #list of all words used to describe negati
            else:
#                 print("Eliminated as it is a stopword")
                continue
        else:
#             print("Eliminated as it is a numerical value or character of lenght less than
            continue
#     print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
            #encoding as byte-string/utf-8

    final_string.append(str1)
#     print("***********************************************************************")
#     print("Finally selected words from the review:\n",final_string)
    i+=1
```

Wall time: 18min 12s

In [13]:

```
%%time
# Code takes a while to run as it needs to run on around 500k sentences.
i=0
str1=' '
final_string_nostem=[]
s=''
for sent in final['Text'].values:
    filtered_sentence=[]
    sent=striphtml(sent)# remove HTML tags
    sent=strippunc(sent)# remove Punctuation Symbols
    for w in sent.split():
        if((w.isalpha()) and (len(w)>2)):#If it is a numerical value or character of lenght
            if(w.lower() not in stop):# If it is a stopword
                s=w.lower().encode('utf8') #encoding as byte-string/utf-8
            else:
                continue
        else:
            continue
    str1 = b" ".join(filtered_sentence)
    final_string_nostem.append(str1)
    i+=1
```

```
Wall time: 2min 6s
```

In [14]:

```
#Adding a column of CleanedText which displays the data after pre-processing of the review
final['CleanedText']=final_string
final['CleanedText_NoStem']=final_string_nostem
final.head(3)
```

Out[14]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpful
0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
1	138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	
2	417839	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	

In [ ]:

In [15]:

```
final.sort_values('Time',inplace=True)
final.head(10)
```

Out[15]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Help
0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
1	138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	
2	417839	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	
3	346055	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	1	
4	417838	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0	
5	346116	374422	B00004CI84	A1048CYU0OV4O8	Judy L. Eans	2	
6	346041	374343	B00004CI84	A1B2IZU1JLZA6	Wes	19	
7	70688	76882	B00002N8SM	A32DW342WBJ6BX	Buttersugar	0	
8	346141	374450	B00004CI84	ACJR7EQF9S6FP	Jeremy Robertson	2	
10	417883	451903	B00004CXX9	A2DEE7F9XKP3ZR	jerome	0	

In [16]:

```
final=final[:100000]
```

In [17]:

```
savetofile(final,"sample_lr")
```

In [18]:

```
final = openfromfile("sample_lr")
```

In [ ]:

```

```

[7.2.2] Bag of Words (BoW)

In [19]:

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

#Breaking into Train and test
X_train, X_test, y_train, y_test = train_test_split(final['CleanedText'].values,final['Scor

#Text -> Uni gram Vectors
uni_gram = CountVectorizer()
X_train = uni_gram.fit_transform(X_train)
#Normalize Data
X_train = preprocessing.normalize(X_train)

X_test = uni_gram.transform(X_test)
#Normalize Data
X_test = preprocessing.normalize(X_test)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler(with_mean=False)
X_train = sc.fit_transform(X_train)
X_test= sc.transform(X_test)
print("Test Data Size: ",X_test.shape)
print("Train Data Size: ",X_train.shape)
```

```
Test Data Size:  (30000, 64925)
Train Data Size:  (70000, 64925)
```

In [20]:

```
#To show how Time Series Split splits the data
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=10)
for train, cv in tscv.split(X_train):
#     print("%s %s" % (train, cv))
    print(X_train[train].shape, X_train[cv].shape)
```

```
(6370, 64925) (6363, 64925)
(12733, 64925) (6363, 64925)
(19096, 64925) (6363, 64925)
(25459, 64925) (6363, 64925)
(31822, 64925) (6363, 64925)
(38185, 64925) (6363, 64925)
(44548, 64925) (6363, 64925)
(50911, 64925) (6363, 64925)
(57274, 64925) (6363, 64925)
(63637, 64925) (6363, 64925)
```

GridSearchCV

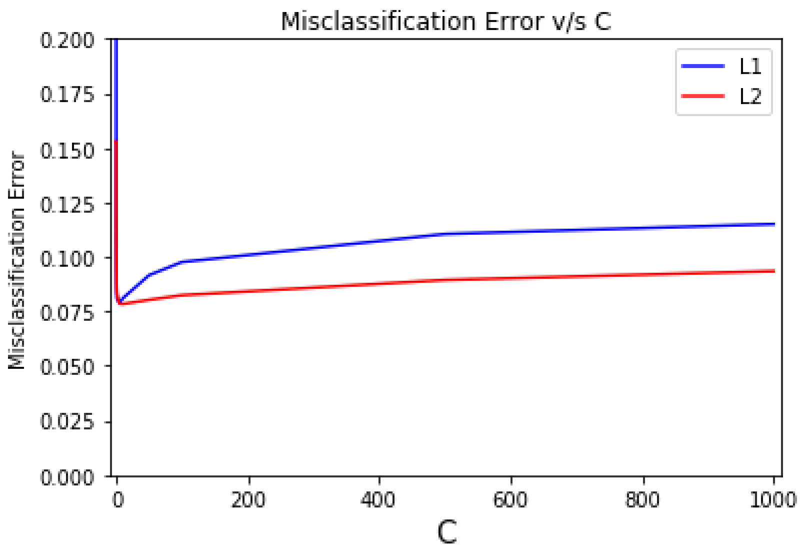
In [21]:

```
%time
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()
#params we need to try on classifier
param_grid = {'C':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001],
             'penalty':['l1','l2']}
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = GridSearchCV(clf,param_grid,cv=tscv,verbose=1,scoring='f1_micro')
gsv.fit(X_train,y_train)
savetofile(gsv,"Log Reg/gsv_bi")
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
Wall time: 0 ns
Fitting 10 folds for each of 30 candidates, totalling 300 fits

[Parallel(n_jobs=1)]: Done 300 out of 300 | elapsed: 75.3min finished

Best HyperParameter:  {'C': 0.05, 'penalty': 'l1'}
Best Accuracy: 91.79%
```

In [22]:

```
def plot_error_vs_c(gsv):
    x1=[]
    y1=[]
    x2=[]
    y2=[]
    for a in gsv.grid_scores_:
        if (a[0]['penalty']) == 'l1':
            y1.append(1-a[1])
            x1.append(a[0]['C'])
        else:
            y2.append(1-a[1])
            x2.append(a[0]['C'])
    plt.xlim(-10,1010)
    plt.ylim(0,0.2)
    plt.xlabel("C",fontsize=15)
    plt.ylabel("Misclassification Error")
    plt.title('Misclassification Error v/s C')
    plt.plot(x1,y1,'b',label="L1")
    plt.plot(x2,y2,'r',label="L2")
    plt.legend()
    plt.show()
gsv = openfromfile("Log Reg/gsv_uni")
plot_error_vs_c(gsv)
```



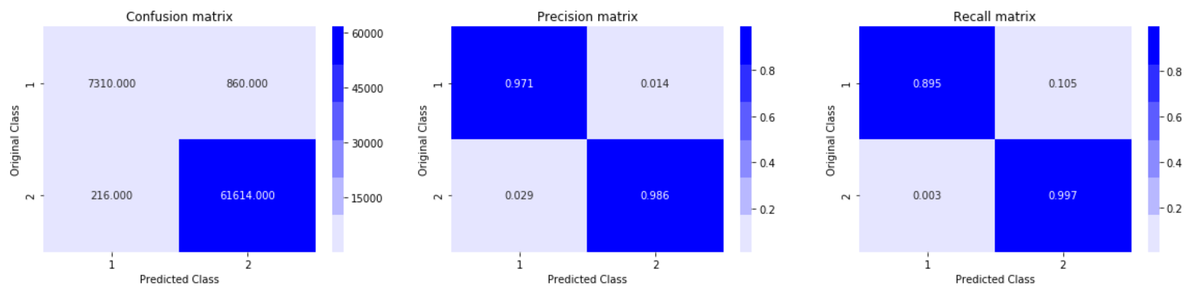
In [23]:

```
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

    A =(((C.T)/(C.sum(axis=1))).T)


    B =(C/C.sum(axis=0))

    plt.figure(figsize=(20,4))

    labels = [1,2]
    #representing A in heatmap format
    cmap=sns.light_palette("blue")
    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")

    plt.subplot(1, 3, 2)
    sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Precision matrix")

    plt.subplot(1, 3, 3)
    #representing B in heatmap format
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

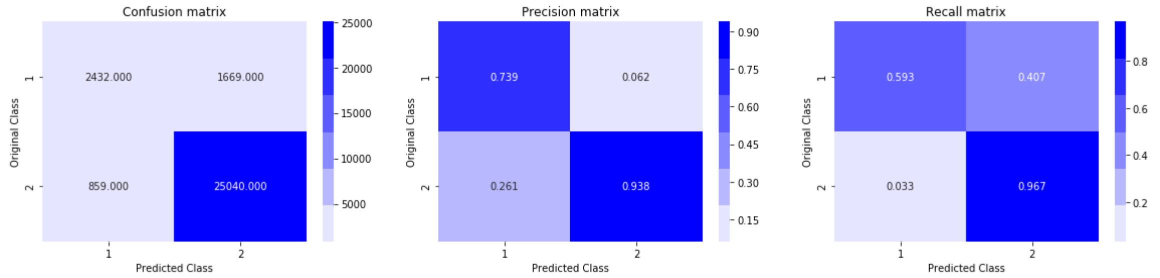
In [26]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score,recall_score,f1_score
clf = LogisticRegression(C= 0.05, penalty= 'l1')
clf.fit(X_train,y_train)
y_train_pred = clf.predict(X_train)
y_pred = clf.predict(X_test)
print("Accuracy on train set: %0.3f%%"%(accuracy_score(y_train, y_train_pred)*100))
print("Precision on train set: %0.3f"%(precision_score(y_train, y_train_pred)))
print("Recall on train set: %0.3f"%(recall_score(y_train, y_train_pred)))
print("F1-Score on train set: %0.3f"%(f1_score(y_train, y_train_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
plot_confusion_matrix(y_train, y_train_pred)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print('Confusion matrix for the model is:')
plot_confusion_matrix(y_test, y_pred)
```

```
Accuracy on train set: 98.463%
Precision on train set: 0.986
Recall on train set: 0.997
F1-Score on train set: 0.985
Non Zero weights: 13254
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```




```
Accuracy on test set: 91.573%
Precision on test set: 0.938
Recall on test set: 0.967
F1-Score on test set: 0.916
Non Zero weights: 13254
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

Confusion matrix for the model is:
```



Showing how sparsity increases as we increase lambda or decrease C when L1 Regularizer is used

In [27]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 1000, penalty= 'l1')
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Accuracy on test set: 84.600%
F1-Score on test set: 0.846
Non Zero weights: 19430
```

In [28]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 100, penalty= 'l1')
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Accuracy on test set: 84.740%
F1-Score on test set: 0.847
Non Zero weights: 17013
```

In [29]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 10, penalty= 'l1')
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Accuracy on test set: 85.710%
F1-Score on test set: 0.857
Non Zero weights: 17390
```

In [30]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 1, penalty= 'l1')
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Accuracy on test set: 88.003%
F1-Score on test set: 0.880
Non Zero weights: 18082
```

In [31]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 0.1, penalty= 'l1')
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Accuracy on test set: 90.887%
F1-Score on test set: 0.909
Non Zero weights: 15583
```

In [32]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 0.01, penalty= 'l1')
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Accuracy on test set: 91.660%
F1-Score on test set: 0.917
Non Zero weights: 5953
```

We can see how drastically the sparsity decreases from 19430 non-zero weights(@ C=1000) to only 5953 non-zero weights(@ C=0.01) when we use L1 Regularization

Using Randomized Search CV to find best parameters

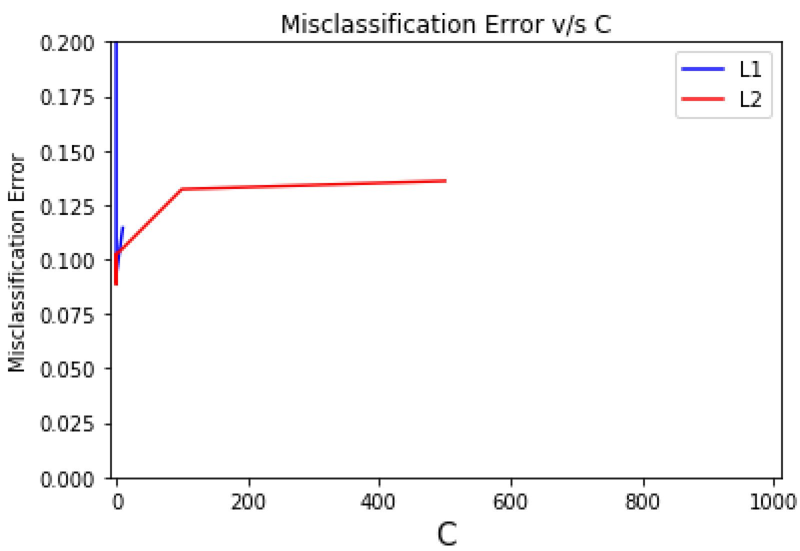
In [33]:

```
%time
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()
#params we need to try on classifier
param_grid = { 'C':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001],
              'penalty':['l1','l2']}
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = RandomizedSearchCV(clf,param_grid,cv=tscv,verbose=1,scoring='f1_micro')
gsv.fit(X_train,y_train)
savetofile(gsv,"Log Reg/gsv_uni_r")
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
Wall time: 0 ns
Fitting 10 folds for each of 10 candidates, totalling 100 fits

[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:  6.5min finished

Best HyperParameter:  {'penalty': 'l2', 'C': 0.001}
Best Accuracy: 91.14%
```

In [34]:

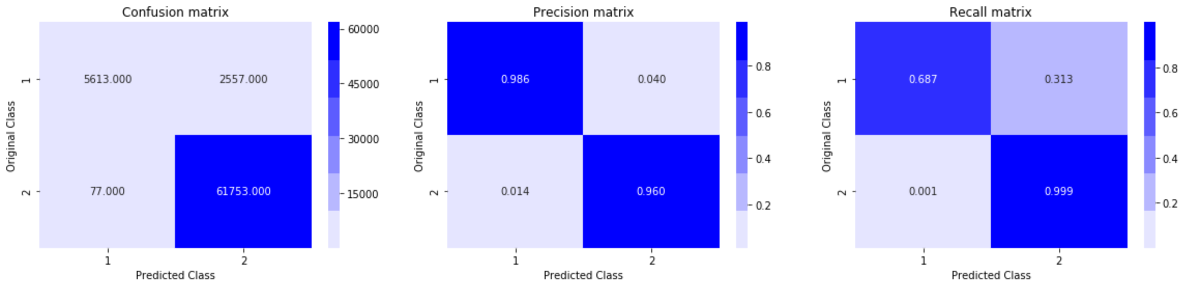
```
def plot_error_vs_c_r(gsv):
    x1=[]
    y1=[]
    x2=[]
    y2=[]
    for a in gsv.grid_scores_:
        if (a[0]['penalty']) == 'l1':
            y1.append(1-a[1])
            x1.append(a[0]['C'])
        else:
            y2.append(1-a[1])
            x2.append(a[0]['C'])

    ind1 = np.argsort(x1)
    x1=np.array(x1)
    y1=np.array(y1)
    ind2 = np.argsort(x2)
    x2=np.array(x2)
    y2=np.array(y2)
    plt.xlim(-10,1010)
    plt.ylim(0,0.2)
    plt.xlabel("C",fontsize=15)
    plt.ylabel("Misclassification Error")
    plt.title('Misclassification Error v/s C')
    plt.plot(x1[ind1],y1[ind1],'b',label="L1")
    plt.plot(x2[ind2],y2[ind2],'r',label="L2")
    plt.legend()
    plt.show()
gsv = openfromfile("Log Reg/gsv_uni_r")
plot_error_vs_c_r(gsv)
```



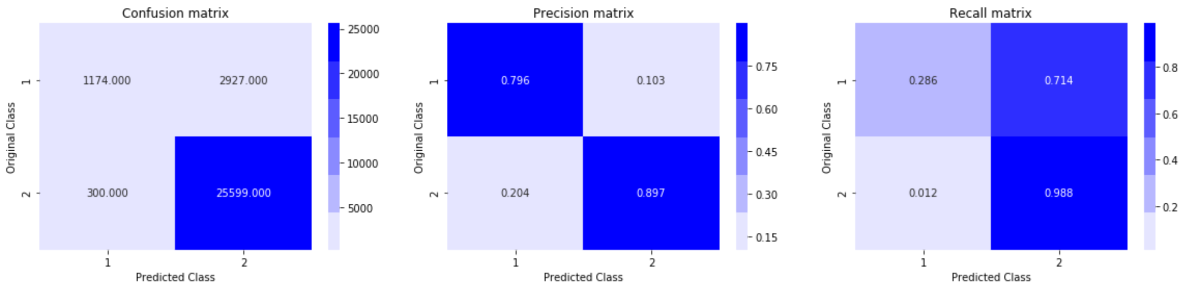
In [35]:

```
clf = LogisticRegression(C= 0.0001, penalty= 'l2')
clf.fit(X_train,y_train)
y_train_pred = clf.predict(X_train)
y_pred = clf.predict(X_test)
print("Accuracy on train set: %0.3f%%"%(accuracy_score(y_train, y_train_pred)*100))
print("Precision on train set: %0.3f"%(precision_score(y_train, y_train_pred)))
print("Recall on train set: %0.3f"%(recall_score(y_train, y_train_pred)))
print("F1-Score on train set: %0.3f"%(f1_score(y_train, y_train_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
plot_confusion_matrix(y_train, y_train_pred)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print('Confusion matrix for the model is:')
plot_confusion_matrix(y_test, y_pred)
```

```
Accuracy on train set: 96.237%
Precision on train set: 0.960
Recall on train set: 0.999
F1-Score on train set: 0.962
Non Zero weights: 64925
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```




```
Accuracy on test set: 89.243%
Precision on test set: 0.897
Recall on test set: 0.988
F1-Score on test set: 0.892
Non Zero weights: 64925
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

Confusion matrix for the model is:
```



[7.2.4] Bi-Grams

Motivation

Now that we have our list of words describing positive and negative reviews lets analyse them.

We begin analysis by getting the frequency distribution of the words as shown below

In [36]:

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

#Breaking into Train and test
X_train, X_test, y_train, y_test = train_test_split(final['CleanedText'].values,final['Scor

#taking one words and two consecutive words together
bi_gram = CountVectorizer(ngram_range=(1,2))
X_train = bi_gram.fit_transform(X_train)
#Normalize Data
X_train = preprocessing.normalize(X_train)

X_test = bi_gram.transform(X_test)
#Normalize Data
X_test = preprocessing.normalize(X_test)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler(with_mean=False)
X_train = sc.fit_transform(X_train)
X_test= sc.transform(X_test)
print("Train Data Size: ",X_train.shape)
print("Test Data Size: ",X_test.shape)
```

```
Train Data Size:  (70000, 1003102)
Test Data Size:  (30000, 1003102)
```

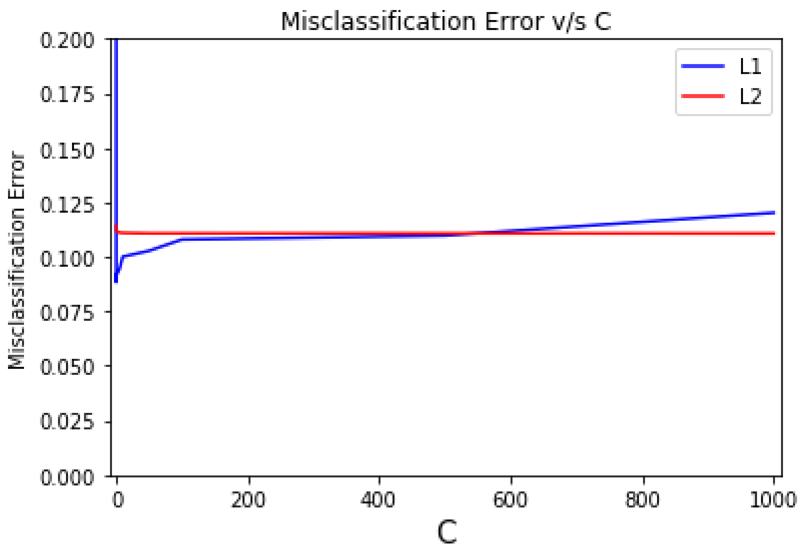
In [37]:

```
%time
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()
#params we need to try on classifier
param_grid = {'C':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001],
              'penalty':['l1','l2']}
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = GridSearchCV(clf,param_grid,cv=tscv,verbose=1,scoring='f1_micro')
gsv.fit(X_train,y_train)
savetofile(gsv,"Log Reg/gsv_bi")
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
Wall time: 0 ns
Fitting 10 folds for each of 30 candidates, totalling 300 fits

[Parallel(n_jobs=1)]: Done 300 out of 300 | elapsed: 42.9min finished

Best HyperParameter:  {'C': 0.01, 'penalty': 'l1'}
Best Accuracy: 91.16%
```

In [38]:

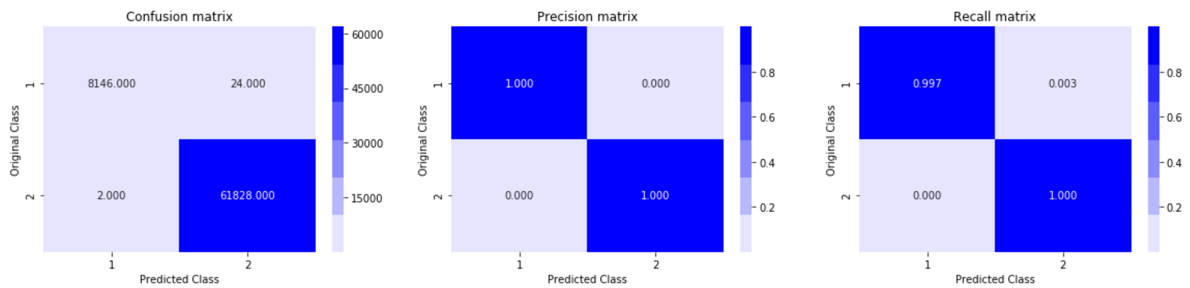
```
gsv = openfromfile("Log Reg/gsv_bi")
plot_error_vs_c(gsv)
```



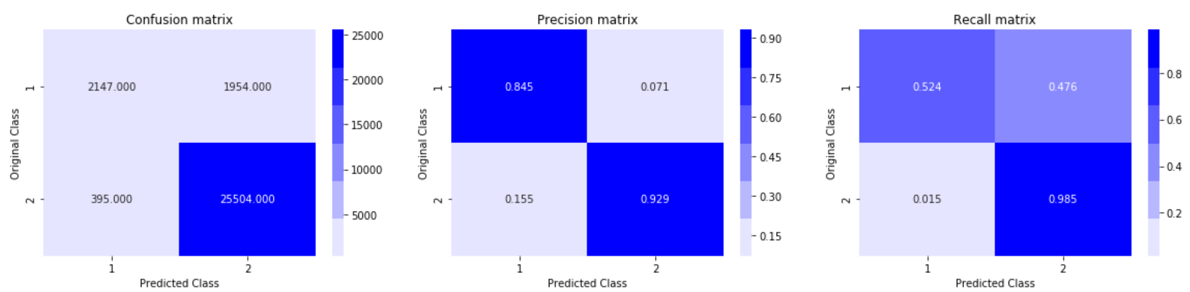
In [39]:

```
clf = LogisticRegression(C= 0.01, penalty= 'l1')
clf.fit(X_train,y_train)
y_train_pred = clf.predict(X_train)
y_pred = clf.predict(X_test)
print("Accuracy on train set: %0.3f%%"%(accuracy_score(y_train, y_train_pred)*100))
print("Precision on train set: %0.3f"%(precision_score(y_train, y_train_pred)))
print("Recall on train set: %0.3f"%(recall_score(y_train, y_train_pred)))
print("F1-Score on train set: %0.3f"%(f1_score(y_train, y_train_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
plot_confusion_matrix(y_train, y_train_pred)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print('Confusion matrix for the model is:')
plot_confusion_matrix(y_test, y_pred)
```

```
Accuracy on train set: 99.963%
Precision on train set: 1.000
Recall on train set: 1.000
F1-Score on train set: 1.000
Non Zero weights: 27198
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```




```
Accuracy on test set: 92.170%
Precision on test set: 0.929
Recall on test set: 0.985
F1-Score on test set: 0.922
Non Zero weights: 27198
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

Confusion matrix for the model is:
```



Showing how sparsity increases as we increase lambda or decrease C when L1 Regularizer is used

In [40]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 1000, penalty= 'l1')
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Accuracy on test set: 84.673%
F1-Score on test set: 0.847
Non Zero weights: 467782
```

In [41]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 100, penalty= 'l1')
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Accuracy on test set: 87.683%
F1-Score on test set: 0.877
Non Zero weights: 211933
```

In [42]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 10, penalty= 'l1')
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Accuracy on test set: 90.337%
F1-Score on test set: 0.903
Non Zero weights: 65503
```

In [43]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 1, penalty= 'l1')
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Accuracy on test set: 91.657%
F1-Score on test set: 0.917
Non Zero weights: 43566
```

In [44]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 0.1, penalty= 'l1')
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Accuracy on test set: 91.650%
F1-Score on test set: 0.916
Non Zero weights: 37455
```

In [45]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 0.01, penalty= 'l1')
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Accuracy on test set: 92.177%
F1-Score on test set: 0.922
Non Zero weights: 25153
```

We can see how drastically the sparsity decreases from 467782 non-zero weights(@ C=1000) to only 25153 non-zero weights(@ C=0.01) when we use L1 Regularization

Using Randomized Search CV to find best parameters

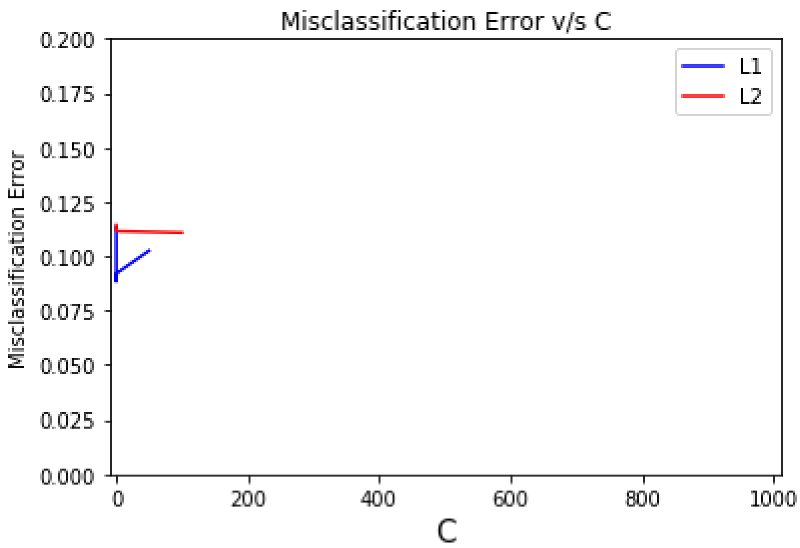
In [46]:

```
%time
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()
#params we need to try on classifier
param_grid = {'C':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001],
              'penalty':['l1','l2']}
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = RandomizedSearchCV(clf,param_grid,cv=tscv,verbose=1,scoring='f1_micro')
gsv.fit(X_train,y_train)
savetofile(gsv,"Log Reg/gsv_bi_r")
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
Wall time: 0 ns
Fitting 10 folds for each of 10 candidates, totalling 100 fits

[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed: 18.6min finished

Best HyperParameter:  {'penalty': 'l1', 'C': 0.01}
Best Accuracy: 91.16%
```

In [47]:

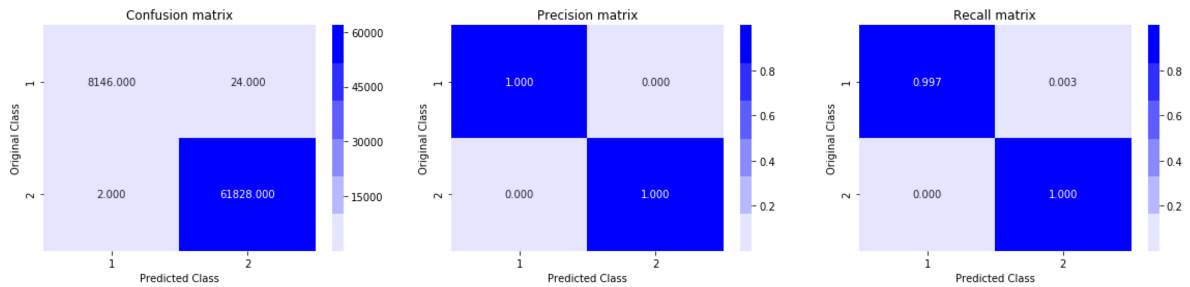
```
gsv = openfromfile("Log Reg/gsv_bi_r")
plot_error_vs_c_r(gsv)
```



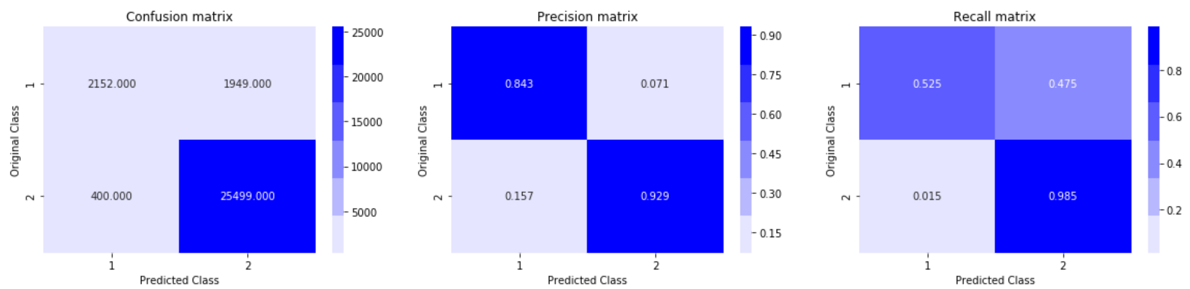
In [48]:

```
clf = LogisticRegression(C= 0.01, penalty= 'l1')
clf.fit(X_train,y_train)
y_train_pred = clf.predict(X_train)
y_pred = clf.predict(X_test)
print("Accuracy on train set: %0.3f%%"%(accuracy_score(y_train, y_train_pred)*100))
print("Precision on train set: %0.3f"%(precision_score(y_train, y_train_pred)))
print("Recall on train set: %0.3f"%(recall_score(y_train, y_train_pred)))
print("F1-Score on train set: %0.3f"%(f1_score(y_train, y_train_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
plot_confusion_matrix(y_train, y_train_pred)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print('Confusion matrix for the model is:')
plot_confusion_matrix(y_test, y_pred)
```

```
Accuracy on train set: 99.963%
Precision on train set: 1.000
Recall on train set: 1.000
F1-Score on train set: 1.000
Non Zero weights: 25528
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```




```
Accuracy on test set: 92.170%
Precision on test set: 0.929
Recall on test set: 0.985
F1-Score on test set: 0.922
Non Zero weights: 25528
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

Confusion matrix for the model is:
```



Perturbation Test

In [84]:

```
import scipy as sp
epsilon = sp.stats.distributions.norm.rvs(loc=0,scale=0.0001)
# Vector before the addition of epsilon
W_before_epsilon = clf.coef_

# Number of non zero elements in X_train_vec_standardized sparse matrix
no_of_non_zero = X_train.count_nonzero()

# Importing library to create a sparse matrix of epsilon
from scipy.sparse import csr_matrix

# Creating new sparse matrix with epsilon at same position of non-zero elements of X_train_
indices_X_train = X_train.indices
indptr_X_train = X_train.indptr

# Creating a list of same element with repetition
data = [epsilon] * no_of_non_zero
Shape = X_train.shape

# Creating sparse matrix
sparse_epsilon = csr_matrix((data,indices_X_train,indptr_X_train),shape=Shape,dtype=float)

# Add sparse_epsilon and X-train_vec_standardized to get a new sparse matrix with epsilon a
# non-zero element of X_train_vec_standardized
epsilon_train = X_train + sparse_epsilon

print(X_train.shape)
print(epsilon_train.shape)
```

```
(70000, 1003102)
(70000, 1003102)
```

In [86]:

```
epsilon_lr = LogisticRegression(penalty='l1', C=0.01, n_jobs=-1)
epsilon_lr.fit(epsilon_train,y_train)

# Vector after the addition of epsilon
W_after_epsilon = epsilon_lr.coef_

# Change in vectors after adding epsilon
change_vector = W_after_epsilon - W_before_epsilon

# Sort this change_vector array after making all the elements positive in ascending order t
sorted_change_vector = np.sort(np.absolute(change_vector))[:,::-1]

sorted_change_vector[0,0:20]
```

Out[86]:

```
array([0.0177024 , 0.01069277, 0.01045934, 0.01036312, 0.00995046,
       0.00994137, 0.00987692, 0.00977656, 0.0096977 , 0.00931577,
       0.00919783, 0.00919306, 0.00918648, 0.0091204 , 0.00908224,
       0.00895077, 0.008722  , 0.00870868, 0.00864846, 0.0085583 ])
```

In [88]:

```
absolute_weights = np.absolute(W_before_epsilon)
sorted_absolute_index = np.argsort(absolute_weights)[:,::-1]
top_index = sorted_absolute_index[0,0:20]

all_features = bi_gram.get_feature_names()
weight_values = clf.coef_

# Top 20 features are
print("Top 20 features with their weight values :")

for j in top_index:
    print("%12s\t--> \t%f"%(all_features[j],weight_values[0,j]))
```

```
Top 20 features with their weight values :
       great    -->     0.683128
        best    -->     0.472134
        love    -->     0.463073
        good    -->     0.369411
  disappoint    -->     -0.360893
      delici    -->     0.353219
     perfect    -->     0.307629
       excel    -->     0.272745
         not    -->     -0.231705
     favorit    -->     0.223190
        nice    -->     0.211375
       worst    -->     -0.207587
      wonder    -->     0.197292
   not worth    -->     -0.187525
        find    -->     0.184926
     terribl    -->     -0.176707
not disappoint  -->     0.170614
high recommend  -->     0.166500
       tasti    -->     0.165060
          aw    -->     -0.151817
```

[7.2.5] TF-IDF

In [90]:

```
%%time
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

#Breaking into Train and test
X_train, X_test, y_train, y_test = train_test_split(final['CleanedText'].values,final['Scor

tfidf = TfidfVectorizer()
X_train = tfidf.fit_transform(X_train)
#Normalize Data
X_train = preprocessing.normalize(X_train)

X_test = tfidf.transform(X_test)
#Normalize Data
X_test = preprocessing.normalize(X_test)
sc = StandardScaler(with_mean=False)
X_train = sc.fit_transform(X_train)
X_test= sc.transform(X_test)

print("Train Data Size: ",X_train.shape)
print("Test Data Size: ",X_test.shape)
```

```
Train Data Size:  (70000, 64925)
Test Data Size:  (30000, 64925)
Wall time: 9.48 s
```

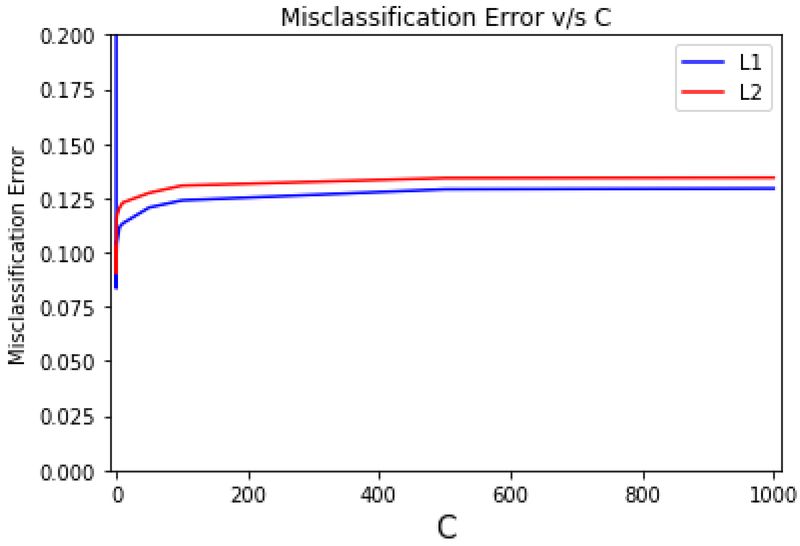
In [91]:

```
%time
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()

#params we need to try on classifier
param_grid = {'C':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001],
             'penalty':['l1','l2']}
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = GridSearchCV(clf,param_grid,cv=tscv,verbose=1,scoring='f1_micro')
gsv.fit(X_train,y_train)
savetofile(gsv,"Log Reg/gsv_tfidf")
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
Wall time: 0 ns
Fitting 10 folds for each of 30 candidates, totalling 300 fits

[Parallel(n_jobs=1)]: Done 300 out of 300 | elapsed: 101.7min finished

Best HyperParameter:  {'C': 0.05, 'penalty': 'l1'}
Best Accuracy: 91.65%
```

In [92]:

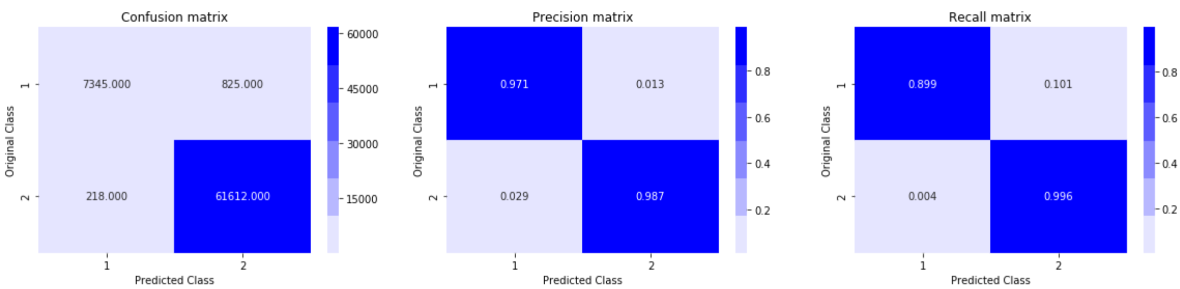
```
gsv = openfromfile("Log Reg/gsv_tfidf")
plot_error_vs_c(gsv)
```



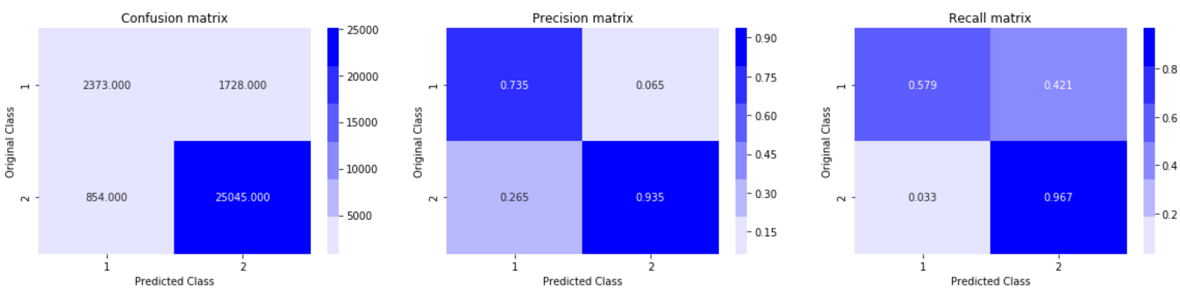
In [93]:

```
clf = LogisticRegression(C= 0.05, penalty= 'l1')
clf.fit(X_train,y_train)
y_train_pred = clf.predict(X_train)
y_pred = clf.predict(X_test)
print("Accuracy on train set: %0.3f%%"%(accuracy_score(y_train, y_train_pred)*100))
print("Precision on train set: %0.3f"%(precision_score(y_train, y_train_pred)))
print("Recall on train set: %0.3f"%(recall_score(y_train, y_train_pred)))
print("F1-Score on train set: %0.3f"%(f1_score(y_train, y_train_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
plot_confusion_matrix(y_train, y_train_pred)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print('Confusion matrix for the model is:')
plot_confusion_matrix(y_test, y_pred)
```

```
Accuracy on train set: 98.510%
Precision on train set: 0.987
Recall on train set: 0.996
F1-Score on train set: 0.985
Non Zero weights: 14008
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```




```
Accuracy on test set: 91.393%
Precision on test set: 0.935
Recall on test set: 0.967
F1-Score on test set: 0.914
Non Zero weights: 14008
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

Confusion matrix for the model is:
```



Showing how sparsity increases as we increase lambda or decrease C when L1 Regularizer is use

In [103]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 1000, penalty= 'l1')
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Accuracy on test set: 84.490%
F1-Score on test set: 0.845
Non Zero weights: 24250
```

In [105]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 100, penalty= 'l1')
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Accuracy on test set: 84.697%
F1-Score on test set: 0.847
Non Zero weights: 19858
```

In [106]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 10, penalty= 'l1')
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Accuracy on test set: 85.553%
F1-Score on test set: 0.856
Non Zero weights: 17594
```

In [107]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 1, penalty= 'l1')
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Accuracy on test set: 87.723%
F1-Score on test set: 0.877
Non Zero weights: 18521
```

In [108]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 0.1, penalty= 'l1')
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Accuracy on test set: 90.703%
F1-Score on test set: 0.907
Non Zero weights: 16211
```

In [109]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 0.01, penalty= 'l1')
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Accuracy on test set: 91.597%
F1-Score on test set: 0.916
Non Zero weights: 6124
```

We can see how drastically the sparsity increases from 24250 non-zero weights(@ C=1000) to only 3 non-zero weights(@ C=0.01) when we use L1 Regularization

Regularization Using Randomized Search CV to find best parameters

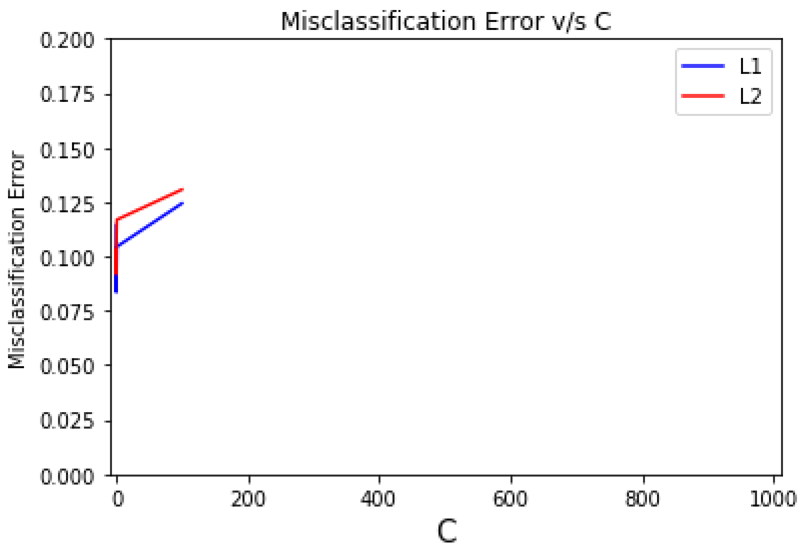
In [100]:

```
%time
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()
#params we need to try on classifier
param_grid = {'C':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001]
              ,'penalty':['l1','l2']}
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = RandomizedSearchCV(clf,param_grid,cv=tscv,verbose=1,scoring='f1_micro')
gsv.fit(X_train,y_train)
savetofile(gsv,"Log Reg/gsv_tfidf_r")
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
Wall time: 0 ns
Fitting 10 folds for each of 10 candidates, totalling 100 fits

[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:  7.7min finished

Best HyperParameter:  {'penalty': 'l1', 'C': 0.05}
Best Accuracy: 91.65%
```

In [101]:

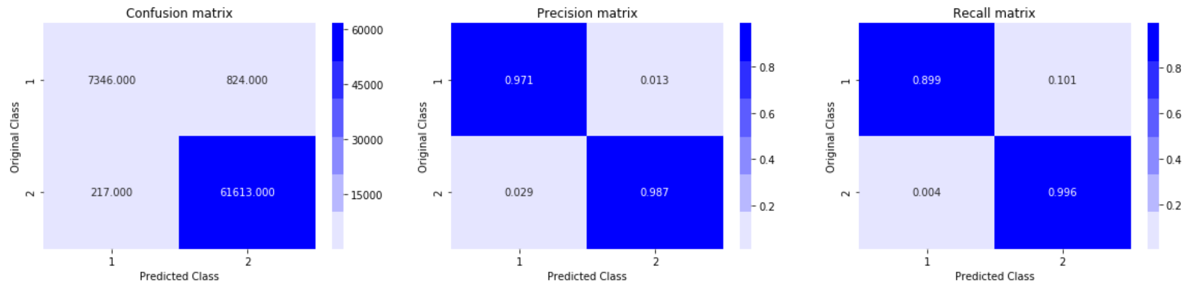
```
gsv = openfromfile("Log Reg/gsv_tfidf_r")
plot_error_vs_c_r(gsv)
```



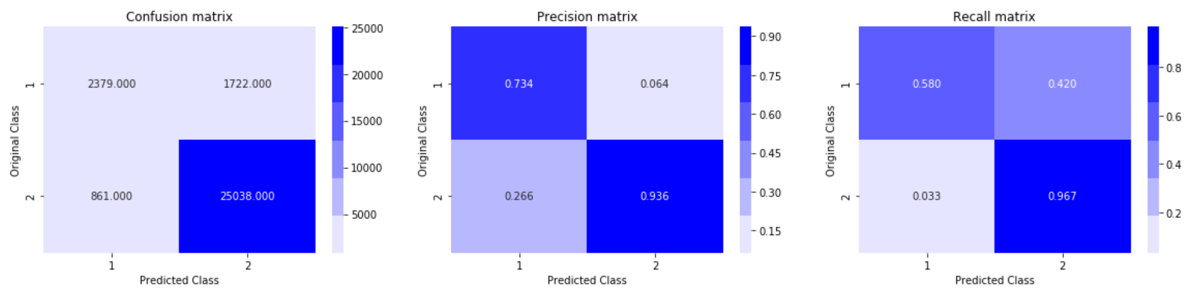
In [102]:

```
clf = LogisticRegression(C= 0.05, penalty= 'l1')
clf.fit(X_train,y_train)
y_train_pred = clf.predict(X_train)
y_pred = clf.predict(X_test)
print("Accuracy on train set: %0.3f%%"%(accuracy_score(y_train, y_train_pred)*100))
print("Precision on train set: %0.3f"%(precision_score(y_train, y_train_pred)))
print("Recall on train set: %0.3f"%(recall_score(y_train, y_train_pred)))
print("F1-Score on train set: %0.3f"%(f1_score(y_train, y_train_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
plot_confusion_matrix(y_train, y_train_pred)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print('Confusion matrix for the model is:')
plot_confusion_matrix(y_test, y_pred)
```

```
Accuracy on train set: 98.513%
Precision on train set: 0.987
Recall on train set: 0.996
F1-Score on train set: 0.985
Non Zero weights: 13903
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```




```
Accuracy on test set: 91.390%
Precision on test set: 0.936
Recall on test set: 0.967
F1-Score on test set: 0.914
Non Zero weights: 13903
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

Confusion matrix for the model is:
```



Perturbation Test

In [110]:

```
import scipy as sp
epsilon = sp.stats.distributions.norm.rvs(loc=0,scale=0.0001)
# Vector before the addition of epsilon
W_before_epsilon = clf.coef_

# Number of non zero elements in X_train_vec_standardized sparse matrix
no_of_non_zero = X_train.count_nonzero()

# Importing library to create a sparse matrix of epsilon
from scipy.sparse import csr_matrix

# Creating new sparse matrix with epsilon at same position of non-zero elements of X_train_
indices_X_train = X_train.indices
indptr_X_train = X_train.indptr

# Creating a list of same element with repetition
data = [epsilon] * no_of_non_zero
Shape = X_train.shape

# Creating sparse matrix
sparse_epsilon = csr_matrix((data,indices_X_train,indptr_X_train),shape=Shape,dtype=float)

# Add sparse_epsilon and X-train_vec_standardized to get a new sparse matrix with epsilon a
# non-zero element of X_train_vec_standardized
epsilon_train = X_train + sparse_epsilon

print(X_train.shape)
print(epsilon_train.shape)
```

```
(70000, 64925)
(70000, 64925)
```

In [111]:

```
epsilon_lr = LogisticRegression(penalty='l1', C=0.05, n_jobs=-1)
epsilon_lr.fit(epsilon_train,y_train)

# Vector after the addition of epsilon
W_after_epsilon = epsilon_lr.coef_

# Change in vectors after adding epsilon
change_vector = W_after_epsilon - W_before_epsilon

# Sort this change_vector array after making all the elements positive in ascending order t
sorted_change_vector = np.sort(np.absolute(change_vector))[:,::-1]

sorted_change_vector[0,0:20]
```

Out[111]:

```
array([0.20545409, 0.18718968, 0.16820735, 0.16599216, 0.14544775,
       0.14533518, 0.13881392, 0.13196279, 0.12740095, 0.12252001,
       0.12242056, 0.1218007 , 0.11695892, 0.11260441, 0.11123762,
       0.11024841, 0.10760662, 0.10624811, 0.10495961, 0.10492818])
```

In [112]:

```
absolute_weights = np.absolute(W_before_epsilon)
sorted_absolute_index = np.argsort(absolute_weights)[:,::-1]
top_index = sorted_absolute_index[0,0:20]

all_features =tfidf.get_feature_names()
weight_values = clf.coef_

# Top 20 features are
print("Top 20 features with their weight values :")

for j in top_index:
    print("%12s\t--> \t%f"%(all_features[j],weight_values[0,j]))
```

```
Top 20 features with their weight values :
       great    -->     0.684374
        best    -->     0.519821
        love    -->     0.461286
      delici    -->     0.393307
     perfect    -->     0.367119
         not    -->     -0.361515
        good    -->     0.331659
       excel    -->     0.313476
        nice    -->     0.253738
     favorit    -->     0.243472
  disappoint    -->     -0.234664
        find    -->     0.219518
      wonder    -->     0.217783
       worst    -->     -0.208236
       tasti    -->     0.177924
        amaz    -->     0.176945
        keep    -->     0.169720
      addict    -->     0.156525
     terribl    -->     -0.152146
       yummi    -->     0.145016
```

In [ ]:

In [ ]:

[7.2.6] Word2Vec

In [113]:

```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need
is_your_ram_gt_16g=False
want_to_read_sub_set_of_google_w2v = True
want_to_read_whole_google_w2v = True
if not is_your_ram_gt_16g:
    if want_to_read_sub_set_of_google_w2v and  os.path.isfile('google_w2v_for_amazon.pkl'):
        with open('google_w2v_for_amazon.pkl', 'rb') as f:
            # model is dict object, you can directly access any word vector using model[wor
            model = pickle.load(f)
else:
    if want_to_read_whole_google_w2v and os.path.isfile('GoogleNews-vectors-negative300.bin
        model = KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', bin

# print("the vector representation of word 'computer'",model.wv['computer'])
# print("the similarity between the words 'woman' and 'man'",model.wv.similarity('woman', '
# print("the most similar words to the word 'woman'",model.wv.most_similar('woman'))
# this will raise an error
# model.wv.most_similar('tasti')  # "tasti" is the stemmed word for tasty, tastful
```

In [19]:

```
final_string = []
for sent in final['CleanedText'].values:
    sent = str(sent)
    sentence=[]
#     print(sent)
    for word in sent.split():
#         print(word)
        sentence.append(word)
#         print(sentence)
    final_string.append(sentence)
```

In [20]:

```
%%time
# Train your own Word2Vec model using your own text corpus
import gensim

w2v_model=gensim.models.Word2Vec(final_string,min_count=5,size=50, workers=-1)
#min-count: Ignoring the words which occurs less than 5 times
#size:Creating vectors of size 50 for each word
#workers: Use these many worker threads to train the model (faster training with multicore
```

```
Wall time: 4.37 s
```

In [21]:

```
w2v_model.save('w2vmodel')
```

In [22]:

```
w2v_model = gensim.models.Word2Vec.load('w2vmodel')
```

In [23]:

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

In [24]:

```
w2v_vocub = w2v_model.wv.vocab
len(w2v_vocub)
```

Out[24]:

```
16909
```

In [25]:

```
w2v_model.wv.most_similar('like')
```

Out[25]:

```
[('alabama', 0.48587822914123535),
 ('creek', 0.48546063899993896),
 ('crown', 0.47647255659103394),
 ('coffeenot', 0.47441166639328003),
 ('rachel', 0.46372172236442566),
 ('cleanser', 0.4460267424583435),
 ('upright', 0.4457867443561554),
 ('fazer', 0.4450550377368927),
 ('elder', 0.44065648317337036),
 ('twirl', 0.4406422972679138)]
```

avg w2c

In [26]:

```
%%time
avg_vec = [] #List to store all the avg w2vec's
for sent in final_string[0:1]:
    cnt = 0 #to count no of words in each reviews
    sent_vec = np.zeros(50) #Initializing with zeroes
    print("sent:",sent)
    for word in sent:
        try:
            wvec = w2v_model.wv[word] #Vector of each using w2v model
            print("wvec:",wvec)
            sent_vec += wvec #Adding the vectors
            cnt += 1
        except:
            pass #When the word is not in the dictionary then do nothing
    print("sent_vec:",sent_vec)
    a_vec =sent_vec / cnt #Taking average of vectors sum of the particular review
    print("avg_vec:",a_vec)
    avg_vec.append(a_vec) #Storing the avg w2vec's for each review
    print("*********************************************************************")
```

```
sent: ["b'witti", 'littl', 'book', 'make', 'son', 'laugh', 'loud', 'reci
t', 'car', 'drive', 'along', 'alway', 'sing', 'refrain', 'hes', 'learn',
'whale', 'india', 'droop', 'love', 'new', 'word', 'book', 'introduc', 'sil
li', 'classic', 'book', 'will', 'bet', 'son', 'still', 'abl', 'recit', 'me
mori', "colleg'"]
wvec: [ 0.00927044  0.00268178 -0.00347884 -0.00993292  0.00485403 -0.0069
3354
 -0.00206989  0.00108701  0.00409818 -0.00782471  0.00949951  0.00545466
 -0.00430164 -0.00083933 -0.00801887 -0.00657124  0.00879726  0.00099144
  0.00530099 -0.00108125 -0.00536537  0.00178298 -0.00106748  0.00753624
 -0.00535857 -0.00839065 -0.00351354  0.00754232  0.00982998  0.00291407
 -0.00958644 -0.00290739  0.00253408 -0.00085875 -0.00593013 -0.00779566
  0.0021525  -0.00535808 -0.00353203 -0.00941841 -0.00127577 -0.00498264
 -0.00182707  0.00336423  0.00300162  0.00791903  0.00984028 -0.00442397
 -0.00916059  0.00627346]
wvec: [-5.4075806e-03 -5.0162775e-03  5.9144095e-05 -3.2706622e-03
 -6.0786502e-03  4.4440157e-03  7.0022922e-03  5.0159995e-03
 -5.4730033e-03 -6.7887851e-03  1.5975880e-03  5.9613758e-03
 -2.2616868e-03 -1.8917979e-03 -4.5878594e-03  6.1382530e-03
```

In [27]:

```
%%time
np.seterr(divide='ignore', invalid='ignore')
avg_vec = [] #List to store all the avg w2vec's
for sent in final_string:
    cnt = 0 #to count no of words in each reviews
    sent_vec = np.zeros(50) #Initializing with zeroes
    for word in sent:
        try:
            wvec = w2v_model.wv[word] #Vector of each using w2v model
            sent_vec += wvec #Adding the vectors
            cnt += 1
        except:
            pass #When the word is not in the dictionary then do nothing
    sent_vec /= cnt #Taking average of vectors sum of the particular review
    avg_vec.append(sent_vec) #Storing the avg w2vec's for each review
    #print("*******************************************************************")
    # Average Word2Vec
```

```
Wall time: 41.3 s
```

In [28]:

```
savetofile(avg_vec,"avg_w2v_vec")
```

In [29]:

```
avg_vec = openfromfile("avg_w2v_vec")
```

In [30]:

```
avg_vec = np.array(avg_vec)
avg_vec.shape
```

Out[30]:

```
(100000, 50)
```

In [31]:

```
np.isnan(avg_vec).any()
```

Out[31]:

```
True
```

In [32]:

```
mask = ~np.any(np.isnan(avg_vec), axis=1)
# print(mask)
avg_vec_new = avg_vec[mask]
final_sample_new = final['Score'][mask]
print(avg_vec_new.shape)
print(final_sample_new.shape)
```

```
(99996, 50)
(99996,)
```

In [174]:

```
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
#Normalizing the data
avg_vec_norm = preprocessing.normalize(avg_vec_new)

#Not shuffling the data as we want it on time basis
X_train, X_test, y_train, y_test = train_test_split(avg_vec_norm,final_sample_new.values,te
sc = StandardScaler(with_mean=False)
X_train = sc.fit_transform(X_train)
X_test= sc.transform(X_test)

print("Train Data Size: ",X_train.shape)
print("Test Data Size: ",X_test.shape)
```

```
Train Data Size:  (69997, 50)
Test Data Size:  (29999, 50)
```

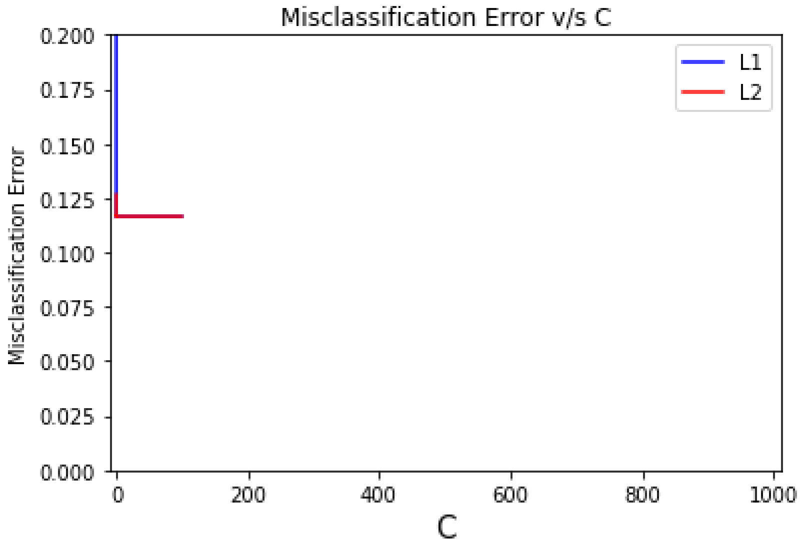
In [131]:

```
%time
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()
#params we need to try on classifier
param_grid = {'C':[100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001],
             'penalty':['l1','l2']}
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = GridSearchCV(clf,param_grid,cv=tscv,verbose=1,scoring='f1_micro')
gsv.fit(X_train,y_train)
savetofile(gsv,"Log Reg/gsv_w2v")
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
Wall time: 0 ns
Fitting 10 folds for each of 26 candidates, totalling 260 fits

[Parallel(n_jobs=1)]: Done 260 out of 260 | elapsed:  1.2min finished

Best HyperParameter:  {'C': 0.1, 'penalty': 'l2'}
Best Accuracy: 88.34%
```

In [132]:

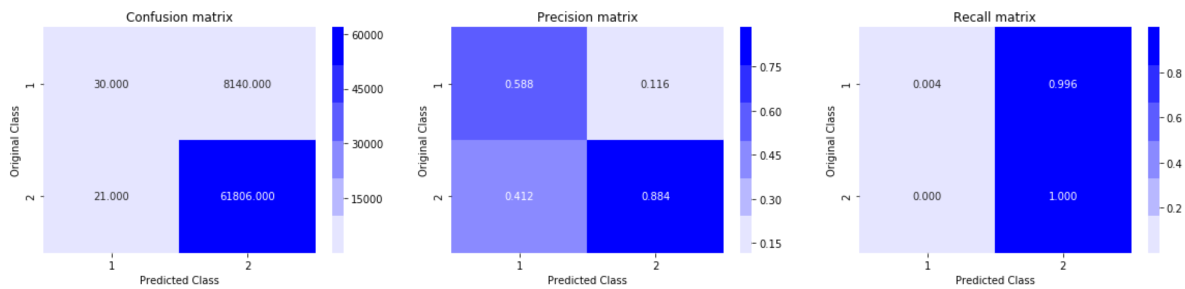
```
gsv = openfromfile("Log Reg/gsv_w2v")
plot_error_vs_c(gsv)
```



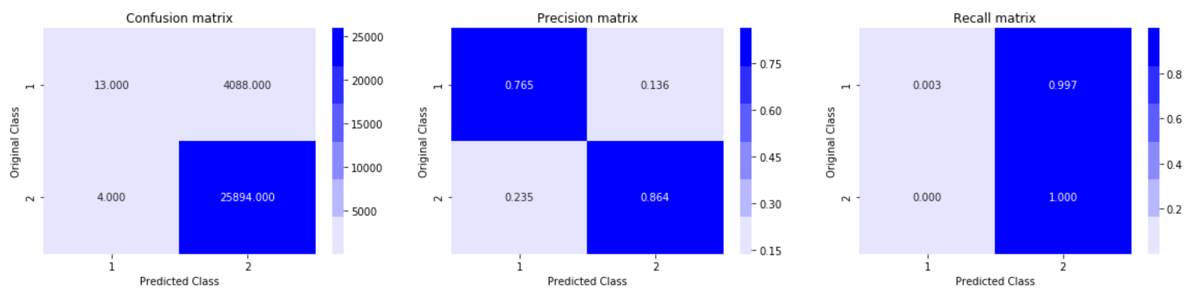
In [133]:

```
clf = LogisticRegression(C= 0.1, penalty= 'l2')
clf.fit(X_train,y_train)
y_train_pred = clf.predict(X_train)
y_pred = clf.predict(X_test)
print("Accuracy on train set: %0.3f%%"%(accuracy_score(y_train, y_train_pred)*100))
print("Precision on train set: %0.3f"%(precision_score(y_train, y_train_pred)))
print("Recall on train set: %0.3f"%(recall_score(y_train, y_train_pred)))
print("F1-Score on train set: %0.3f"%(f1_score(y_train, y_train_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
plot_confusion_matrix(y_train, y_train_pred)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print('Confusion matrix for the model is:')
plot_confusion_matrix(y_test, y_pred)
```

```
Accuracy on train set: 88.341%
Precision on train set: 0.884
Recall on train set: 1.000
F1-Score on train set: 0.883
Non Zero weights: 50
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```




```
Accuracy on test set: 86.360%
Precision on test set: 0.864
Recall on test set: 1.000
F1-Score on test set: 0.864
Non Zero weights: 50
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

Confusion matrix for the model is:
```



Using Randomized Search CV to find best parameter

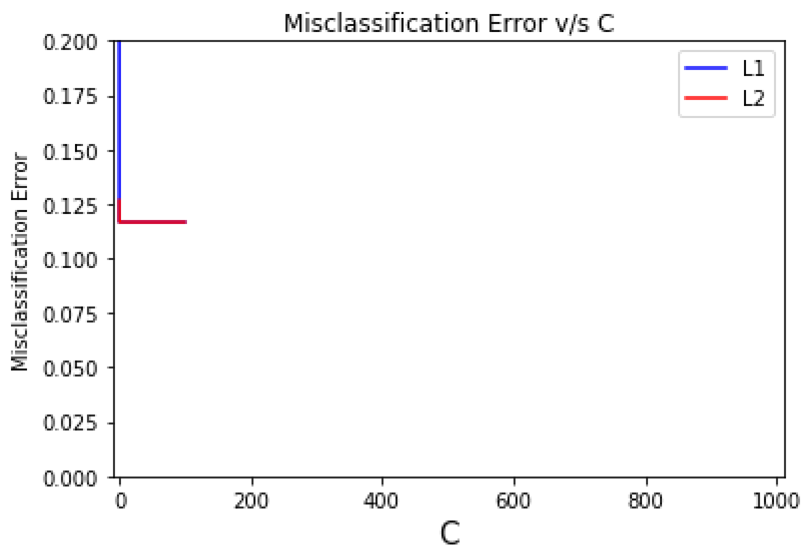
In [134]:

```
%time
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()
#params we need to try on classifier
param_grid = {'C': [100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001]
              ,'penalty':['l1','l2']}
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = RandomizedSearchCV(clf,param_grid,cv=tscv,verbose=1,scoring='f1_micro')
gsv.fit(X_train,y_train)
savetofile(gsv,"Log Reg/gsv_w2v_r")
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
Wall time: 0 ns
Fitting 10 folds for each of 10 candidates, totalling 100 fits

[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:   24.8s finished

Best HyperParameter:  {'penalty': 'l2', 'C': 100}
Best Accuracy: 88.34%
```

In [135]:

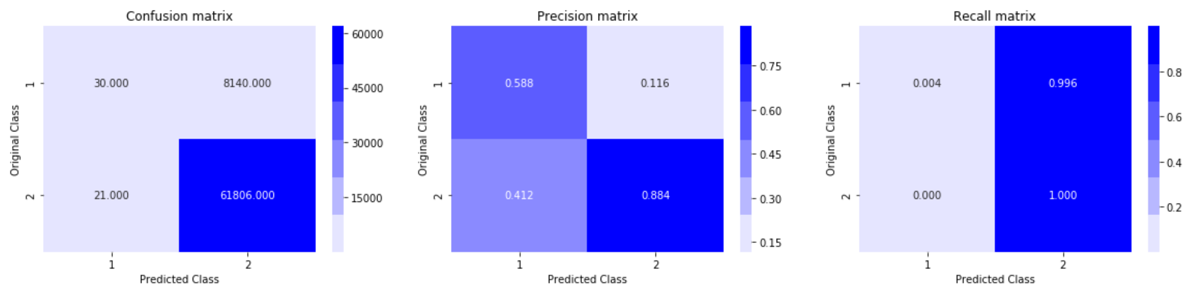
```
gsv = openfromfile("Log Reg/gsv_w2v_r")
plot_error_vs_c_r(gsv)
```



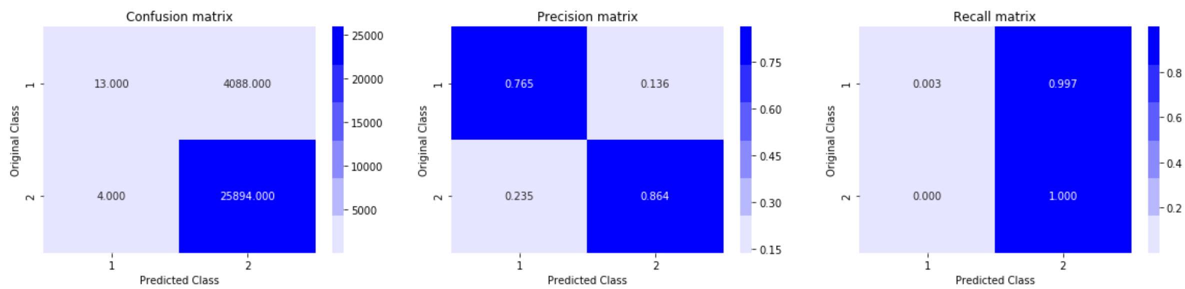
In [136]:

```
clf = LogisticRegression(C= 100, penalty= 'l2')
clf.fit(X_train,y_train)
y_train_pred = clf.predict(X_train)
y_pred = clf.predict(X_test)
print("Accuracy on train set: %0.3f%%"%(accuracy_score(y_train, y_train_pred)*100))
print("Precision on train set: %0.3f"%(precision_score(y_train, y_train_pred)))
print("Recall on train set: %0.3f"%(recall_score(y_train, y_train_pred)))
print("F1-Score on train set: %0.3f"%(f1_score(y_train, y_train_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
plot_confusion_matrix(y_train, y_train_pred)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print('Confusion matrix for the model is:')
plot_confusion_matrix(y_test, y_pred)
```

```
Accuracy on train set: 88.341%
Precision on train set: 0.884
Recall on train set: 1.000
F1-Score on train set: 0.883
Non Zero weights: 50
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```




```
Accuracy on test set: 86.360%
Precision on test set: 0.864
Recall on test set: 1.000
F1-Score on test set: 0.864
Non Zero weights: 50
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

Confusion matrix for the model is:
```



In [192]:

```
import re
def cleanhtml(sentence): #function to clean the word of any html-tags
    cleanr = re.compile('<.*?>')
    sentence = sentence.decode('utf-8')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext
def cleanpunc(sentence): #function to clean the word of any punctuation or special characte
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

In [34]:

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

#Breaking into Train and test
X_train, X_test, y_train, y_test = train_test_split(final['CleanedText'].values,final['Scor
```

In [35]:

```
print ("Training Set - ", X_train.shape)
print ("Test Set - ", X_test.shape)
```

```
Training Set -  (70000,)
Test Set -  (30000,)
```

In [36]:

```
tf_idf_vect = TfidfVectorizer()
final_tf_idf = tf_idf_vect.fit_transform(X_train)
```

In [37]:

```
i=0
list_of_sent=[]
for sent in X_train:
    list_of_sent.append(sent.split())

w2v_model=Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-37-3b080838274d> in <module>()
      4     list_of_sent.append(sent.split())
      5 
----> 6 w2v_model=Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
      7 w2v_words = list(w2v_model.wv.vocab)

~\Anaconda3\lib\site-packages\gensim\models\word2vec.py in __init__(self, se
ntences, size, alpha, window, min_count, max_vocab_size, sample, seed, worke
rs, min_alpha, sg, hs, negative, cbow_mean, hashfxn, iter, null_word, trim_r
ule, sorted_vocab, batch_words, compute_loss, callbacks)
    525             batch_words=batch_words, trim_rule=trim_rule, sg=sg, alp
ha=alpha, window=window, seed=seed,
    526             hs=hs, negative=negative, cbow_mean=cbow_mean, min_alpha
=min_alpha, compute_loss=compute_loss,
--> 527             fast_version=FAST_VERSION)
    528 
    529     def _do_train_job(self, sentences, alpha, inits):

~\Anaconda3\lib\site-packages\gensim\models\base_any2vec.py in __init__(sel
f, sentences, workers, vector_size, epochs, callbacks, batch_words, trim_rul
e, sg, alpha, window, seed, hs, negative, cbow_mean, min_alpha, compute_los
s, fast_version, **kwargs)
    333             if isinstance(sentences, GeneratorType):
    334                 raise TypeError("You can't pass a generator as the s
entences argument. Try an iterator.")
--> 335             self.build_vocab(sentences, trim_rule=trim_rule)
    336             self.train(
    337                 sentences, total_examples=self.corpus_count, epochs=
self.epochs, start_alpha=self.alpha,

~\Anaconda3\lib\site-packages\gensim\models\base_any2vec.py in build_vocab(s
elf, sentences, update, progress_per, keep_raw_vocab, trim_rule, **kwargs)
    484             trim_rule=trim_rule, **kwargs)
    485         report_values['memory'] = self.estimate_memory(vocab_size=re
port_values['num_retained_words'])
--> 486         self.trainables.prepare_weights(self.hs, self.negative, self
.wv, update=update, vocabulary=self.vocabulary)
    487 
    488     def build_vocab_from_freq(self, word_freq, keep_raw_vocab=False,
corpus_count=None, trim_rule=None, update=False):

~\Anaconda3\lib\site-packages\gensim\models\word2vec.py in prepare_weights(s
elf, hs, negative, wv, update, vocabulary)
   1400         # set initial input/projection and hidden weights
   1401         if not update:
-> 1402             self.reset_weights(hs, negative, wv)
   1403         else:
   1404             self.update_weights(hs, negative, wv)
```

```
~\Anaconda3\lib\site-packages\gensim\models\word2vec.py in reset_weights(sel
f, hs, negative, wv)
   1417         for i in xrange(len(wv.vocab)):
   1418             # construct deterministic seed from word AND seed argume
nt
-> 1419             wv.vectors[i] = self.seeded_vector(wv.index2word[i] + st
r(self.seed), wv.vector_size)
   1420         if hs:
   1421             self.syn1 = zeros((len(wv.vocab), self.layer1_size), dty
pe=REAL)

TypeError: can't concat str to bytes
```

In [ ]:

```
tfidf_feat = tf_idf_vect.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

X_train_tfidfw2v = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in list_of_sent: # 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 sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = final_tf_idf[row, tfidf_feat.index(word)]
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    X_train_tfidfw2v.append(sent_vec)
    row += 1
```

In [ ]:

```
tf_idf_vect = TfidfVectorizer()
final_tf_idf = tf_idf_vect.fit_transform(X_test)
```

In [ ]:

```
i=0
list_of_sent=[]
for sent in X_test:
    list_of_sent.append(sent.split())
w2v_model=Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
```

In [ ]:

```
tfidf_feat = tf_idf_vect.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

X_test = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in list_of_sent: # 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 sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = final_tf_idf[row, tfidf_feat.index(word)]
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    X_test.append(sent_vec)
    row += 1
```

In [ ]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler(with_mean=False)
X_train_tfidfw2v_vec = sc.fit_transform(X_train_tfidfw2v)
X_test_tfidfw2v_vec = sc.transform(X_test)
```

In [ ]:

```
#To show how Time Series Split splits the data
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=10)
for train, cv in tscv.split(X_train_tfidfw2v_vec):
#     print("%s %s" % (train, cv))
    print(X_train_tfidfw2v_vec[train].shape, X_train_tfidfw2v_vec[cv].shape)
```

grid

In [207]:

```
%time
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()
#params we need to try on classifier
param_grid = {'C':[100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001],
             'penalty':['l1','l2']}
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = GridSearchCV(clf,param_grid,cv=tscv,verbose=1,scoring='f1_micro')
gsv.fit(X_train_tfidfw2v_vec,y_train)
savetofile(gsv,"Log Reg/gsv_w2v")
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
Wall time: 0 ns

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-207-9fe5b98a73aa> in <module>()
      9 tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
     10 gsv = GridSearchCV(clf,param_grid,cv=tscv,verbose=1,scoring='f1_micr
o')
---> 11 gsv.fit(X_train_tfidfw2v_vec,y_train)
     12 savetofile(gsv,"Log Reg/gsv_w2v")
     13 print("Best HyperParameter: ",gsv.best_params_)

~\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py in fit(sel
f, X, y, groups, **fit_params)
    614             refit_metric = 'score'
    615 
--> 616         X, y, groups = indexable(X, y, groups)
    617         n_splits = cv.get_n_splits(X, y, groups)
    618         # Regenerate parameter iterable for each fit

~\Anaconda3\lib\site-packages\sklearn\utils\validation.py in indexable(*iter
ables)
    227         else:
    228             result.append(np.array(X))
--> 229     check_consistent_length(*result)
    230     return result
    231 

~\Anaconda3\lib\site-packages\sklearn\utils\validation.py in check_consisten
t_length(*arrays)
    202     if len(uniques) > 1:
    203         raise ValueError("Found input variables with inconsistent nu
mbers of"
--> 204                          " samples: %r" % [int(l) for l in lengths])
    205 
    206 

ValueError: Found input variables with inconsistent numbers of samples: [300
00, 70000]
```

In [ ]:

```

```

Showing how sparsity increases as we increase lambda or decrease C when L1 Regularizer is used

In [ ]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 1000, penalty= 'l1')
clf.fit(X_train_tfidfw2v_vec,y_train)
y_pred = clf.predict(X_train_tfidfw2v_vec)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

In [ ]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 100, penalty= 'l1')
clf.fit(X_train_tfidfw2v_vec,y_train)
y_pred = clf.predict(X_train_tfidfw2v_vec)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

In [ ]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 10, penalty= 'l1')
clf.fit(X_train_tfidfw2v_vec,y_train)
y_pred = clf.predict(X_train_tfidfw2v_vec)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

In [ ]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 1, penalty= 'l1')
clf.fit(X_train_tfidfw2v_vec,y_train)
y_pred = clf.predict(X_train_tfidfw2v_vec)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

In [ ]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 0.1, penalty= 'l1')
clf.fit(X_train_tfidfw2v_vec,y_train)
y_pred = clf.predict(X_train_tfidfw2v_vec)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

In [ ]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 0.01, penalty= 'l1')
clf.fit(X_train_tfidfw2v_vec,y_train)
y_pred = clf.predict(X_train_tfidfw2v_vec)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

We can see how drastically the sparsity decreases from 19430 non-zero weights(@ C=1000) to only 5953 non-zero weights(@ C=0.01) when we use L1 Regularization

Randomized Search CV

In [ ]:

```
%time
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()
#params we need to try on classifier
param_grid = { 'C':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001],
              'penalty':['l1','l2']}
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = RandomizedSearchCV(clf,param_grid,cv=tscv,verbose=1,scoring='f1_micro')
gsv.fit(X_train_tfidfw2v_vec,y_train)
savetofile(gsv,"Log Reg/gsv_uni_r")
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

In [ ]:

```
clf = LogisticRegression(C= 0.0001, penalty= 'l2')
clf.fit(X_train_tfidfw2v_vec,y_train)
y_train_pred = clf.predict(X_train)
y_pred = clf.predict(X_train_tfidfw2v_vec)
print("Accuracy on train set: %0.3f%%"%(accuracy_score(y_train, y_train_pred)*100))
print("Precision on train set: %0.3f"%(precision_score(y_train, y_train_pred)))
print("Recall on train set: %0.3f"%(recall_score(y_train, y_train_pred)))
print("F1-Score on train set: %0.3f"%(f1_score(y_train, y_train_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
plot_confusion_matrix(y_train, y_train_pred)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='micro')))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print('Confusion matrix for the model is:')
plot_confusion_matrix(y_test, y_pred)
```

In [39]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(ngram_range=(1,2)) #Using bi-grams
tfidf_vec_ns = tfidf.fit_transform(final['CleanedText_NoStem'].values)

#Saving the variable to access later without recomputing
# savetofile(tfidf_vec,"tfidf")

#Loading the variable from file
# tfidf_vec = openfromfile("tfidf")

print(tfidf_vec_ns.shape)

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

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

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-39-20d4449e4d18> in <module>()
      1 from sklearn.feature_extraction.text import TfidfVectorizer
      2 tfidf = TfidfVectorizer(ngram_range=(1,2)) #Using bi-grams
----> 3 tfidf_vec_ns = tfidf.fit_transform(final['CleanedText_NoStem'].value
s)
      4
      5 #Saving the variable to access later without recomputing

~\Anaconda3\lib\site-packages\sklearn\feature_extraction\text.py in fit_tran
sform(self, raw_documents, y)
   1379             Tf-idf-weighted document-term matrix.
   1380         """
-> 1381         X = super(TfidfVectorizer, self).fit_transform(raw_documents
)
   1382         self._tfidf.fit(X)
   1383         # X is already a transformed view of raw_documents so

~\Anaconda3\lib\site-packages\sklearn\feature_extraction\text.py in fit_tran
sform(self, raw_documents, y)
    867
    868         vocabulary, X = self._count_vocab(raw_documents,
--> 869                                           self.fixed_vocabulary_)
    870
    871         if self.binary:

~\Anaconda3\lib\site-packages\sklearn\feature_extraction\text.py in _count_v
ocab(self, raw_documents, fixed_vocab)
    809             vocabulary = dict(vocabulary)
    810             if not vocabulary:
--> 811                 raise ValueError("empty vocabulary; perhaps the docu
ments only"
    812                                  " contain stop words")
    813

ValueError: empty vocabulary; perhaps the documents only contain stop words
```

In [7]:

```
con = sqlite3.connect("final.sqlite")#Connection object that represents the database

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

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

Out[7]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpful
0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
1	138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	
2	138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1	
3	138690	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg " (Kate)"	1	
4	138691	150509	0006641040	A3CMRKGE0P909G	Teresa	3	

In [8]:

```
final1 =  final.drop_duplicates(subset={"UserId","ProfileName","Time","Text"},keep="first")
```

In [9]:

```
size_diff = final1['Id'].size/final['Id'].size
print("%.1f %% reduction in data after deleting duplicates"%((1-size_diff)*100))
print("Size of data",final1['Id'].size," rows ")
```

```
0.0 % reduction in data after deleting duplicates
Size of data 364171  rows
```

In [11]:

```
import re #Regex (Regualar Expr Operations)
#string = r"sdfsdfd" :- r is for raw string as Regex often uses \ backslashes(\w), so they

########Function to remove html tags from data
def striphtml(data):
    p = re.compile('<.*?>')#Find this kind of pattern
#     print(p.findall(data))#List of strings which follow the regex pattern
    return p.sub('',data) #Substitute nothing at the place of strings which matched the pat

striphtml('<a href="foo.com" class="bar">I Want This <b>text!</b></a><>')
```

Out[11]:

```
'I Want This text!'
```

In [12]:

```
def strippunc(data):
    p = re.compile(r'[?|!|\'|"|#|.|,|)|(|\|/|~|%|*]')
    return p.sub('',data)
strippunc("fsd*?~,,,( sdfsdfdsvv)#")
```

Out[12]:

```
'fsd sdfsdfdsvv'
```

In [13]:

```
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

stop = stopwords.words('english') #All the stopwords in English language
#excluding some useful words from stop words list as we doing sentiment analysis
excluding = ['against','not','don', "don't",'ain', 'aren', "aren't", 'couldn', "couldn't",
             'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'i
             'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't",'shouldn', "shoul
             "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
stop = [words for words in stop if words not in excluding]
print(stop)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you'r
e", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves',
'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'i
t', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselve
s', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'tho
se', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has',
'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'bu
t', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for',
'with', 'about', 'between', 'into', 'through', 'during', 'before', 'after',
'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'ove
r', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'w
here', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'o
ther', 'some', 'such', 'no', 'nor', 'only', 'own', 'same', 'so', 'than', 'to
o', 'very', 's', 't', 'can', 'will', 'just', 'should', "should've", 'now',
'd', 'll', 'm', 'o', 're', 've', 'y', 'ma', 'shan', "shan't"]
```

In [14]:

```
from nltk.stem import SnowballStemmer
snow = SnowballStemmer('english') #initialising the snowball stemmer
print("Stem/Root words of the some of the words using SnowBall Stemmer:")
print(snow.stem('tasty'))
print(snow.stem('tasteful'))
print(snow.stem('tastiest'))
print(snow.stem('delicious'))
print(snow.stem('amazing'))
print(snow.stem('amaze'))
print(snow.stem('initialize'))
print(snow.stem('fabulous'))
print(snow.stem('Honda City'))
print(snow.stem('unpleasant'))
```

```
Stem/Root words of the some of the words using SnowBall Stemmer:
tasti
tast
tastiest
delici
amaz
amaz
initi
fabul
honda c
unpleas
```

In [16]:

```
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
for sent in final1['Text'][2:3].values: #Running only for 2nd review
    filtered_sentence=[]
    print(sent) #Each review
    sent=striphtml(sent)# remove HTML tags
    sent=strippunc(sent)# remove Punctuation Symbols
    print(sent.split())
    for w in sent.split():
        print("=================================>",w)
        if((w.isalpha()) and (len(w)>2)):#If it is a numerical value or character of lenght
            if(w.lower() not in stop):# If it is a stopword
                s=(snow.stem(w.lower())).encode('utf8') #Stemming the word using SnowBall S
                print("Selected: Stem Word->",s)
                filtered_sentence.append(s)
            else:
                print("Eliminated as it is a stopword")
                continue
        else:
            print("Eliminated as it is a numerical value or character of lenght less than 2
            continue
#     print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words

    final_string.append(str1)
    print("***********************************************************************")
    print("Finally selected words from the review:\n",final_string)
```

```
This is a fun way for children to learn their months of the year!  We will
learn all of the poems throughout the school year.  they like the handmoti
ons which I invent for each poem.
['This', 'is', 'a', 'fun', 'way', 'for', 'children', 'to', 'learn', 'thei
r', 'months', 'of', 'the', 'year', 'We', 'will', 'learn', 'all', 'of', 'th
e', 'poems', 'throughout', 'the', 'school', 'year', 'they', 'like', 'the',
'handmotions', 'which', 'I', 'invent', 'for', 'each', 'poem']
=================================> This
Eliminated as it is a stopword
=================================> is
Eliminated as it is a numerical value or character of lenght less than 2
=================================> a
Eliminated as it is a numerical value or character of lenght less than 2
=================================> fun
Selected: Stem Word-> b'fun'
=================================> way
Selected: Stem Word-> b'way'
=================================> for
Eliminated as it is a stopword
=================================> children
Selected: Stem Word-> b'children'
=================================> to
Eliminated as it is a numerical value or character of lenght less than 2
=================================> learn
Selected: Stem Word-> b'learn'
=================================> their
Eliminated as it is a stopword
```

```
================================> months
Selected: Stem Word-> b'month'
================================> of
Eliminated as it is a numerical value or character of lenght less than 2
================================> the
Eliminated as it is a stopword
================================> year
Selected: Stem Word-> b'year'
================================> We
Eliminated as it is a numerical value or character of lenght less than 2
================================> will
Eliminated as it is a stopword
================================> learn
Selected: Stem Word-> b'learn'
================================> all
Eliminated as it is a stopword
================================> of
Eliminated as it is a numerical value or character of lenght less than 2
================================> the
Eliminated as it is a stopword
================================> poems
Selected: Stem Word-> b'poem'
================================> throughout
Selected: Stem Word-> b'throughout'
================================> the
Eliminated as it is a stopword
================================> school
Selected: Stem Word-> b'school'
================================> year
Selected: Stem Word-> b'year'
================================> they
Eliminated as it is a stopword
================================> like
Selected: Stem Word-> b'like'
================================> the
Eliminated as it is a stopword
================================> handmotions
Selected: Stem Word-> b'handmot'
================================> which
Eliminated as it is a stopword
================================> I
Eliminated as it is a numerical value or character of lenght less than 2
================================> invent
Selected: Stem Word-> b'invent'
================================> for
Eliminated as it is a stopword
================================> each
Eliminated as it is a stopword
================================> poem
Selected: Stem Word-> b'poem'
***********************************************************************
Finally selected words from the review:
 [b'fun way children learn month year learn poem throughout school year li
ke handmot invent poem']
```

In [20]:

```
import time
# Code takes a while to run as it needs to run on around 500k sentences.
i=0
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
for sent in final1['Text'].values:
    filtered_sentence=[]
#     print(sent) #Each review
    sent=striphtml(sent)# remove HTML tags
    sent=strippunc(sent)# remove Punctuation Symbols
#     print(sent.split())
    for w in sent.split():
#         print("===============================>",w)
        if((w.isalpha()) and (len(w)>2)):#If it is a numerical value or character of lenght
            if(w.lower() not in stop):# If it is a stopword
                s=(snow.stem(w.lower())).encode('utf8') #Stemming the word using SnowBall S
                                                        #encoding as byte-string/utf-8
#                 print("Selected: Stem Word->",s)
                filtered_sentence.append(s)
                if (final1['Score'].values)[i] == 'Positive':
                    all_positive_words.append(s) #list of all words used to describe positi
                if(final1['Score'].values)[i] == 'Negative':
                    all_negative_words.append(s) #list of all words used to describe negati
            else:
#                 print("Eliminated as it is a stopword")
                continue
        else:
#             print("Eliminated as it is a numerical value or character of lenght less than
            continue
#     print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
            #encoding as byte-string/utf-8

    final_string.append(str1)
#     print("***********************************************************************")
#     print("Finally selected words from the review:\n",final_string)
    i+=1
print("Preprocessing completed in ")
```

```
Preprocessing completed in
```

In [22]:

```
%%time
# Code takes a while to run as it needs to run on around 500k sentences.
i=0
str1=' '
final_string_nostem=[]
s=''
for sent in final1['Text'].values:
    filtered_sentence=[]
    sent=striphtml(sent)# remove HTML tags
    sent=strippunc(sent)# remove Punctuation Symbols
    for w in sent.split():
        if((w.isalpha()) and (len(w)>2)):#If it is a numerical value or character of lenght
            if(w.lower() not in stop):# If it is a stopword
                s=w.lower().encode('utf8') #encoding as byte-string/utf-8
            else:
                continue
        else:
            continue
    str1 = b" ".join(filtered_sentence)
    final_string_nostem.append(str1)
    i+=1
print("Preprocessing completed in ")
```

```
Preprocessing completed in
Wall time: 2min 20s
```

In [23]:

```
final_string = []
for sent in final1['CleanedText'].values:
    sent = str(sent)
    sentence=[]
#     print(sent)
    for word in sent.split():
#         print(word)
        sentence.append(word)
#         print(sentence)
    final_string.append(sentence)
```

In [24]:

```
import gensim

w2v_model=gensim.models.Word2Vec(final_string,min_count=5,size=50, workers=-1)
```

In [25]:

```
w2v_model.save('w2vmodel')
```

In [26]:

```
w2v_model = gensim.models.Word2Vec.load('w2vmodel')
```

In [27]:

```
w2v_vocub = w2v_model.wv.vocab
len(w2v_vocub)
```

Out[27]:

```
21938
```

In [28]:

```
w2v_model.wv.most_similar('like')
```

Out[28]:

```
[('cartlidg', 0.5013499855995178),
 ('piquillo', 0.4672944247722626),
 ('comma', 0.45924538373947144),
 ('weav', 0.45906633138656616),
 ('glutam', 0.4582684636116028),
 ('societi', 0.4522799551486969),
 ('jitteri', 0.45100224018096924),
 ('polym', 0.4467247426509857),
 ('lunch', 0.4382770359516144),
 ('clifford', 0.43546944856643677)]
```

In [29]:

```
w2v_model.wv.most_similar('tast')
```

Out[29]:

```
[('gimm', 0.5285957455635071),
 ('margarita', 0.48446333408355713),
 ('hardi', 0.48394066095352173),
 ('digestif', 0.4833998680114746),
 ('argu', 0.4808456599712372),
 ('abat', 0.4770086705684662),
 ('misrepresent', 0.47510796785354614),
 ('moccamast', 0.47300833463668823),
 ('forthcom', 0.4661412835121155),
 ('alimentum', 0.46350759267807007)]
```

In [ ]:

```
tfidf = TfidfVectorizer(ngram_range=(1,2)) #Using bi-grams
tfidf_vec_ns = tfidf.fit_transform(final1['CleanedText'].values)

#Saving the variable to access later without recomputing
# savetofile(tfidf_vec,"tfidf")

#Loading the variable from file
# tfidf_vec = openfromfile("tfidf")

print(tfidf_vec_ns.shape)

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

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

In [ ]:

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

for sent in final1['CleanedText'].values:
    cnt = 0
    weighted_sum  = 0
    sent_vec = np.zeros(300)
    sent = sent.decode("utf-8")
    for word in sent.split():
        try:
#             print(word)
            wvec = w2v_model.wv[word] #Vector of each using w2v model
#             print("w2vec:",wvec)
#             print("tfidf:",tfidf_vec_ns[review,features.index(word)])
            tfidf = tfidf_vec_ns[review,features.index(word)]
#             print(tfidf)
            sent_vec += (wvec * tfidf)
            weighted_sum += tfidf
        except:
            pass
    sent_vec /= weighted_sum
    tfidf_w2v_vec_google.append(sent_vec)
    review += 1
```

In [ ]:

```
```

In [ ]:

```
```

Conclusion

In [182]:

```
from prettytable import PrettyTable

# Names of models
featurization = ['Bag of Words','Bag of Words','bigram','bigram','TFIDF ','TFIDF ','avg w2v
model=['gridsearch ','randomsearch','gridsearch ','randomsearch','gridsearch ','randomsearc

# Training accuracies
F1score= [0.916,0.892,0.922,0.922,0.967,0.914,0.864,0.864]
accuracy = [91.57,89.24,92.17,92.17,91.39,91.39,86.36,86.36]
alpha=[0.05,0.001,0.01,0.01,0.05,0.05,0.1,100]
precision=[0.938,0.897,0.929,0.929,0.935,0.936,0.864,0.864]
recall=[0.967,0.988,0.985,0.985,0.967,0.967,1.00,1.00]
numbering = [1,2,3,4,5,6,7,8]
regularization=['l1','l2','l1','l1','l1','l1','l2','l2']

# Initializing prettytable
ptable = PrettyTable()

# Adding columns
ptable.add_column("S.NO.",numbering)
ptable.add_column("MODEL",featurization)
ptable.add_column("Method",model)
ptable.add_column("C",alpha)
ptable.add_column("regularization",regularization)
ptable.add_column("accuracy",accuracy)
ptable.add_column("f1score",F1score)
ptable.add_column("precision",precision)
ptable.add_column("recall",recall)


# Printing the Table
print(ptable)
```

```
+-------+--------------+--------------+-------+----------------+----------+-
--------+-----------+--------+
| S.NO. |    MODEL     |    Method    |   C   | regularization | accuracy |
f1score | precision | recall |
+-------+--------------+--------------+-------+----------------+----------+-
--------+-----------+--------+
|   1   | Bag of Words | gridsearch   |  0.05 |       l1       |  91.57   |
0.916  |   0.938   | 0.967  |
|   2   | Bag of Words | randomsearch | 0.001 |       l2       |  89.24   |
0.892  |   0.897   | 0.988  |
|   3   |    bigram    | gridsearch   |  0.01 |       l1       |  92.17   |
0.922  |   0.929   | 0.985  |
|   4   |    bigram    | randomsearch |  0.01 |       l1       |  92.17   |
0.922  |   0.929   | 0.985  |
|   5   |    TFIDF     | gridsearch   |  0.05 |       l1       |  91.39   |
0.967  |   0.935   | 0.967  |
|   6   |    TFIDF     | randomsearch |  0.05 |       l1       |  91.39   |
0.914  |   0.936   | 0.967  |
|   7   |   avg w2v    | gridsearch   |  0.1  |       l2       |  86.36   |
0.864  |   0.864   |  1.0   |
|   8   |   avg w2vw   | randomsearch |  100  |       l2       |  86.36   |
0.864  |   0.864   |  1.0   |
+-------+--------------+--------------+-------+----------------+----------+-
--------+-----------+--------+
```

In [ ]:

```

```