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)

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

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

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

Attribute Information:

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 Score/Rating. A rating of 4 or 5 can be cosnidered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

[1]. Reading Data

[1.1] 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 is 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 is above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [4]:

```
%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
```

In [2]:

```
# using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data points
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3  """, con)

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative rat
def partition(x):
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (525814, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenom
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	

In [3]:

```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [5]:

```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ET0	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ET0	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBE1U	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [6]:

```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[6]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I was recommended to try green tea extract to ...	

In [7]:

```
display['COUNT(*)'].sum()
```

Out[7]:

```
393063
```

[2] Exploratory Data Analysis

[2.1] 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 [8]:

```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[8]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDen
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that

ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8)

ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

In [9]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, k
```

In [10]:

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='firs
final.shape
```

Out[10]:

```
(364173, 10)
```

In [11]:

```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[11]:

```
69.25890143662969
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

In [12]:

```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[12]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenc
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	

In [13]:

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [14]:

```
#Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)

#How many positive and negative reviews are present in our dataset?
final['Score'].value_counts()
```

```
(364171, 10)
```

Out[14]:

```
1    307061
0     57110
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags

2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

In [15]:

```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("="*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("="*50)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as
we're driving along and he always can sing the refrain. he's learned about w
hales, India, drooping roses:  i love all the new words this book  introduce
s and the silliness of it all.  this is a classic book i am  willing to bet
my son will STILL be able to recite from memory when he is  in college
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks
is good, but I prefer bolder taste.... imagine my surprise when I ordered 2
boxes - both were expired! One expired back in 2005 for gosh sakes.  I admit
that Amazon agreed to credit me for cost plus part of shipping, but geez, 2
years expired!!!  I'm hoping to find local San Diego area shoppe that carrie
s pods so that I can try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chicken
broth, the only thing I do not think belongs in it is Canola oil. Canola or
rapeseed is not someting a dog would ever find in nature and if it did find
rapeseed in nature and eat it, it would poison them. Today's Food industries
have convinced the masses that Canola oil is a safe and even better oil than
olive or virgin coconut, facts though say otherwise. Until the late 70's it
was poisonous until they figured out a way to fix that. I still like it but
it could be better.
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the
excellence of this product.<br /><br />Thick, delicious.  Perfect.  3 ingred
ients: Water, Maltitol, Natural Maple Flavor.  PERIOD.  No chemicals.  No ga
rbage.<br /><br />Have numerous friends & family members hooked on this stuf
f.  My husband & son, who do NOT like "sugar free" prefer this over major la
bel regular syrup.<br /><br />I use this as my SWEETENER in baking: cheeseca
kes, white brownies, muffins, pumpkin pies, etc... Unbelievably delicious...
<br /><br />Can you tell I like it? :)
==================================================
```

In [16]:

```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as
we're driving along and he always can sing the refrain. he's learned about w
hales, India, drooping roses:  i love all the new words this book  introduce
s and the silliness of it all.  this is a classic book i am  willing to bet
my son will STILL be able to recite from memory when he is  in college
```

In [17]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-
from bs4 import BeautifulSoup

soup = BeautifulSoup(sent_0, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1000, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1500, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_4900, 'lxml')
text = soup.get_text()
print(text)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as
we're driving along and he always can sing the refrain. he's learned about w
hales, India, drooping roses:  i love all the new words this book  introduce
s and the silliness of it all.  this is a classic book i am  willing to bet
my son will STILL be able to recite from memory when he is  in college
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks
is good, but I prefer bolder taste.... imagine my surprise when I ordered 2
boxes - both were expired! One expired back in 2005 for gosh sakes.  I admit
that Amazon agreed to credit me for cost plus part of shipping, but geez, 2
years expired!!!  I'm hoping to find local San Diego area shoppe that carrie
s pods so that I can try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chicken
broth, the only thing I do not think belongs in it is Canola oil. Canola or
rapeseed is not someting a dog would ever find in nature and if it did find
rapeseed in nature and eat it, it would poison them. Today's Food industries
have convinced the masses that Canola oil is a safe and even better oil than
olive or virgin coconut, facts though say otherwise. Until the late 70's it
was poisonous until they figured out a way to fix that. I still like it but
it could be better.
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the
excellence of this product.Thick, delicious.  Perfect.  3 ingredients: Wate
r, Maltitol, Natural Maple Flavor.  PERIOD.  No chemicals.  No garbage.Have
numerous friends & family members hooked on this stuff.  My husband & son, w
ho do NOT like "sugar free" prefer this over major label regular syrup.I use
this as my SWEETENER in baking: cheesecakes, white brownies, muffins, pumpki
n pies, etc... Unbelievably delicious...Can you tell I like it? :)
```

In [18]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

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

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

In [19]:

```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
Great ingredients although, chicken should have been 1st rather than chicken
broth, the only thing I do not think belongs in it is Canola oil. Canola or
rapeseed is not someting a dog would ever find in nature and if it did find
rapeseed in nature and eat it, it would poison them. Today is Food industrie
s have convinced the masses that Canola oil is a safe and even better oil th
an olive or virgin coconut, facts though say otherwise. Until the late 70 is
it was poisonous until they figured out a way to fix that. I still like it b
ut it could be better.
==================================================
```

In [20]:

```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as
we're driving along and he always can sing the refrain. he's learned about w
hales, India, drooping roses:  i love all the new words this book  introduce
s and the silliness of it all.  this is a classic book i am  willing to bet
my son will STILL be able to recite from memory when he is  in college
```

In [21]:

```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
Great ingredients although chicken should have been 1st rather than chicken
broth the only thing I do not think belongs in it is Canola oil Canola or ra
peseed is not someting a dog would ever find in nature and if it did find ra
peseed in nature and eat it it would poison them Today is Food industries ha
ve convinced the masses that Canola oil is a safe and even better oil than o
live or virgin coconut facts though say otherwise Until the late 70 is it wa
s poisonous until they figured out a way to fix that I still like it but it
could be better
```

In [22]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', '
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they'
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'l
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'u
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'd
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any',
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'v
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'do
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn'
            'won', "won't", 'wouldn', "wouldn't"])
```

In [23]:

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

```
100%|██████████| 364171/364171 [09:47<00:00, 620.37it/s]
```

In [24]:

```
preprocessed_reviews[1500]
```

Out[24]:

```
'great ingredients although chicken rather chicken broth thing not think bel
ongs canola oil canola rapeseed not someting dog would ever find nature find
rapeseed nature eat would poison today food industries convinced masses cano
la oil safe even better oil olive virgin coconut facts though say otherwise
late poisonous figured way fix still like could better'
```

In [25]:

```
final['Cleaned_text']=preprocessed_reviews
```

[3.2] Preprocessing Review Summary

In [25]:

```
## Similartly you can do preprocessing for review summary also.
```

[5] Assignment 5: Apply Logistic Regression

1)Apply Logistic Regression on these feature sets SET 1:Review text, preprocessed one converted into vectors using (BOW) SET 2:Review text, preprocessed one converted into vectors using (TFIDF) SET 3:Review text, preprocessed one converted into vectors using (AVG W2v) SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)

2)Hyper paramter tuning (find best hyper parameters corresponding the algorithm that you choose) Find the best hyper parameter which will give the maximum AUC value Find the best hyper paramter using k-fold cross validation or simple cross validation data Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3)Pertubation Test Get the weights W after fit your model with the data X i.e Train data. Add a noise to the X (X' = X + e) and get the new data set X' (if X is a sparse matrix, X.data+=e) Fit the model again on data X' and get the weights W' Add a small eps value(to eliminate the divisible by zero error) to W and W' i.e W=W+10^-6 and W' = W'+10^-6 Now find the % change between W and W' (| (W-W') / (W) |)*100) Calculate the 0th, 10th, 20th, 30th, ...100th percentiles, and observe any sudden rise in the values of percentage_change_vector Ex: consider your 99th percentile is 1.3 and your 100th percentiles are 34.6, there is sudden rise from 1.3 to 34.6, now calculate the 99.1, 99.2, 99.3,..., 100th percentile values and get the proper value after which there is sudden rise the values, assume it is 2.5 Print the feature names whose % change is more than a threshold x(in our example it's 2.5)

4)Sparsity Calculate sparsity on weight vector obtained after using L1 regularization

5)Feature importance Get top 10 important features for both positive and negative classes separately.

6)Feature engineering To increase the performance of your model, you can also experiment with with feature engineering like : Taking length of reviews as another feature. Considering some features from review summary as well.

7)Representation of results You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.
Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

8)Conclusion You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link. (https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf)

In [26]:

```
final.head()
```

Out[26]:

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

Sorting dataset based on 'Time' feature

In [27]:

```
final_reviews = final.sort_values('Time', axis=0, ascending=True, inplace=False, kind='quic
final_reviews.shape
```

Out[27]:

```
(364171, 11)
```

Spliting data

In [28]:

```
#split data into train, cross validate and test
%matplotlib inline
import warnings
from sklearn.model_selection import train_test_split
X = final_reviews['Cleaned_text']
Y = final_reviews['Score']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=.33, random_state=0)
X_tr, X_cv, Y_tr, Y_cv = train_test_split(X_train, Y_train, test_size=.33, random_state=0)
```

In [29]:

```
print('X_train, Y_train', X_train.shape, Y_train.shape)
print('X_test, Y_test', X_test.shape, Y_test.shape)
print('X_tr, Y_tr', X_tr.shape, Y_tr.shape)
print('X_cv, Y_cv', X_cv.shape, Y_cv.shape)
```

```
X_train, Y_train (243994,) (243994,)
X_test, Y_test (120177,) (120177,)
X_tr, Y_tr (163475,) (163475,)
X_cv, Y_cv (80519,) (80519,)
```

Applying Logistic Regression

[5.1] Logistic Regression on BOW, SET 1

[5.1.1] Applying Logistic Regression with L1 regularization on BOW, SET 1

In [30]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import model_selection
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as pyplt
```

Grid Search for optimal HyperParameter

In [114]:

```
#tuned_parameter=[{'C':[10**-4,10**-3,10**-2,10**-1,1,10**1,10**2,10**3,10**4]}]
#model = GridSearchCV(LogisticRegression(), tuned_parameters, scoring = 'f1', cv=5)
#model.fit(X_train, y_train)
#model=GridSearchCV(LogisticRegression(),tuned_parameter,scoring='roc_auc',cv=5)
#tqdm(model.fit(train_bows,y_train))
```

Grid Search Cross Validation Taking too much time

Logistic Regression with simple CV

Training Model

In [115]:

```
from sklearn.linear_model import LogisticRegression
# logistic Regression with penality = 'l1'
def logistic_l1(X_train,X_cv,Y_train,Y_cv):
    best_C=0
    max_roc_auc=-1
    pred_cv = []
    pred_train = []
    C=[10000,5000,1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001,0.0000
    for i in C[-19:]:
        logisticl1 = LogisticRegression(C=i,penalty='l1')
        logisticl1.fit(X_train,Y_train)
        probs = logisticl1.predict_proba(X_cv)
        prob = logisticl1.predict_proba(X_train)
        probs = probs[:,1]
        prob = prob[:,1]
        auc_score_cv = roc_auc_score(Y_cv,probs)
        auc_score_train = roc_auc_score(Y_train,prob)
        print(i," for CV data auc score is --> ",auc_score_cv," and for train data auc scor
        pred_cv.append(auc_score_cv)
        pred_train.append(auc_score_train)
        if(max_roc_auc<auc_score_cv):
            max_roc_auc=auc_score_cv
            best_C=i
    print(f"\n Best C Value {best_C} with highest roc_auc Score is {max_roc_auc}")
    sns.set_style("darkgrid")
    plt.xscale('log')
    plt.plot(C, pred_cv,'r-', label = 'CV Data',marker='*')
    plt.plot(C,pred_train,'g-', label ='Train Data',marker='*')
    plt.legend(loc='upper right')
    plt.title(r'Auc Score v/s $\lambda$')
    plt.xlabel(r"alpha values",fontsize=12)
    plt.ylabel("roc_auc",fontsize=12)
    plt.show()
    # calculate roc curve
    fpr, tpr, thresholds = roc_curve(Y_train,prob)
    # plot no skill
    pyplt.plot([0, 1], [0, 1], linestyle='--')
    # plot the roc curve for the model
    pyplt.plot(fpr, tpr, marker='*')
    pyplt.title("Line Plot of ROC Curve on Train Data")
    pyplt.ylabel('True Positive Rate')
    pyplt.xlabel('False Positive Rate')
    pyplt.show()
```

In [116]:

```
# logistic Regression with penality = 'l2'
def logistic_l2(X_train,X_cv,Y_train,Y_cv):
    best_C=0
    max_roc_auc=-1
    pred_cv = []
    pred_train = []
    C=[10000,5000,1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001,0.0000
    for i in C[-19:]:
        logisticl2 = LogisticRegression(penalty='l2',C=i)
        logisticl2.fit(X_train,Y_train)
        probs = logisticl2.predict_proba(X_cv)
        prob = logisticl2.predict_proba(X_train)
        probs = probs[:,1]
        prob = prob[:,1]
        auc_score_cv = roc_auc_score(Y_cv,probs)
        auc_score_train = roc_auc_score(Y_train,prob)
        print(i," for CV data auc score is --> ",auc_score_cv," and for train data auc scor
        pred_cv.append(auc_score_cv)
        pred_train.append(auc_score_train)
        if(max_roc_auc<auc_score_cv):
            max_roc_auc=auc_score_cv
            best_C=i
    print(f"\n Best C Value {best_C} with highest roc_auc Score is {max_roc_auc}")
    sns.set_style("darkgrid")
    plt.xscale('log')
    plt.plot(C, pred_cv,'r-', label = 'CV Data',marker='*')
    plt.plot(C,pred_train,'g-', label ='Train Data',marker='*')
    plt.legend(loc='upper right')
    plt.title(r'Auc Score v/s $\lambda$')
    plt.xlabel(r"alpha values",fontsize=12)
    plt.ylabel("roc_auc",fontsize=12)
    plt.show()
    # calculate roc curve
    fpr, tpr, thresholds = roc_curve(Y_train,prob)
    # plot no skill
    pyplt.plot([0, 1], [0, 1], linestyle='--')
    # plot the roc curve for the model
    pyplt.plot(fpr, tpr, marker='*')
    pyplt.title("Line Plot of ROC Curve on Train Data")
    pyplt.ylabel('True Positive Rate')
    pyplt.xlabel('False Positive Rate')
    pyplt.show()
```

Testing Model

In [117]:

```
import scikitplot.metrics as skplt
def testing_l1(X_train,Y_train,X_test,Y_test,optimal_C):
    log1 = LogisticRegression(penalty='l1',C=optimal_C)
    log1.fit(X_train,Y_train)
    probs_train = log1.predict_proba(X_train)
    probs_test = log1.predict_proba(X_test)
    # keep probabilities for the positive outcome only
    probs_train = probs_train[:, 1]
    probs_test = probs_test[:, 1]
    print("AUC Score",roc_auc_score(Y_train,probs_train))
    print("AUC Score",roc_auc_score(Y_test,probs_test))
    # calculate roc curve
    fpr_train, tpr_train, thresholds = roc_curve(Y_train,probs_train)
    fpr_test, tpr_test, thresholds = roc_curve(Y_test,probs_test)

    # plot the roc curve for the model
    plt.plot(fpr_train, tpr_train, color='darkorange',marker='*',label="ROC curve of train
    plt.plot(fpr_test, tpr_test,color='green',marker='*', label="ROC curve of Test data="+s
    plt.plot([0, 1], [0, 1],color='navy', linestyle='--')
    plt.title("Line Plot of ROC Curve on Test Data")
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show
    prediction=log1.predict(X_test)
    skplt.plot_confusion_matrix(Y_test,prediction)
    print("macro f1 score for data :",metrics.f1_score(Y_test, prediction, average = 'macro
    print("micro f1 scoore for data:",metrics.f1_score(Y_test, prediction, average = 'micro
    print("hamming loss for data:",metrics.hamming_loss(Y_test,prediction))
    print("Precision recall report for data:\n",metrics.classification_report(Y_test, predi
```

In [118]:

```
import scikitplot.metrics as skplt
def testing_l2(X_train,Y_train,X_test,Y_test,optimal_C):
    log2 = LogisticRegression(penalty='l2',C=optimal_C)
    log2.fit(X_train,Y_train)
    prob_train = log2.predict_proba(X_train)
    prob_test = log2.predict_proba(X_test)
    # keep probabilities for the positive outcome only
    probs_train = prob_train[:, 1]
    probs_test = prob_test[:, 1]
    print("AUC Score",roc_auc_score(Y_train,probs_train))
    print("AUC Score",roc_auc_score(Y_test,probs_test))
    # calculate roc curve
    fpr_train, tpr_train, thresholds = roc_curve(Y_train,probs_train)
    fpr_test, tpr_test, thresholds = roc_curve(Y_test,probs_test)
    # plot the roc curve for the model
    plt.plot(fpr_train, tpr_train, color='darkorange',marker='*',label="ROC curve of train
    plt.plot(fpr_test, tpr_test,color='green',marker='*', label="ROC curve of Test data="+s
    plt.plot([0, 1], [0, 1],color='navy', linestyle='--')
    plt.title("Line Plot of ROC Curve on Test Data")
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show
    prediction=np.argmax(prob_test,axis=1)
    skplt.plot_confusion_matrix(Y_test,prediction)
    print("macro f1 score for data :",metrics.f1_score(Y_test, prediction, average = 'macro
    print("micro f1 scoore for data:",metrics.f1_score(Y_test, prediction, average = 'micro
    print("hamming loss for data:",metrics.hamming_loss(Y_test,prediction))
    print("Precision recall report for data:\n",metrics.classification_report(Y_test, predi
```

Top 10 features

In [119]:

```
#Code Reference:https://stackoverflow.com/questions/11116697/how-to-get-most-informative-fe

def imp_feature(vectorizer,classifier, n =10):
    feature_names = vectorizer.get_feature_names()
    coefs_with_fns = sorted(zip(classifier.coef_[0], feature_names))
    top = zip(coefs_with_fns[:n], coefs_with_fns[:-(n + 1):-1])
    print("\t\t\tNegative\t\t\t\t\t\tPositive")
    print("_____")
    for (coef_1, fn_1), (coef_2, fn_2) in top:
        print("\t%.4f\t%-15s\t\t\t\t%.4f\t%-15s" % (coef_1, fn_1, coef_2, fn_2))
```

Techniques for vectorization :--

Bag of Words (BoW)

In [120]:

```
#BoW
count_vect = CountVectorizer() #in scikit-learn
bow_train = count_vect.fit_transform(X_tr)
print("The type of count vectorizer ",type(bow_train))
print("The shape of out text BOW vectorizer ",bow_train.get_shape())
#print("the number of unique words ", final_counts.get_shape()[1])
bow_cv = count_vect.transform(X_cv)
bow_test = count_vect.transform(X_test)
print("CV Data Size: ",bow_cv.shape)
print("Test Data Size: ",bow_test.shape)
```

```
The type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
The shape of out text BOW vectorizer  (163475, 77052)
CV Data Size:  (80519, 77052)
Test Data Size:  (120177, 77052)
```

In [121]:

```
#Normalize Data
from sklearn import preprocessing
from sklearn.preprocessing import Normalizer
bow_train=preprocessing.normalize(bow_train)
bow_cv=preprocessing.normalize(bow_cv)
bow_test=preprocessing.normalize(bow_test)
print("The shape of out text BOW vectorizer ",bow_train.get_shape())
print("CV Data Size: ",bow_cv.shape)
print("Test Data Size: ",bow_test.shape)
```

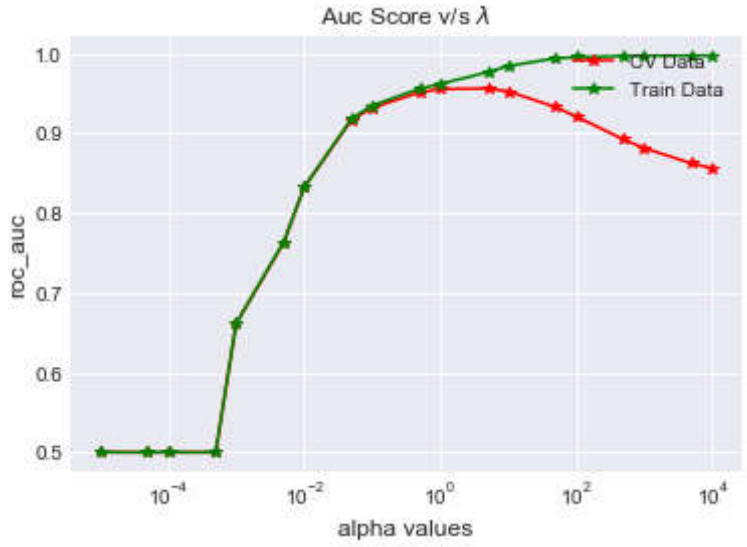
```
The shape of out text BOW vectorizer  (163475, 77052)
CV Data Size:  (80519, 77052)
Test Data Size:  (120177, 77052)
```

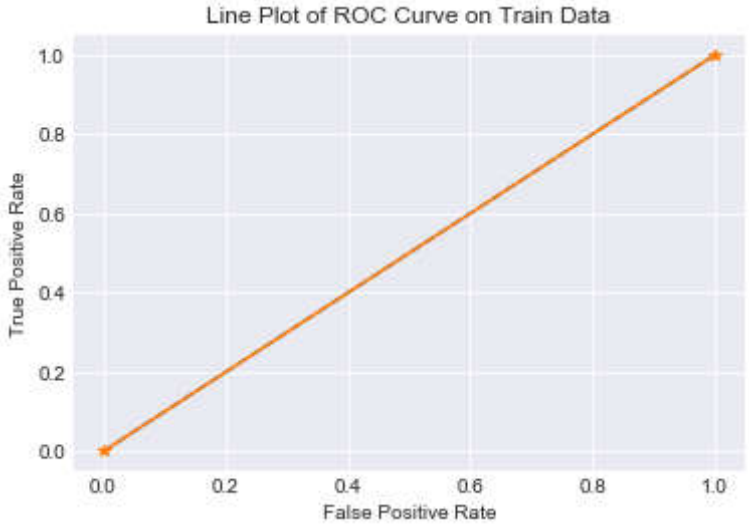
In [122]:

```
# find optimal C using l1 regularization
logistic_l1(bow_train,bow_cv,Y_tr,Y_cv)
```

```
10000  for CV data auc score is -->  0.8561280344452495  and for train data
auc score is -->  0.9972953015408291
5000  for CV data auc score is -->  0.8621461554422962  and for train data a
uc score is -->  0.9972170424461708
1000  for CV data auc score is -->  0.8813725008808286  and for train data a
uc score is -->  0.9969284944008447
500  for CV data auc score is -->  0.8921921620373259  and for train data au
c score is -->  0.9966903053024607
100  for CV data auc score is -->  0.9209385187420597  and for train data au
c score is -->  0.995301111025509
50  for CV data auc score is -->  0.9328333686371433  and for train data auc
score is -->  0.9938857106162442
10  for CV data auc score is -->  0.9524409353063401  and for train data auc
score is -->  0.9842409442022246
5  for CV data auc score is -->  0.9562051224558022  and for train data auc
score is -->  0.9763666198158011
1  for CV data auc score is -->  0.9551747290952217  and for train data auc
score is -->  0.9617415877043983
0.5  for CV data auc score is -->  0.9510979651014626  and for train data au
c score is -->  0.9557647751879038
0.1  for CV data auc score is -->  0.9319271778447294  and for train data au
c score is -->  0.9346585174417756
0.05  for CV data auc score is -->  0.9169420788697793  and for train data a
uc score is -->  0.9183992684163866
0.01  for CV data auc score is -->  0.832050703736344  and for train data au
c score is -->  0.8319509166752528
0.005  for CV data auc score is -->  0.7627494756126076  and for train data
auc score is -->  0.7635289346353271
0.001  for CV data auc score is -->  0.6611996510961426  and for train data
auc score is -->  0.6615700988698514
0.0005  for CV data auc score is -->  0.5  and for train data auc score is -
->  0.5
0.0001  for CV data auc score is -->  0.5  and for train data auc score is -
->  0.5
5e-05  for CV data auc score is -->  0.5  and for train data auc score is --
>  0.5
1e-05  for CV data auc score is -->  0.5  and for train data auc score is --
>  0.5

 Best C Value 5 with highest roc_auc Score is 0.9562051224558022
```


Auc Score v/s λ
CV Data
Train Data
roc_auc
alpha values

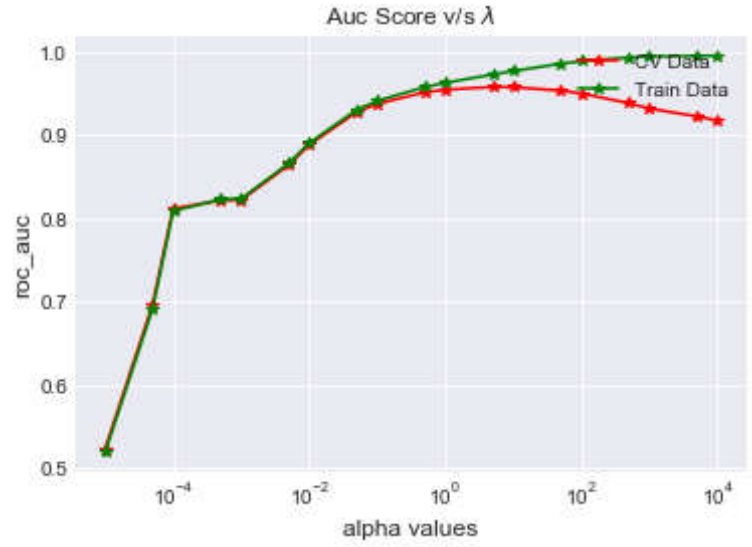

Line Plot of ROC Curve on Train Data
True Positive Rate
False Positive Rate

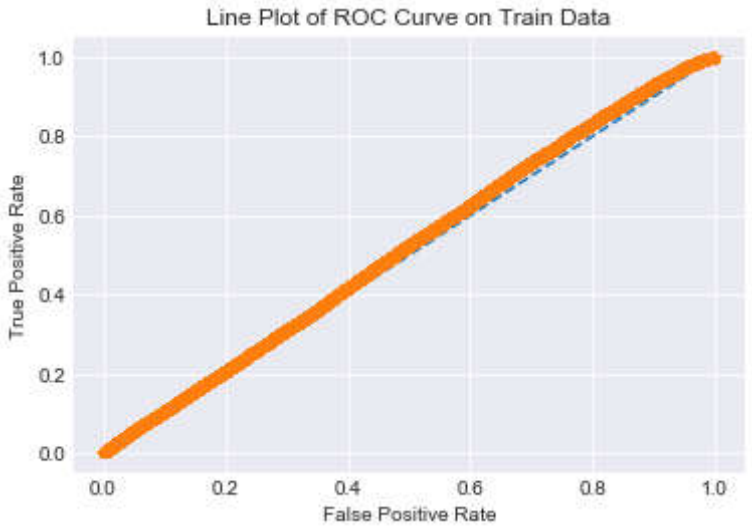
In [123]:

```
# find optimal C using l2 regularization
logistic_l2(bow_train,bow_cv,Y_tr,Y_cv)
```

```
10000  for CV data auc score is -->  0.9177646571342174  and for train data
auc score is -->  0.9957718898827131
5000  for CV data auc score is -->  0.9228582967868977  and for train data a
uc score is -->  0.9953968759841704
1000  for CV data auc score is -->  0.9321078566985592  and for train data a
uc score is -->  0.994551774085449
500  for CV data auc score is -->  0.9386165341918222  and for train data au
c score is -->  0.9934954369884456
100  for CV data auc score is -->  0.9503659126988239  and for train data au
c score is -->  0.9889699856882173
50  for CV data auc score is -->  0.9537341602284057  and for train data auc
score is -->  0.9859465110743391
10  for CV data auc score is -->  0.9578301920915766  and for train data auc
score is -->  0.9771512977735065
5  for CV data auc score is -->  0.9580191948526  and for train data auc sco
re is -->  0.973026048056298
1  for CV data auc score is -->  0.9547710043532381  and for train data auc
score is -->  0.962950934909348
0.5  for CV data auc score is -->  0.9515707839013745  and for train data au
c score is -->  0.9578921884201547
0.1  for CV data auc score is -->  0.937764476351007  and for train data auc
score is -->  0.9413429578211119
0.05  for CV data auc score is -->  0.9277513588863503  and for train data a
uc score is -->  0.9306461220510492
0.01  for CV data auc score is -->  0.8883793224183362  and for train data a
uc score is -->  0.890503904035882
0.005  for CV data auc score is -->  0.8640961283168413  and for train data
auc score is -->  0.8660785783045808
0.001  for CV data auc score is -->  0.8220856419150758  and for train data
auc score is -->  0.8237597401992517
0.0005  for CV data auc score is -->  0.8216884933998135  and for train data
auc score is -->  0.8230277780302426
0.0001  for CV data auc score is -->  0.8119086986066035  and for train data
auc score is -->  0.8089611582908889
5e-05  for CV data auc score is -->  0.6968163320344973  and for train data
auc score is -->  0.6920634013649127
1e-05  for CV data auc score is -->  0.5232250231156979  and for train data
auc score is -->  0.5190235786388253

 Best C Value 5 with highest roc_auc Score is 0.9580191948526
```





LR simple CV Over the unseen or Test data in BoW

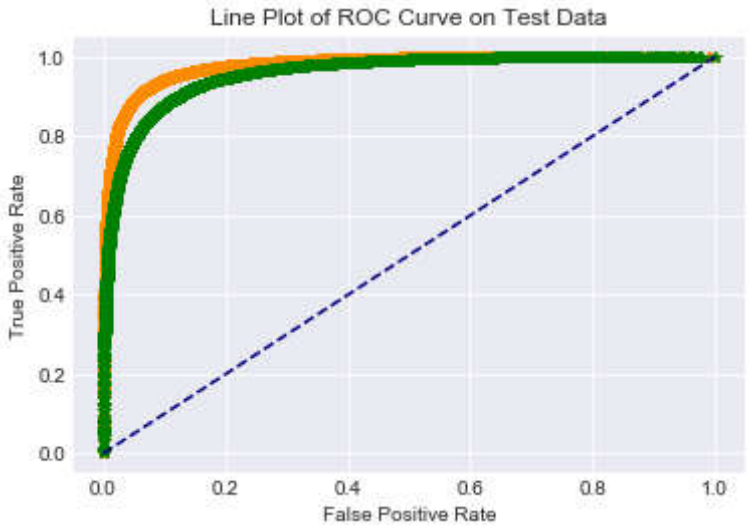
In [124]:

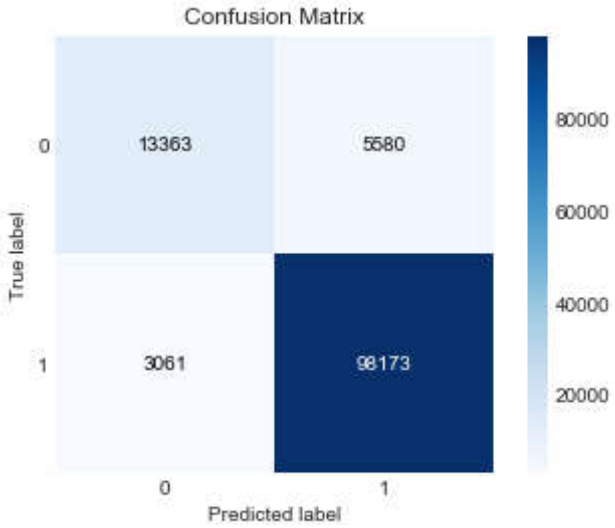
```
# optimal_C=5 for l1 regularization
testing_l1(bow_train,Y_tr,bow_test,Y_test,optimal_C=5)
```

```
AUC Score 0.9763666025443755
AUC Score 0.9565382269006446
macro f1 score for data : 0.8567611513187232
micro f1 scoore for data: 0.9280977225259409
hamming loss for data: 0.0719022774740591
Precision recall report for data:
              precision    recall  f1-score   support

           0       0.81      0.71      0.76     18943
           1       0.95      0.97      0.96    101234

avg / total       0.93      0.93      0.93    120177
```





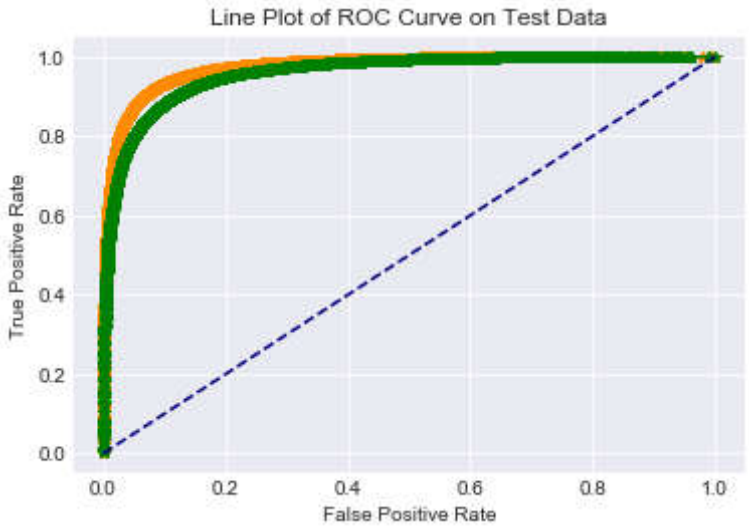
In [125]:

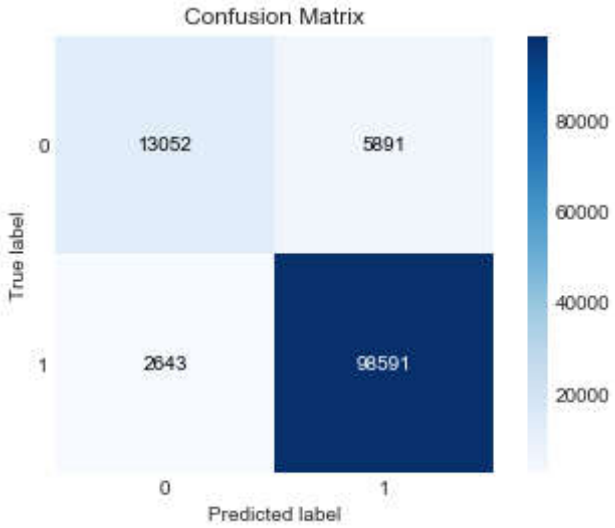
```
# optimal_C=5 for l2 regularization
testing_l2(bow_train,Y_tr,bow_test,Y_test,optimal_C=5)
```

```
AUC Score 0.973026048056298
AUC Score 0.9581572133963913
macro f1 score for data : 0.8560694059433562
micro f1 scoore for data: 0.9289880759213492
hamming loss for data: 0.07101192407865066
Precision recall report for data:
              precision    recall  f1-score   support

           0       0.83      0.69      0.75     18943
           1       0.94      0.97      0.96    101234

avg / total       0.93      0.93      0.93    120177
```





In [70]:

```
#top 10 important feature for l1 regularization
clf1 = LogisticRegression(penalty='l1',C=5)
clf1.fit(bow_train,Y_tr)
imp_feature(count_vect,clf1)
```

```
                        Negative
Positive
-------------------------------------------------------------------------------
-------------------
        -20.7609        verytasty                               22.8478 plea
santly
        -20.4963        deceptive                               22.5536 scov
ille
        -19.5992        mattress                                15.9572 down
side
        -19.3819        thrash                                  15.8895 skep
tical
        -18.8507        embarrassed                             15.8018 coon
        -18.8078        unappealing                             15.2655 lyle
s
        -18.4530        worst                                   15.2246 chil
l
        -17.9769        nome                                    15.1310 rese
rvation
        -17.8102        lawsuit                                 14.8338 than
kful
        -17.5692        undrinkable                             14.6914 hydr
ated
```

In [71]:

```
#top 10 important feature for l2 regularization
clf2 = LogisticRegression(penalty='l2',C=5)
clf2.fit(bow_train,Y_tr)
imp_feature(count_vect,clf2)
```

```
                        Negative
Positive
-------------------------------------------------------------------------------
-------------------
        -14.8077        worst                                   10.4203 deli
cious
        -13.6182        disappointing                           10.2292 plea
santly
        -13.3558        disappointment                          9.7546  perf
ect
        -11.1657        terrible                                8.8724  high
ly
        -10.9025        awful                                   8.7397  exce
llent
        -9.7827 threw                                   8.6546  amazing
        -9.6480 horrible                                8.2612  hooked
        -9.4791 hopes                                   8.1443  beat
        -9.3406 tasteless                               7.8068  pleased
        -8.5563 yuck                                    7.7783  skeptical
```

[5.1.1.1] Calculating sparsity on weight vector obtained using L1 regularization on BOW, SET 1

In [72]:

```
def sparsity(train_data,Y_train,test_data,Y_test):
    C_value = [1000,100,10,1,0.1,0.01,0.001,0.0001]
    for i in C_value[-8:]:
        lr = LogisticRegression(C=i,penalty='l1')
        lr.fit(train_data,Y_train)
        pred = lr.predict(test_data)
        print("C_value = ",i ," and l1_value = l1" )
        print("roc_auc score on test set: %0.3f%%"%(roc_auc_score(Y_test,pred)*100))
        print("Non Zero weights:",np.count_nonzero(lr.coef_))
        print("=+=+"*30)
```

In [73]:

```
sparsity(bow_train,Y_tr,bow_test,Y_test)
```

```
C_value =  1000  and l1_value = l1
roc_auc score on test set: 79.437%
Non Zero weights: 31048
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
C_value =  100  and l1_value = l1
roc_auc score on test set: 82.002%
Non Zero weights: 23428
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
C_value =  10  and l1_value = l1
roc_auc score on test set: 83.880%
Non Zero weights: 10941
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
C_value =  1  and l1_value = l1
roc_auc score on test set: 82.315%
Non Zero weights: 2025
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
C_value =  0.1  and l1_value = l1
roc_auc score on test set: 74.968%
Non Zero weights: 393
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
C_value =  0.01  and l1_value = l1
roc_auc score on test set: 54.765%
Non Zero weights: 35
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
C_value =  0.001  and l1_value = l1
roc_auc score on test set: 50.000%
Non Zero weights: 1
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
C_value =  0.0001  and l1_value = l1
roc_auc score on test set: 50.000%
Non Zero weights: 0
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Pertubation Test on BoW

In [75]:

```
from scipy.sparse import find
logreg1 = LogisticRegression(C= 5, penalty= 'l2')
logreg1.fit(bow_train,Y_tr)
weights1 = find(logreg1.coef_[0])[2]  #Weights before adding random noise
print("Non Zero weights:",np.count_nonzero(logreg1.coef_))
```

```
Non Zero weights: 77052
```

In [76]:

```
bow_train_noise = bow_train
#Random noise
epsilon = np.random.uniform(low=-0.00001, high=0.00001, size=(find(bow_train_noise)[0].size
a,b,c = find(bow_train)
bow_train_noise[a,b] = epsilon + bow_train[a,b]
```

In [77]:

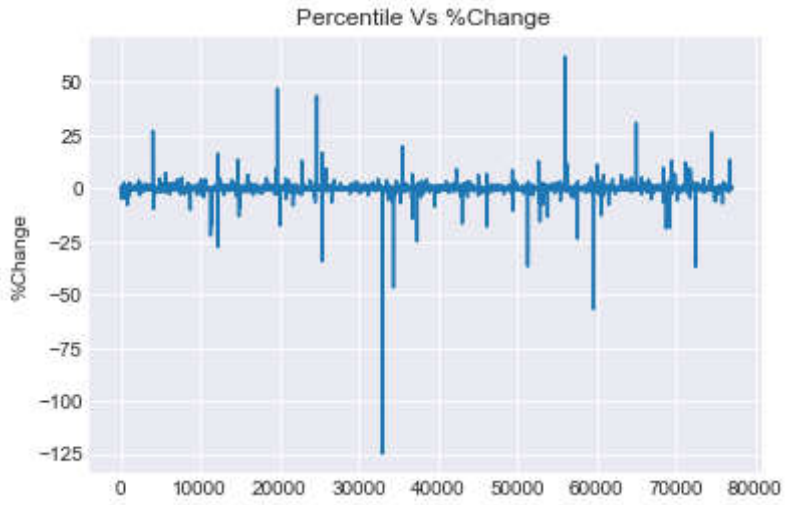
```

logreg2 = LogisticRegression(C= 5, penalty= 'l2')
logreg2.fit(bow_train_noise,Y_tr)
print("Non Zero weights:",np.count_nonzero(logreg2.coef_))
weights2 = find(logreg2.coef_[0])[2]  #Weights after adding random noise
```

```
Non Zero weights: 77052
```

In [85]:

```
percentage_change_vector = (abs(weights1 - weights2)/weights1) * 100
plt.plot(percentage_change_vector)
plt.ylabel('%Change')
plt.title('Percentile Vs %Change ')
plt.show()
```



In [86]:

```
print(percentage_change_vector[np.where(percentage_change_vector > 10)].size)
```

```
17
```

Top 10 Collinear feature after Pertubation test

In [88]:

```
index = np.argsort(np.abs(weights1 - weights2))[::-1]
features = count_vect.get_feature_names()
features = np.array(features)
a = features[index]
print(a[:10])
```

```
['bit' 'thamk' 'skip' 'adverage' 'ic' 'hopes' 'inferior' 'trifles'
 'thrash' 'ovaeasy']
```

TF-IDF

In [126]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
tfidf_train = tf_idf_vect.fit_transform(X_tr)
print("The type of count vectorizer ",type(tfidf_train))
print("The shape of out text TFIDF vectorizer ",tfidf_train.get_shape())
tfidf_cv = tf_idf_vect.transform(X_cv)
tfidf_test = tf_idf_vect.transform(X_test)
print("CV Data Size: ",tfidf_cv.shape)
print("Test Data Size: ",tfidf_test.shape)
```

```
The type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
The shape of out text TFIDF vectorizer  (163475, 2241966)
CV Data Size:  (80519, 2241966)
Test Data Size:  (120177, 2241966)
```

In [127]:

```
#Normalize Data
from sklearn import preprocessing
from sklearn.preprocessing import Normalizer
tfidf_train=preprocessing.normalize(tfidf_train)
tfidf_cv=preprocessing.normalize(tfidf_cv)
tfidf_test=preprocessing.normalize(tfidf_test)
print("The shape of out text BOW vectorizer ",tfidf_train.get_shape())
print("CV Data Size: ",tfidf_cv.shape)
print("Test Data Size: ",tfidf_test.shape)
```

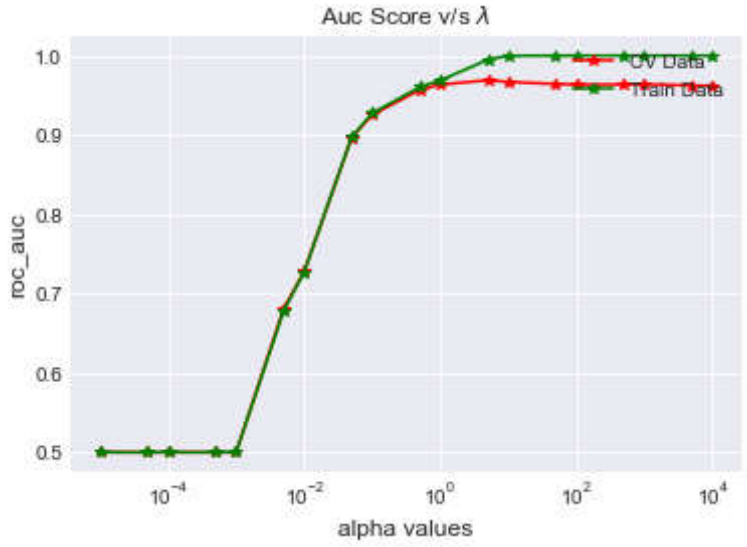
```
The shape of out text BOW vectorizer  (163475, 2241966)
CV Data Size:  (80519, 2241966)
Test Data Size:  (120177, 2241966)
```

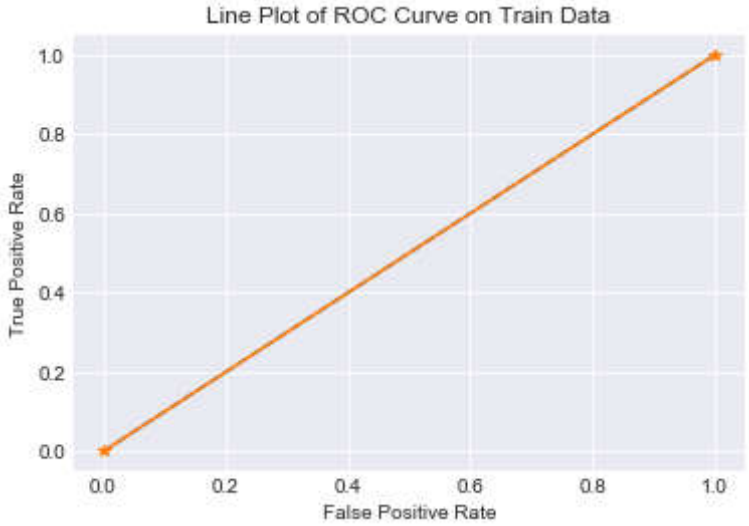
In [128]:

```
# find optimal_C using l1 regularization
logistic_l1(tfidf_train,tfidf_cv,Y_tr,Y_cv)
```

```
10000  for CV data auc score is -->  0.9618489654559422  and for train data
auc score is -->  0.9999963214694778
5000  for CV data auc score is -->  0.9623577755688983  and for train data a
uc score is -->  0.9999963214694778
1000  for CV data auc score is -->  0.9638692371798849  and for train data a
uc score is -->  0.9999963214694778
500  for CV data auc score is -->  0.9636182811687666  and for train data au
c score is -->  0.9999963214694778
100  for CV data auc score is -->  0.9634868041224727  and for train data au
c score is -->  0.9999957379218019
50  for CV data auc score is -->  0.9642021471520364  and for train data auc
score is -->  0.999957914349075
10  for CV data auc score is -->  0.9668652557374677  and for train data auc
score is -->  0.999734401007564
5  for CV data auc score is -->  0.9688293853514832  and for train data auc
score is -->  0.9947144638364804
1  for CV data auc score is -->  0.9634052652428797  and for train data auc
score is -->  0.9685848198721965
0.5  for CV data auc score is -->  0.9566151302878967  and for train data au
c score is -->  0.9602821304365812
0.1  for CV data auc score is -->  0.9252448491547378  and for train data au
c score is -->  0.9272411781309435
0.05  for CV data auc score is -->  0.8965776586818709  and for train data a
uc score is -->  0.8973940427058933
0.01  for CV data auc score is -->  0.7276475150075509  and for train data a
uc score is -->  0.7269464075590404
0.005  for CV data auc score is -->  0.6812005472968393  and for train data
auc score is -->  0.6782085161379245
0.001  for CV data auc score is -->  0.5  and for train data auc score is --
>  0.5
0.0005  for CV data auc score is -->  0.5  and for train data auc score is -
->  0.5
0.0001  for CV data auc score is -->  0.5  and for train data auc score is -
->  0.5
5e-05  for CV data auc score is -->  0.5  and for train data auc score is --
>  0.5
1e-05  for CV data auc score is -->  0.5  and for train data auc score is --
>  0.5

 Best C Value 5 with highest roc_auc Score is 0.9688293853514832
```


Auc Score v/s λ
CV Data
Train Data
roc_auc
alpha values

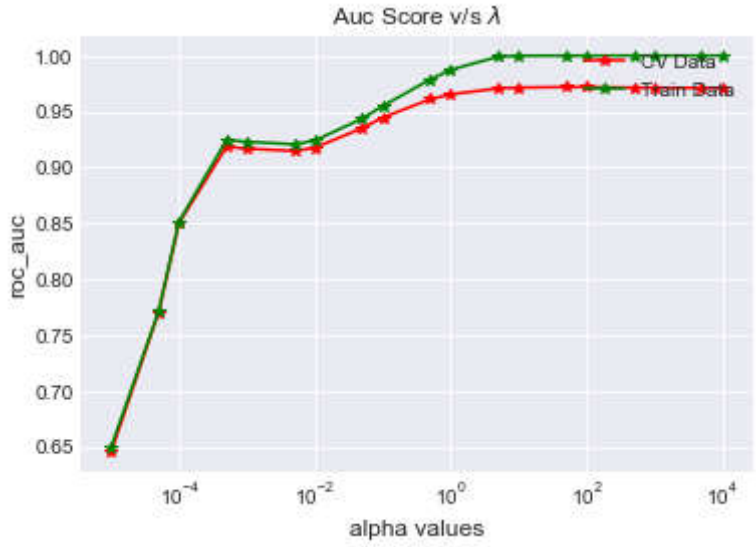

Line Plot of ROC Curve on Train Data
True Positive Rate
False Positive Rate

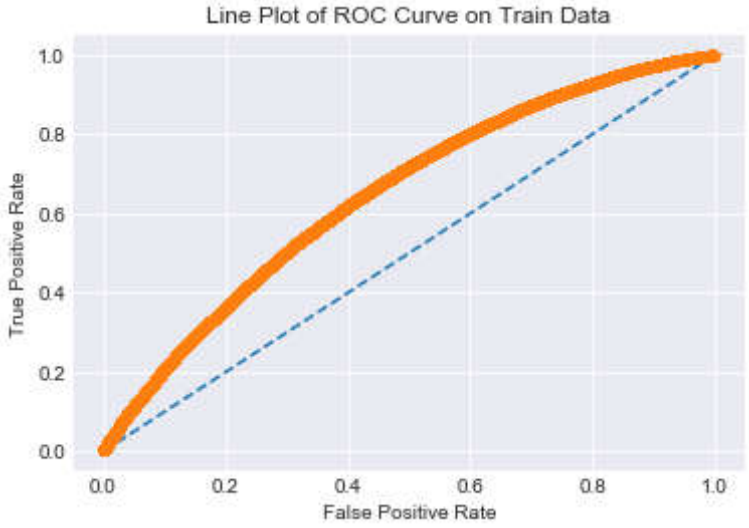
In [129]:

```
# find optimal_C using l2 regularization
logistic_l2(tfidf_train,tfidf_cv,Y_tr,Y_cv)
```

```
10000  for CV data auc score is -->  0.9709659332296431  and for train data
auc score is -->  0.9999962866434884
5000  for CV data auc score is -->  0.97101796732311  and for train data auc
score is -->  0.9999962857940741
1000  for CV data auc score is -->  0.9712049017122092  and for train data a
uc score is -->  0.9999962849446596
500  for CV data auc score is -->  0.9713395628083671  and for train data au
c score is -->  0.9999956906377349
100  for CV data auc score is -->  0.9718350539662859  and for train data au
c score is -->  0.999956892220443
50  for CV data auc score is -->  0.9718779222642138  and for train data auc
score is -->  0.9999344110533046
10  for CV data auc score is -->  0.9713885998731275  and for train data auc
score is -->  0.9998117417437136
5  for CV data auc score is -->  0.9706021882174269  and for train data auc
score is -->  0.9990201402094333
1  for CV data auc score is -->  0.9652961365002775  and for train data auc
score is -->  0.9872061690927498
0.5  for CV data auc score is -->  0.960932242265111  and for train data auc
score is -->  0.9785748777600694
0.1  for CV data auc score is -->  0.9444586495188846  and for train data au
c score is -->  0.9548144260347091
0.05  for CV data auc score is -->  0.9349914767130096  and for train data a
uc score is -->  0.943494838717587
0.01  for CV data auc score is -->  0.9175437523158672  and for train data a
uc score is -->  0.9238874708859683
0.005  for CV data auc score is -->  0.9145939537802172  and for train data
auc score is -->  0.9206541357094058
0.001  for CV data auc score is -->  0.9166796629760079  and for train data
auc score is -->  0.9228093751903219
0.0005  for CV data auc score is -->  0.9183717061194697  and for train data
auc score is -->  0.92439926428257
0.0001  for CV data auc score is -->  0.849980894645525  and for train data
auc score is -->  0.8511599002459083
5e-05  for CV data auc score is -->  0.7699351040335499  and for train data
auc score is -->  0.7708757875593129
1e-05  for CV data auc score is -->  0.6449437075930697  and for train data
auc score is -->  0.6492032544122618

 Best C Value 50 with highest roc_auc Score is 0.9718779222642138
```





Logistic Regression Over the unseen or Test data in TF-IDF

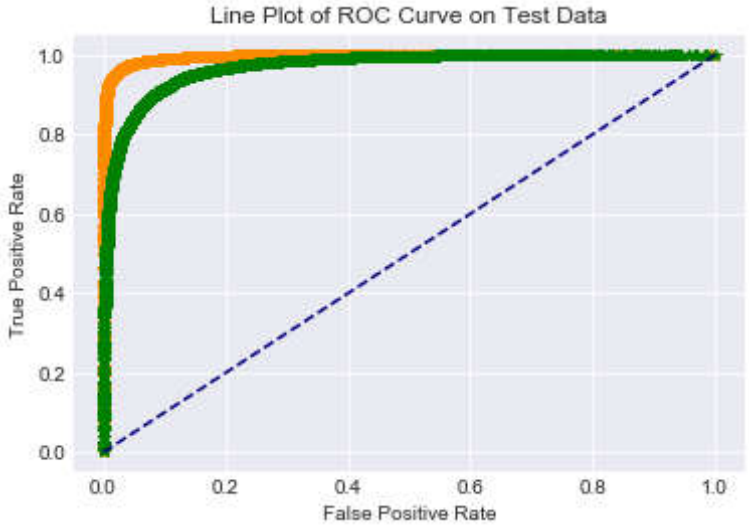
In [130]:

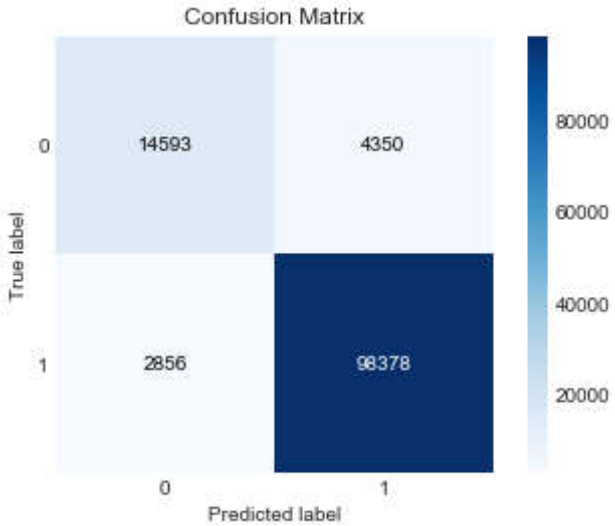
```
testing_l1(tfidf_train,Y_tr,tfidf_test,Y_test,optimal_C=5)
```

```
AUC Score 0.9947145181990004
AUC Score 0.9686596191989425
macro f1 score for data : 0.8833296688597994
micro f1 scoore for data: 0.9400384432961382
hamming loss for data: 0.0599615567038618
Precision recall report for data:
              precision    recall  f1-score   support

           0       0.84      0.77      0.80     18943
           1       0.96      0.97      0.96    101234

avg / total       0.94      0.94      0.94    120177
```





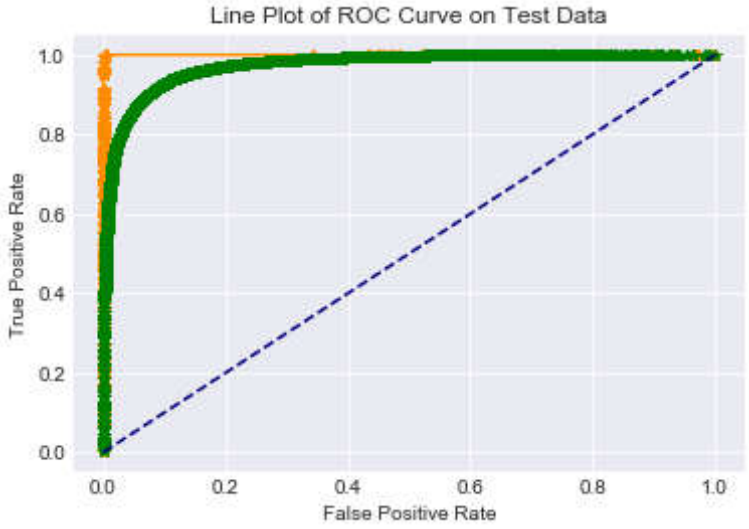
In [131]:

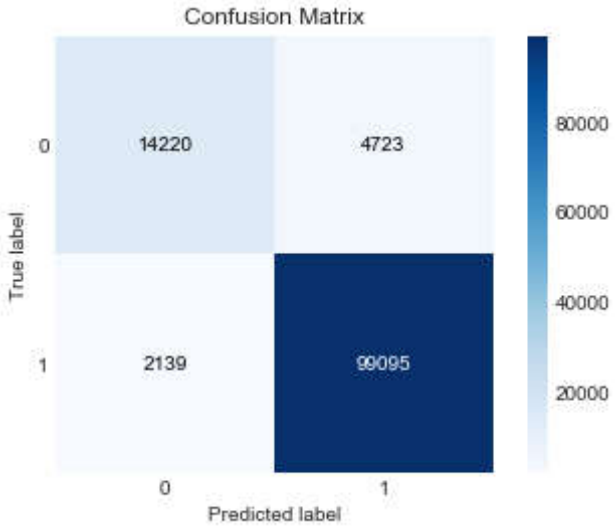
```
testing_l2(tfidf_train,Y_tr,tfidf_test,Y_test,optimal_C=50)
```

```
AUC Score 0.9999344110533046
AUC Score 0.972004740653584
macro f1 score for data : 0.8860776980265641
micro f1 scoore for data: 0.9429008878570775
hamming loss for data: 0.05709911214292252
Precision recall report for data:
              precision    recall  f1-score   support

           0       0.87      0.75      0.81     18943
           1       0.95      0.98      0.97    101234

avg / total       0.94      0.94      0.94    120177
```





In [96]:

```
clf1=LogisticRegression(penalty='l1',C=5)
clf1.fit(tfidf_train,Y_tr)
imp_feature(tf_idf_vect,clf1)
```

```
                        Negative
Positive
-------------------------------------------------------------------------------
-------------------
        -53.8849        two stars                               39.2878 not
disappointed
        -33.6487        great strength                          34.9987 deli
cious
        -33.3007        wanted like                             33.8910 grea
t
        -30.7574        not worth                               30.2209 perf
ect
        -30.3322        not recommend                           28.2921 four
stars
        -29.5098        no thanks                               26.9497 best
        -29.0162        disappointed                            26.3758 good
        -28.5398        worst                                   24.5539 plea
santly surprised
        -28.2912        rather iced                             24.0151 exce
llent
        -28.2322        disappointment                          23.6740 neve
r disappointed
```

In [97]:

```
clf2=LogisticRegression(penalty='l2',C=50)
clf2.fit(tfidf_train,Y_tr)
imp_feature(tf_idf_vect,clf2)
```

```
                        Negative
Positive
-------------------------------------------------------------------------------
-------------------
        -28.2581        disappointed                            33.9787 grea
t
        -27.5421        worst                                   31.3326 deli
cious
        -25.3006        not worth                               25.8766 not
disappointed
        -25.2706        disappointing                           25.8535 best
        -24.9045        not recommend                           25.6850 perf
ect
        -23.4328        terrible                                24.2982 good
        -23.2047        disappointment                          22.0637 exce
llent
        -23.1985        not good                                19.8906 wond
erful
        -21.9575        awful                                   18.4151 amaz
ing
        -21.2778        two stars                               18.2722 happ
y
```

Working with word2vec

Preparing Reviews for gensim model

In [81]:

```
i=0
list_of_sentance_train=[]
for sentance in X_tr:
    list_of_sentance_train.append(sentance.split())
```

Training w2v model

In [82]:

```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
```

In [83]:

```
# this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=4)
```

In [84]:

```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  23535
sample words  ['worst', 'soup', 'ever', 'much', 'sugar', 'remembering', 'mak
es', 'go', 'sick', 'tummy', 'guess', 'stick', 'buying', 'generally', 'buy',
'safeway', 'awfully', 'sweet', 'unpalatable', 'not', 'taste', 'like', 'fres
h', 'pad', 'thai', 'noodles', 'still', 'delicious', 'tanginess', 'tamarind',
'add', 'teaspoon', 'crunchy', 'peanut', 'butter', 'final', 'stir', 'kicks',
'richness', 'bit', 'love', 'individual', 'size', 'rolls', 'worry', 'refriger
ating', 'cut', 'roll', 'little', 'pieces']
```

Avg W2V

Converting Reviews into Numerical Vectors using W2V vectors

Algorithm: Avg W2V

In [85]:

```
from tqdm import tqdm
import numpy as np
```

Converting Train data text

In [86]:

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance_train): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to chan
    cnt_words =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]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_train.append(sent_vec)
sent_vectors_train = np.array(sent_vectors_train)
print(sent_vectors_train.shape)
print(sent_vectors_train[0])
```

```
  0%|          | 0/163475 [00:00<?, ?it/s]
  0%|          | 38/163475 [00:00<07:14, 376.47it/s]
  0%|          | 53/163475 [00:00<10:33, 257.85it/s]
  0%|          | 66/163475 [00:00<15:25, 176.47it/s]
  0%|          | 87/163475 [00:00<14:53, 182.96it/s]
  0%|          | 110/163475 [00:00<14:00, 194.45it/s]
  0%|          | 131/163475 [00:00<13:50, 196.67it/s]
  0%|          | 149/163475 [00:00<15:59, 170.23it/s]
  0%|          | 166/163475 [00:00<16:05, 169.18it/s]
  0%|          | 183/163475 [00:00<16:09, 168.45it/s]
  0%|          | 200/163475 [00:01<16:23, 165.97it/s]
  0%|          | 217/163475 [00:01<17:16, 157.44it/s]
  0%|          | 240/163475 [00:01<17:18, 157.18it/s]
  0%|          | 256/163475 [00:01<17:21, 156.65it/s]
  0%|          | 272/163475 [00:01<18:00, 150.98it/s]
  0%|          | 293/163475 [00:01<16:34, 164.13it/s]
  0%|          | 310/163475 [00:01<17:38, 154.15it/s]
  0%|          | 326/163475 [00:01<21:46, 124.88it/s]
```

Converting CV data text

In [43]:

```
i=0
list_of_sentance_cv=[]
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())
```

In [44]:

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance_cv): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to chan
    cnt_words =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]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_cv.append(sent_vec)
sent_vectors_cv = np.array(sent_vectors_cv)
print(sent_vectors_cv.shape)
print(sent_vectors_cv[0])
```

```
100%|██████████| 80519/80519 [07:52<00:00, 170.54it/s]

(80519, 50)
[-0.23630953 -0.15764741 -0.45580933 -0.0857438  -0.12027711 -0.29373519
 -0.69836357 -0.37131124  0.51424202  0.11287925 -0.93952744  0.23272125
  0.20223512 -0.98464972  0.68767115  0.83059     0.4071628   0.4043956
 -0.07656631 -0.14552595  0.03374348 -0.15886163  0.07129767  0.37491366
 -0.38337408 -0.59502741 -0.34748336 -0.319286    0.12662574 -0.51776885
  0.73546405 -0.24740469  0.04701688  0.89332737 -0.6708107   0.19396324
 -0.70647179 -0.38174529  0.53726104  0.24764472 -0.65073272 -0.27263675
  0.32562323 -0.1359019  -0.35391434  0.14392293 -0.35838907 -0.00706661
  0.44748637 -0.18430013]
```

Converting Test data text

In [45]:

```
i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())
```

In [46]:

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance_test): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to chan
    cnt_words =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]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_test.append(sent_vec)
sent_vectors_test = np.array(sent_vectors_test)
print(sent_vectors_test.shape)
print(sent_vectors_test[0])
```

```
100%|██████████| 120177/120177 [11:48<00:00, 169.52it/s]

(120177, 50)
[ 0.3682589  -0.26585959 -0.80323109 -1.34686356 -0.49062624  0.32282951
 -0.1220367   1.15675126  0.52712699 -0.45334991  0.22349855 -0.18240635
  0.24125669 -0.32660102 -0.14614465  0.09152282  0.17673497 -0.61236313
 -0.00448281 -0.40608427  0.48149021 -0.83567413 -0.44427658  1.15761499
  0.32851466 -0.10359561  0.51606744  0.45280914  0.2452846  -0.56378139
  0.06070574  0.34250891 -0.07639216 -0.04101486 -0.15731841 -0.18202841
 -0.09111401  0.37374244  0.10098203  0.20291001 -1.01396451  0.15584724
  0.78863963  0.3705341  -0.19290103 -0.41138454  0.4254688   0.33931866
  0.22082883 -0.63015809]
```

In [132]:

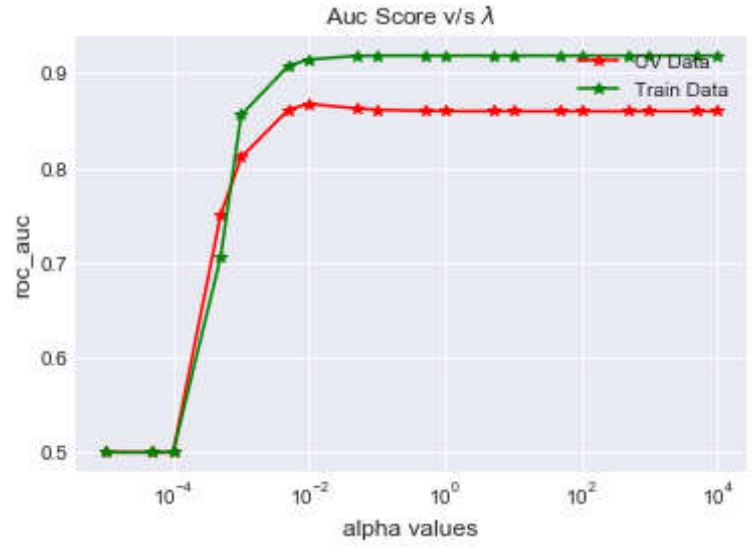
```
from sklearn import preprocessing
from sklearn.preprocessing import Normalizer
normalizer = preprocessing.Normalizer()
train_avgw2v = normalizer.fit_transform(sent_vectors_train)
cv_avgw2v = normalizer.transform(sent_vectors_cv)
test_avgw2v = normalizer.transform(sent_vectors_test)
```

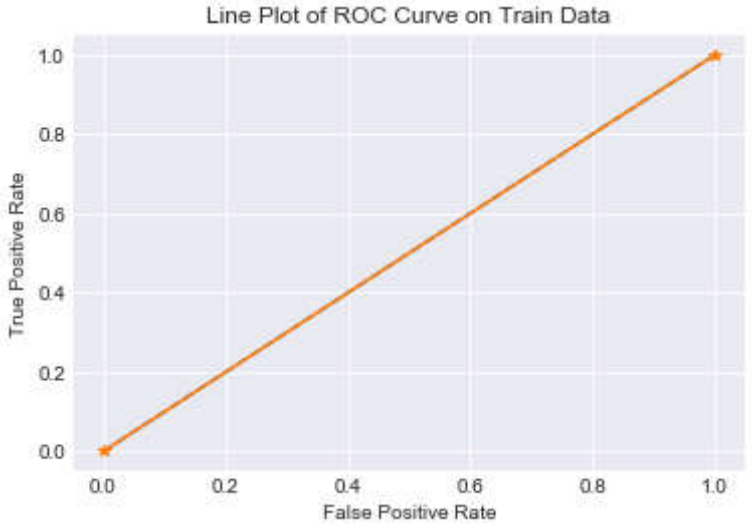
In [133]:

```
# find optimal_C using l1 regularization
logistic_l1(train_avgw2v,cv_avgw2v,Y_tr,Y_cv)
```

```
10000  for CV data auc score is -->  0.8596728950238729  and for train data
auc score is -->  0.9184120402109367
5000  for CV data auc score is -->  0.8597864033822129  and for train data a
uc score is -->  0.9184085978176109
1000  for CV data auc score is -->  0.8596776696329855  and for train data a
uc score is -->  0.9184118482432879
500  for CV data auc score is -->  0.8596715782542184  and for train data au
c score is -->  0.9184120540847049
100  for CV data auc score is -->  0.8596684158964635  and for train data au
c score is -->  0.918412287107382
50  for CV data auc score is -->  0.859676239126326  and for train data auc
score is -->  0.9184122454860775
10  for CV data auc score is -->  0.859689138309737  and for train data auc
score is -->  0.9184131212322987
5  for CV data auc score is -->  0.8597132271038462  and for train data auc
score is -->  0.9184144347100613
1  for CV data auc score is -->  0.8598700094611835  and for train data auc
score is -->  0.9184225553946292
0.5  for CV data auc score is -->  0.8600500598301202  and for train data au
c score is -->  0.9184294936943865
0.1  for CV data auc score is -->  0.8612673283662045  and for train data au
c score is -->  0.9184105622299235
0.05  for CV data auc score is -->  0.8626645393969532  and for train data a
uc score is -->  0.9182232241725437
0.01  for CV data auc score is -->  0.8674162486777018  and for train data a
uc score is -->  0.9142004956559392
0.005  for CV data auc score is -->  0.8615881652139137  and for train data
auc score is -->  0.9080024305142941
0.001  for CV data auc score is -->  0.8109594612892024  and for train data
auc score is -->  0.8556606089547925
0.0005  for CV data auc score is -->  0.7504540422274592  and for train data
auc score is -->  0.7059564427103556
0.0001  for CV data auc score is -->  0.5  and for train data auc score is -
-> 0.5
5e-05  for CV data auc score is -->  0.5  and for train data auc score is --
> 0.5
1e-05  for CV data auc score is -->  0.5  and for train data auc score is --
> 0.5

 Best C Value 0.01 with highest roc_auc Score is 0.8674162486777018
```


Auc Score v/s λ
CV Data
Train Data
roc_auc
alpha values

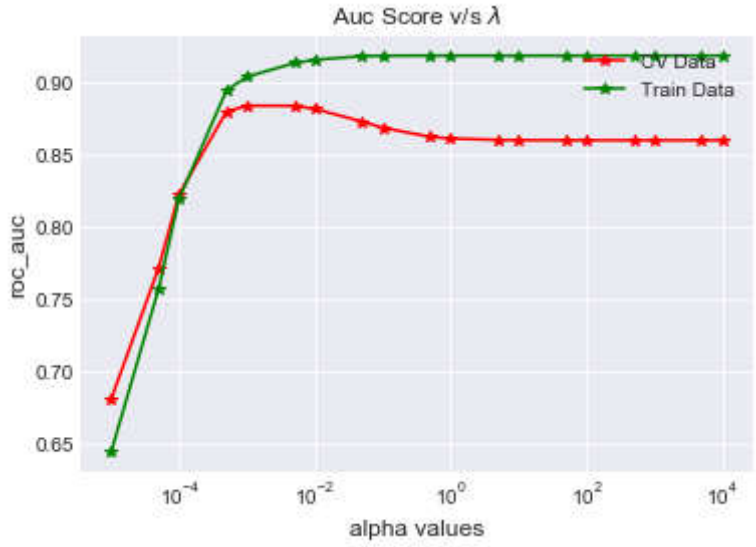

Line Plot of ROC Curve on Train Data
True Positive Rate
False Positive Rate

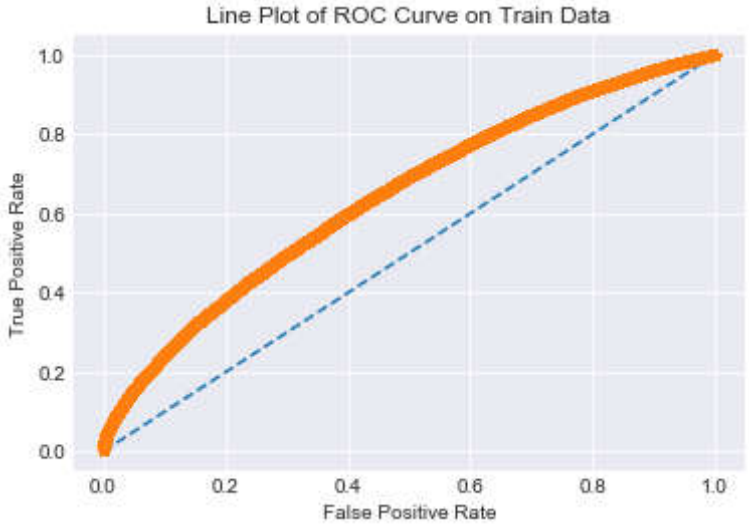
In [134]:

```
# find optimal_C using l2 regularization
logistic_l2(train_avgw2v,cv_avgw2v,Y_tr,Y_cv)
```

```
10000  for CV data auc score is -->  0.8597699267596067  and for train data
auc score is -->  0.9184101596075095
5000  for CV data auc score is -->  0.8597700686377263  and for train data a
uc score is -->  0.918410165270272
1000  for CV data auc score is -->  0.8597711591059176  and for train data a
uc score is -->  0.9184101145885475
500  for CV data auc score is -->  0.8597724676677471  and for train data au
c score is -->  0.9184101185524812
100  for CV data auc score is -->  0.8597834825690254  and for train data au
c score is -->  0.9184102473803282
50  for CV data auc score is -->  0.8597967534496586  and for train data auc
score is -->  0.9184104653966846
10  for CV data auc score is -->  0.8599038303907631  and for train data auc
score is -->  0.9184124244293725
5  for CV data auc score is -->  0.8600372438975067  and for train data auc
score is -->  0.9184149276535374
1  for CV data auc score is -->  0.8610461966601177  and for train data auc
score is -->  0.9184350565091317
0.5  for CV data auc score is -->  0.862194323650143  and for train data auc
score is -->  0.9184462362180041
0.1  for CV data auc score is -->  0.8684822974167709  and for train data au
c score is -->  0.9183605991100798
0.05  for CV data auc score is -->  0.8727234799765722  and for train data a
uc score is -->  0.9180642797726322
0.01  for CV data auc score is -->  0.8815787863901371  and for train data a
uc score is -->  0.9155034958922745
0.005  for CV data auc score is -->  0.8836658369790578  and for train data
auc score is -->  0.9132428338943857
0.001  for CV data auc score is -->  0.8837918294393683  and for train data
auc score is -->  0.9037760733051401
0.0005  for CV data auc score is -->  0.8796413245861306  and for train data
auc score is -->  0.8942742746560431
0.0001  for CV data auc score is -->  0.8223121415387752  and for train data
auc score is -->  0.8189899910955891
5e-05  for CV data auc score is -->  0.7711297412507527  and for train data
auc score is -->  0.7566189304281775
1e-05  for CV data auc score is -->  0.680320111487029  and for train data a
uc score is -->  0.6437144652804847

 Best C Value 0.001 with highest roc_auc Score is 0.8837918294393683
```


Auc Score v/s λ
CV Data
Train Data
roc_auc
0.90
0.85
0.80
0.75
0.70
0.65
10^{-4}
10^{-2}
10^{0}
10^{2}
10^{4}
alpha values


Line Plot of ROC Curve on Train Data
True Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

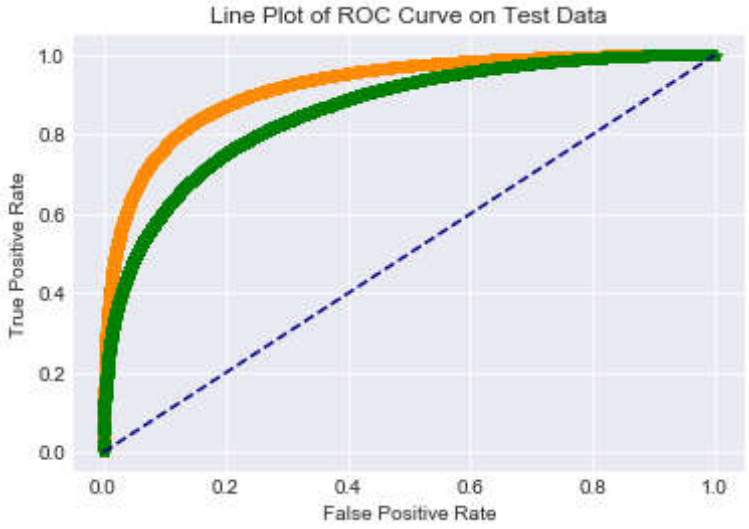
In [135]:

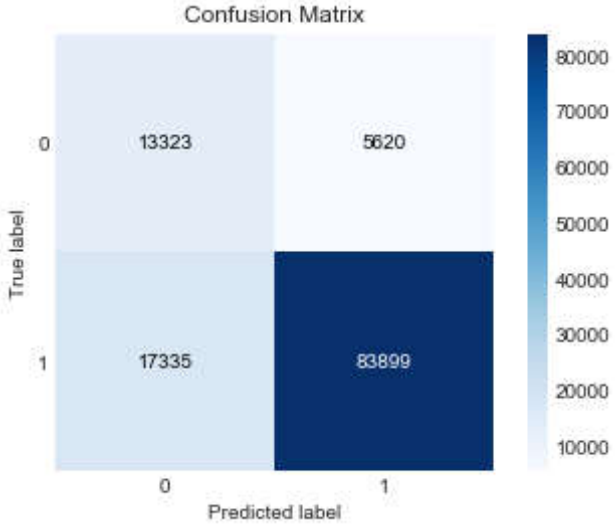
```
testing_l1(train_avgw2v,Y_tr,test_avgw2v,Y_test,optimal_C=0.5)
```

```
AUC Score 0.9184291380729013
AUC Score 0.8587043091440142
macro f1 score for data : 0.7084340218051193
micro f1 scoore for data: 0.8089900729756941
hamming loss for data: 0.1910099270243058
Precision recall report for data:
              precision    recall  f1-score   support

           0       0.43      0.70      0.54     18943
           1       0.94      0.83      0.88    101234

avg / total       0.86      0.81      0.83    120177
```





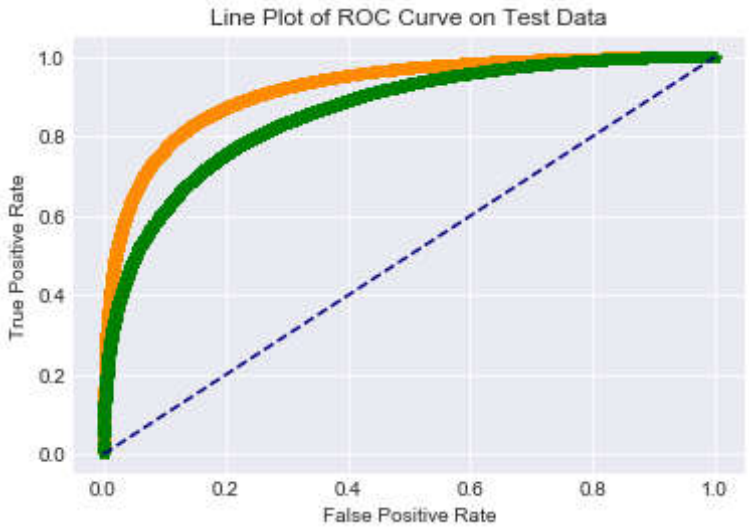
In [136]:

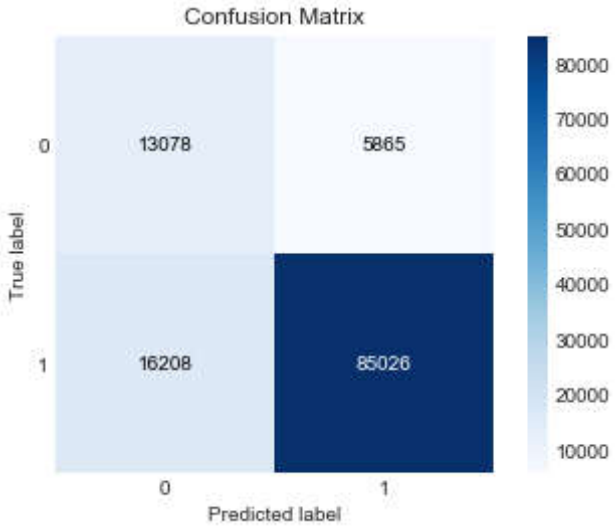
```
testing_l2(train_avgw2v,Y_tr,test_avgw2v,Y_test,optimal_C=0.5)
```

```
AUC Score 0.9184462362180041
AUC Score 0.8607555407875849
macro f1 score for data : 0.7137202798193335
micro f1 scoore for data: 0.8163292476929862
hamming loss for data: 0.18367075230701382
Precision recall report for data:
              precision    recall  f1-score   support

           0       0.45      0.69      0.54     18943
           1       0.94      0.84      0.89    101234

avg / total       0.86      0.82      0.83    120177
```





TF-IDF Weighted W2V

In [93]:

```
tf_idf_vect = TfidfVectorizer()
tfidf_train = tf_idf_vect.fit_transform(X_tr)
print("The type of count vectorizer ",type(tfidf_train))
print("The shape of out text TFIDF vectorizer ",tfidf_train.get_shape())
tfidf_cv = tf_idf_vect.transform(X_cv)
tfidf_test = tf_idf_vect.transform(X_test)
print("CV Data Size: ",tfidf_cv.shape)
print("Test Data Size: ",tfidf_test.shape)
```

```
The type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
The shape of out text TFIDF vectorizer  (163475, 77052)
CV Data Size:  (80519, 77052)
Test Data Size:  (120177, 77052)
```

In [96]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_tr)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [97]:

```
from tqdm import tqdm
import numpy as np
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors_train = []; # the tfidf-w2v for each sentence/review is stored in this l
row=0;
for sent in tqdm(list_of_sentance_train): # 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 and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_train.append(sent_vec)
    row += 1
```

```
  0%|          | 0/163475 [00:00<?, ?it/s]
  0%|          | 2/163475 [00:00<3:10:10, 14.33it/s]
  0%|          | 3/163475 [00:00<4:12:34, 10.79it/s]
  0%|          | 4/163475 [00:00<4:36:29,  9.85it/s]
  0%|          | 5/163475 [00:00<6:48:57,  6.66it/s]
  0%|          | 7/163475 [00:00<6:05:54,  7.45it/s]
  0%|          | 10/163475 [00:01<5:52:14,  7.73it/s]
  0%|          | 12/163475 [00:01<5:22:07,  8.46it/s]
  0%|          | 13/163475 [00:01<5:25:08,  8.38it/s]
  0%|          | 14/163475 [00:01<5:24:47,  8.39it/s]
  0%|          | 16/163475 [00:01<4:31:52, 10.02it/s]
  0%|          | 18/163475 [00:02<5:56:31,  7.64it/s]
  0%|          | 20/163475 [00:02<5:38:35,  8.05it/s]
  0%|          | 23/163475 [00:02<4:54:27,  9.25it/s]
  0%|          | 26/163475 [00:02<4:02:35, 11.23it/s]
  0%|          | 28/163475 [00:03<4:43:44,  9.60it/s]
  0%|          | 30/163475 [00:03<4:25:11, 10.27it/s]
  0%|          | 33/163475 [00:03<3:56:15, 11.53it/s]
```

In [98]:

```
i=0
list_of_sentance_cv=[]
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())
```

In [99]:

```
w2v_model=Word2Vec(list_of_sentance_cv,min_count=5,size=50, workers=4)
print(w2v_model.wv.most_similar('great'))
print('='*50)
```

```
[('awesome', 0.8799434900283813), ('fantastic', 0.8753538131713867), ('terri
fic', 0.8236940503120422), ('good', 0.8167998790740967), ('excellent', 0.798
1976270675659), ('wonderful', 0.7920230627059937), ('perfect', 0.71222174167
63306), ('amazing', 0.6930117607116699), ('decent', 0.6664069890975952), ('f
abulous', 0.6529747843742371)]
==================================================
```

In [100]:

```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  17246
sample words  ['ribena', 'litre', 'case', 'packed', 'well', 'received', 'tim
ely', 'manner', 'happy', 'concentrate', 'nice', 'strong', 'means', 'need',
'little', 'bit', 'water', 'worth', 'every', 'cent', 'american', 'shops', 'no
t', 'able', 'find', 'cordial', 'concentrates', 'really', 'missing', 'thank',
'goodness', 'amazon', 'first', 'came', 'across', 'tea', 'upscale', 'restaura
nt', 'asked', 'decaf', 'dinner', 'liked', 'much', 'copied', 'information',
'bag', 'could', 'locally', 'ordered', 'directly']
```

In [101]:

```
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform( X_cv)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [102]:

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

tfidf_sent_vectors_cv = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance_cv): # 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 and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_cv.append(sent_vec)
    row += 1
```

```
  0%|          | 0/80519 [00:00<?, ?it/s]
  0%|          | 2/80519 [00:00<1:51:18, 12.06it/s]
  0%|          | 5/80519 [00:00<1:57:04, 11.46it/s]
  0%|          | 8/80519 [00:00<1:39:23, 13.50it/s]
  0%|          | 10/80519 [00:00<1:47:35, 12.47it/s]
  0%|          | 12/80519 [00:00<1:52:43, 11.90it/s]
  0%|          | 14/80519 [00:01<1:48:52, 12.32it/s]
  0%|          | 16/80519 [00:01<1:37:55, 13.70it/s]
  0%|          | 18/80519 [00:01<1:35:54, 13.99it/s]
  0%|          | 20/80519 [00:01<1:50:58, 12.09it/s]
  0%|          | 22/80519 [00:01<1:38:11, 13.66it/s]
  0%|          | 24/80519 [00:01<1:35:28, 14.05it/s]
  0%|          | 26/80519 [00:01<1:37:00, 13.83it/s]
  0%|          | 29/80519 [00:02<1:24:31, 15.87it/s]
  0%|          | 31/80519 [00:02<1:39:47, 13.44it/s]
  0%|          | 33/80519 [00:02<1:40:57, 13.29it/s]
  0%|          | 36/80519 [00:02<1:33:15, 14.38it/s]
  0%|          | 38/80519 [00:02<1:38:42, 13.59it/s]
```

In [103]:

```
i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())
```

In [105]:

```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  20561
sample words  ['month', 'old', 'shep', 'mix', 'puppy', 'destroys', 'virtuall
y', 'toy', 'within', 'minutes', 'getting', 'new', 'shreds', 'pieces', 'eve
n', 'made', 'powerful', 'chewers', 'toys', 'galileo', 'bones', 'couple', 'mo
nths', 'gnawed', 'ends', 'still', 'plenty', 'left', 'keep', 'chewing', 'hel
d', 'no', 'including', 'couches', 'several', 'pairs', 'shoes', 'grateful',
'found', 'product', 'nothing', 'else', 'able', 'withstand', 'boy', 'jaws',
'highly', 'recommend', 'simply', 'best']
```

In [104]:

```
w2v_model=Word2Vec(list_of_sentance_test,min_count=5,size=50, workers=4)
print(w2v_model.wv.most_similar('great'))
print('='*50)
```

```
[('fantastic', 0.8609011769294739), ('awesome', 0.8468232750892639), ('terri
fic', 0.8324236869812012), ('good', 0.8319335579872131), ('excellent', 0.818
9373016357422), ('wonderful', 0.7887812256813049), ('perfect', 0.76424908638
00049), ('nice', 0.7176415920257568), ('amazing', 0.7098631262779236), ('fab
ulous', 0.7076770067214966)]
==================================================
```

In [106]:

```
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform( X_test )
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [107]:

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

tfidf_sent_vectors_test = []; # the tfidf-w2v for each sentence/review is stored in this li
row=0;
for sent in tqdm(list_of_sentance_test): # 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 and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_test.append(sent_vec)
    row += 1
```

```
  0%|          | 0/120177 [00:00<?, ?it/s]
  0%|          | 1/120177 [00:00<6:06:18,  5.47it/s]
  0%|          | 3/120177 [00:00<4:54:14,  6.81it/s]
  0%|          | 4/120177 [00:00<5:13:27,  6.39it/s]
  0%|          | 6/120177 [00:00<4:20:14,  7.70it/s]
  0%|          | 7/120177 [00:00<4:20:50,  7.68it/s]
  0%|          | 10/120177 [00:00<3:31:36,  9.46it/s]
  0%|          | 12/120177 [00:01<3:59:55,  8.35it/s]
  0%|          | 14/120177 [00:01<3:19:28, 10.04it/s]
  0%|          | 16/120177 [00:01<3:03:10, 10.93it/s]
  0%|          | 18/120177 [00:01<3:02:50, 10.95it/s]
  0%|          | 20/120177 [00:01<3:21:02,  9.96it/s]
  0%|          | 22/120177 [00:02<3:04:15, 10.87it/s]
  0%|          | 24/120177 [00:02<3:02:24, 10.98it/s]
  0%|          | 26/120177 [00:02<3:36:32,  9.25it/s]
  0%|          | 28/120177 [00:02<3:21:42,  9.93it/s]
  0%|          | 30/120177 [00:02<3:38:02,  9.18it/s]
  0%|          | 31/120177 [00:03<4:35:06,  7.28it/s]
```

In [137]:

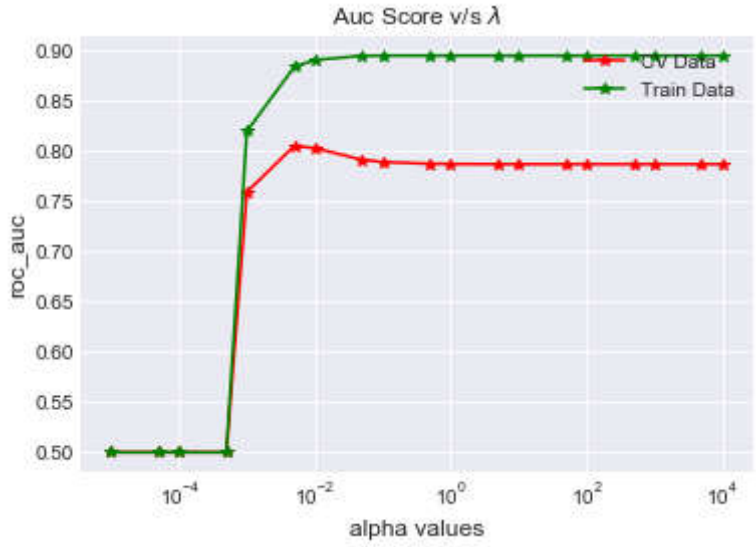
```
normalizer = preprocessing.Normalizer()
train_tfidfw2v = normalizer.fit_transform(tfidf_sent_vectors_train)
cv_tfidfw2v = normalizer.transform(tfidf_sent_vectors_cv)
test_tfidfw2v = normalizer.transform( tfidf_sent_vectors_test)
```

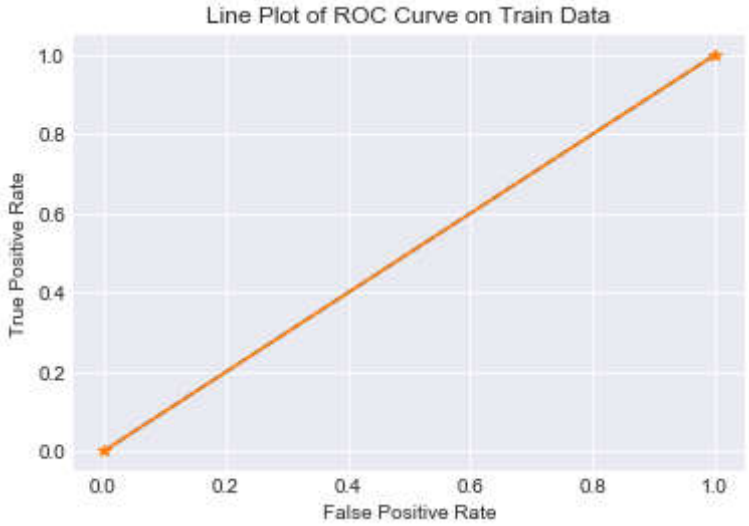
In [138]:

```
# find optimal_C using l1 regularization
logistic_l1(train_tfidfw2v,cv_tfidfw2v,Y_tr,Y_cv)
```

```
10000  for CV data auc score is -->  0.7870487054592462  and for train data
auc score is -->  0.8952835703355166
5000  for CV data auc score is -->  0.7870491135054081  and for train data a
uc score is -->  0.895283269925966
1000  for CV data auc score is -->  0.7870412550991523  and for train data a
uc score is -->  0.8952836527287111
500  for CV data auc score is -->  0.7870479902059165  and for train data au
c score is -->  0.895283763435718
100  for CV data auc score is -->  0.7870524787136973  and for train data au
c score is -->  0.8952835122922009
50  for CV data auc score is -->  0.7870542820901256  and for train data auc
score is -->  0.8952835853418373
10  for CV data auc score is -->  0.7870689740970467  and for train data auc
score is -->  0.8952847343163493
5  for CV data auc score is -->  0.7870883703602942  and for train data auc
score is -->  0.8952862284362358
1  for CV data auc score is -->  0.7872755709174357  and for train data auc
score is -->  0.8952949768380273
0.5  for CV data auc score is -->  0.7875199108352536  and for train data au
c score is -->  0.8953049583063538
0.1  for CV data auc score is -->  0.7892630734859571  and for train data au
c score is -->  0.8953106142735422
0.05  for CV data auc score is -->  0.7914713712397368  and for train data a
uc score is -->  0.8951292235324817
0.01  for CV data auc score is -->  0.8031206541471507  and for train data a
uc score is -->  0.8909520394878019
0.005  for CV data auc score is -->  0.8052951485595471  and for train data
auc score is -->  0.8840592626783094
0.001  for CV data auc score is -->  0.7593322461279479  and for train data
auc score is -->  0.8198780249450918
0.0005  for CV data auc score is -->  0.5  and for train data auc score is -
-> 0.5
0.0001  for CV data auc score is -->  0.5  and for train data auc score is -
-> 0.5
5e-05  for CV data auc score is -->  0.5  and for train data auc score is --
> 0.5
1e-05  for CV data auc score is -->  0.5  and for train data auc score is --
> 0.5

 Best C Value 0.005 with highest roc_auc Score is 0.8052951485595471
```


Auc Score v/s λ
CV Data
Train Data
roc_auc
alpha values
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
10^{-4}
10^{-2}
10^{0}
10^{2}
10^{4}

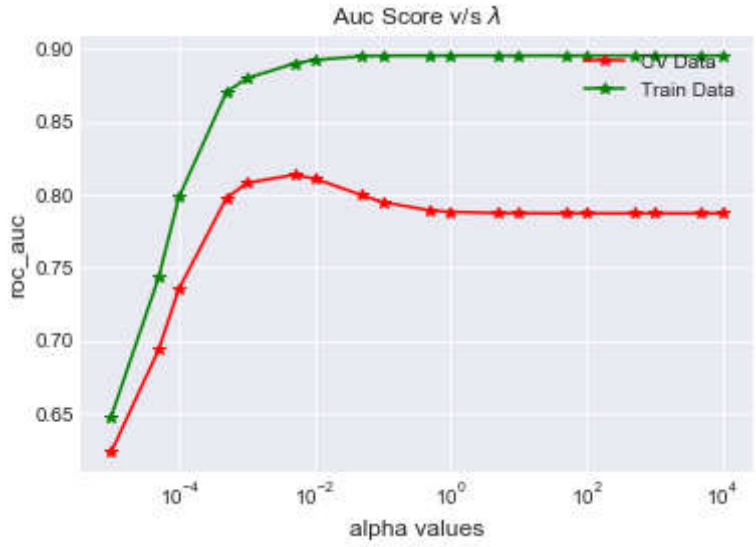

Line Plot of ROC Curve on Train Data
True Positive Rate
False Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

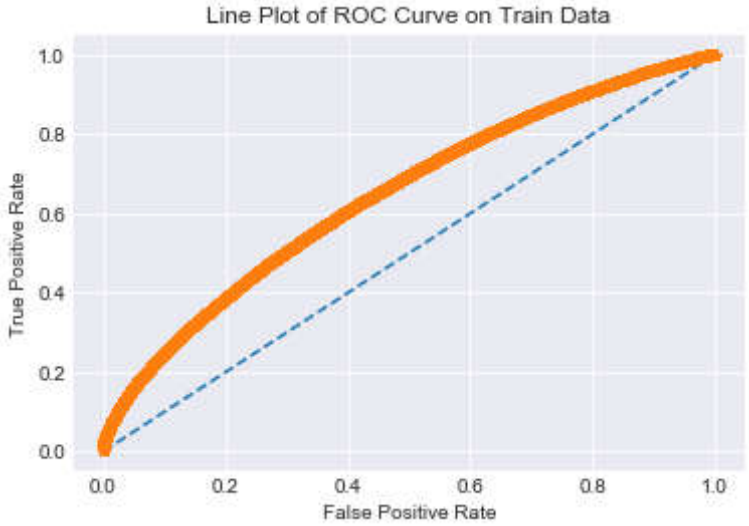
In [139]:

```
# find optimal_C using l1 regularization
logistic_l2(train_tfidfw2v,cv_tfidfw2v,Y_tr,Y_cv)
```

```
10000  for CV data auc score is -->  0.7870694266999735  and for train data
auc score is -->  0.8952821246322495
5000  for CV data auc score is -->  0.787069534574246  and for train data au
c score is -->  0.8952821283130452
1000  for CV data auc score is -->  0.7870703225254554  and for train data a
uc score is -->  0.8952821382228796
500  for CV data auc score is -->  0.7870715701148698  and for train data au
c score is -->  0.8952821294455976
100  for CV data auc score is -->  0.787079909265167  and for train data auc
score is -->  0.8952823485945066
50  for CV data auc score is -->  0.7870901502857934  and for train data auc
score is -->  0.8952825382970504
10  for CV data auc score is -->  0.7871801010135627  and for train data auc
score is -->  0.8952837761769337
5  for CV data auc score is -->  0.7872845279997078  and for train data auc
score is -->  0.895284475811241
1  for CV data auc score is -->  0.7880851884397746  and for train data auc
score is -->  0.8952929048332272
0.5  for CV data auc score is -->  0.7890308647336308  and for train data au
c score is -->  0.8952979316675016
0.1  for CV data auc score is -->  0.7947878907104732  and for train data au
c score is -->  0.8951925459584849
0.05  for CV data auc score is -->  0.799390160119706  and for train data au
c score is -->  0.894898861221292
0.01  for CV data auc score is -->  0.810913285237761  and for train data au
c score is -->  0.8922461675909852
0.005  for CV data auc score is -->  0.8135008126520725  and for train data
auc score is -->  0.8898176448904851
0.001  for CV data auc score is -->  0.8078814330746301  and for train data
auc score is -->  0.8797681551775869
0.0005  for CV data auc score is -->  0.7975616958878764  and for train data
auc score is -->  0.8701477757334262
0.0001  for CV data auc score is -->  0.7352490005085146  and for train data
auc score is -->  0.7981504891027073
5e-05  for CV data auc score is -->  0.6932975119535364  and for train data
auc score is -->  0.7427729973208621
1e-05  for CV data auc score is -->  0.6231669968408926  and for train data
auc score is -->  0.6472890535318211

 Best C Value 0.005 with highest roc_auc Score is 0.8135008126520725
```


Auc Score v/s λ
0.90
0.85
0.80
0.75
0.70
0.65
roc_auc
CV Data
Train Data
10^{-4}
10^{-2}
10^{0}
10^{2}
10^{4}
alpha values


Line Plot of ROC Curve on Train Data
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

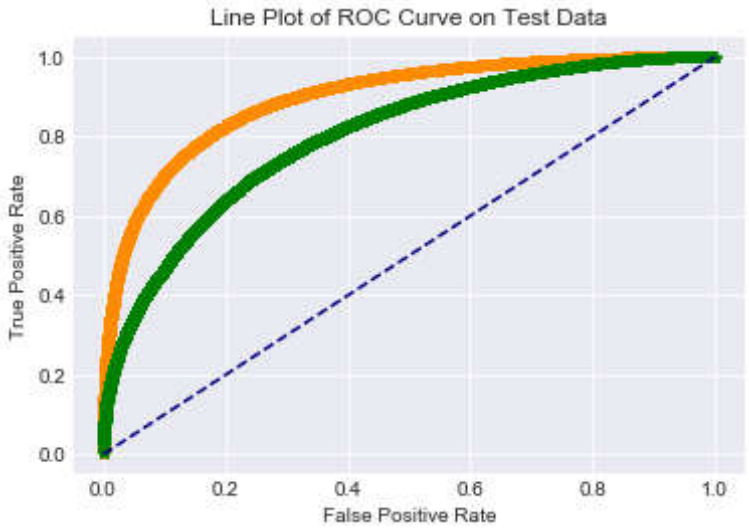
In [140]:

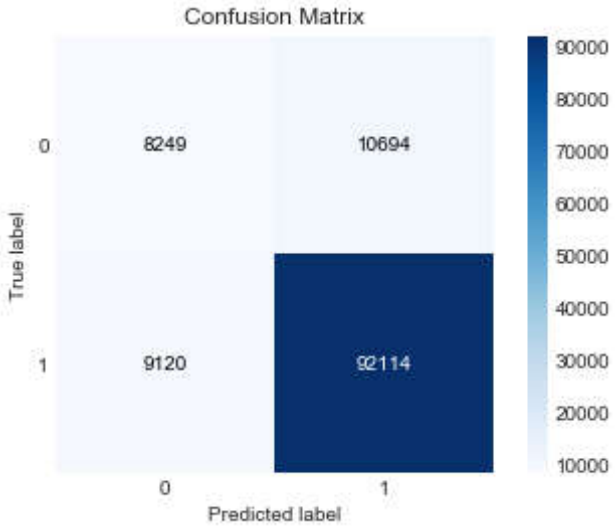
```
testing_l1(train_tfidfw2v,Y_tr,test_tfidfw2v,Y_test,optimal_C=0.1)
```

```
AUC Score 0.895310688455731
AUC Score 0.8018200100095967
macro f1 score for data : 0.6786163523815545
micro f1 scoore for data: 0.8351265217138055
hamming loss for data: 0.16487347828619453
Precision recall report for data:
              precision    recall  f1-score   support

          0       0.47      0.44      0.45     18943
          1       0.90      0.91      0.90    101234

avg / total       0.83      0.84      0.83    120177
```





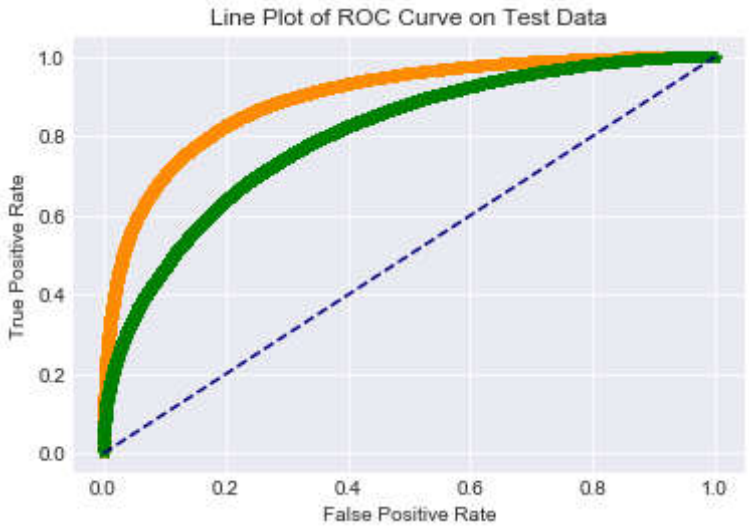
In [141]:

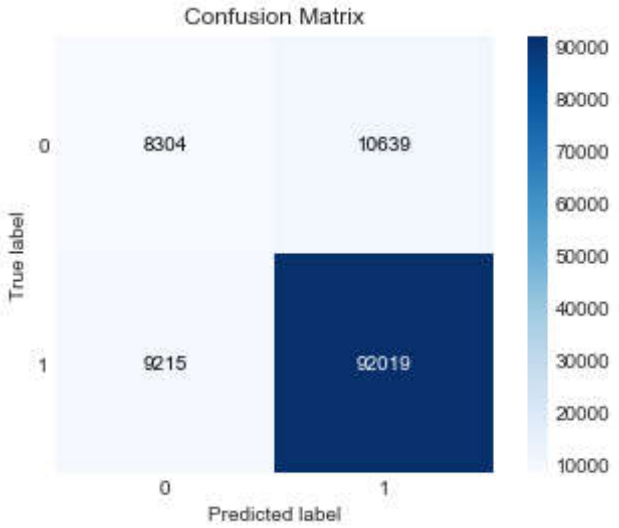
```
testing_l2(train_tfidfw2v,Y_tr,test_tfidfw2v,Y_test,optimal_C=0.5)
```

```
AUC Score 0.8952979316675016
AUC Score 0.8018407627368636
macro f1 score for data : 0.679056412145597
micro f1 scoore for data: 0.8347936793229985
hamming loss for data: 0.16520632067700142
Precision recall report for data:
              precision    recall  f1-score   support

           0       0.47      0.44      0.46     18943
           1       0.90      0.91      0.90    101234

avg / total       0.83      0.83      0.83    120177
```





In [143]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Regularization", "Best Hyper Parameter(C)","Test Auc Score"
x.add_row(["BoW","L1",5,95.62])
x.add_row(["BoW","L2",5,95.80])
x.add_row(["Tf-Idf","L1",5,96.88])
x.add_row(["Tf-Idf","L2",50,97.18])
x.add_row(["Avg-W2V","L1",0.1,86.74])
x.add_row(["Avg-W2V","L2",0.001,88.37])
x.add_row(["TfIdf-W2V","L1",0.005,80.52])
x.add_row(["TfIdf-W2V","L2",0.005,81.35])
from IPython.display import Markdown, display
def printmd(string):
    display(Markdown(string))
printmd('****Final Conclusion:****')
print(x)
```

<IPython.core.display.Markdown object>

```
+------------+----------------+-------------------------+----------------+
| Vectorizer | Regularization | Best Hyper Parameter(C) | Test Auc Score |
+------------+----------------+-------------------------+----------------+
|    BoW     |       L1       |            5            |     95.62      |
|    BoW     |       L2       |            5            |      95.8      |
|   Tf-Idf   |       L1       |            5            |     96.88      |
|   Tf-Idf   |       L2       |            50           |     97.18      |
|  Avg-W2V   |       L1       |           0.1           |     86.74      |
|  Avg-W2V   |       L2       |          0.001          |     88.37      |
| TfIdf-W2V  |       L1       |          0.005          |     80.52      |
| TfIdf-W2V  |       L2       |          0.005          |     81.35      |
+------------+----------------+-------------------------+----------------+
```

[6] Conclusions

In [0]:

```
TFIDF Featurization performs best with L2 Regularization and TEST AUC of 0.935.
Sparsity increases as we increase lambda or decrease C when L1 Regularizer is used.
```