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 [238]:

```
%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 [334]:

```
# using SQLite Table to read data.
con = sqlite3.connect('C:\\Users\\sujpanda\\Desktop\\applied\\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 a
# 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 !=
# for tsne assignment you can take 5k data points

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

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3
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 (100000, 10)
```

Out[334]:

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

In [335]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [336]:
```
print(display.shape)
display.head()
```

(80668, 7)

Out[336]:

	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 [337]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[337]:

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

In [338]:
```
display['COUNT(*)'].sum()
```

Out[338]: 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 [339]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[339]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomir
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 [340]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, ir
```

In [341]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text
final.shape
```

Out[341]:
```
(87775, 10)
```

In [342]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[342]:
```
87.775
```

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 [343]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[343]:

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

In [344]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [345]:
```
#Before starting the next phase of preprocessing lets see the number of entri
print(final.shape)

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

```
(87773, 10)
```

Out[345]:
```
1    73592
0    14181
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 [346]:

```
# 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying i
t anymore.  Its very hard to find any chicken products made in the USA but they
are out there, but this one isnt.  Its too bad too because its a good product b
ut I wont take any chances till they know what is going on with the china impor
ts.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy h
as little taste to it.  Very little of the 2 lbs that I bought were eaten and I
threw the rest away.  I would not buy the candy again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So i
f you are afraid of the fishy smell, don't get it. But I think my dog likes it
because of the smell. These treats are really small in size. They are great for
training. You can give your dog several of these without worrying about him ove
r eating. Amazon's price was much more reasonable than any other retailer. You
can buy a 1 pound bag on Amazon for almost the same price as a 6 ounce bag at o
ther retailers. It's definitely worth it to buy a big bag if your dog eats them
a lot.
==================================================
```

In [347]:

```
# 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying i
t anymore.  Its very hard to find any chicken products made in the USA but they
are out there, but this one isnt.  Its too bad too because its a good product b
ut I wont take any chances till they know what is going on with the china impor
ts.
```

In [348]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-re
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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying i
t anymore.  Its very hard to find any chicken products made in the USA but they
are out there, but this one isnt.  Its too bad too because its a good product b
ut I wont take any chances till they know what is going on with the china impor
ts.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy h
as little taste to it.  Very little of the 2 lbs that I bought were eaten and I
threw the rest away.  I would not buy the candy again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So i
f you are afraid of the fishy smell, don't get it. But I think my dog likes it
because of the smell. These treats are really small in size. They are great for
training. You can give your dog several of these without worrying about him ove
r eating. Amazon's price was much more reasonable than any other retailer. You
can buy a 1 pound bag on Amazon for almost the same price as a 6 ounce bag at o
ther retailers. It's definitely worth it to buy a big bag if your dog eats them
a lot.
```

In [349]:

```
# 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 [350]:

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

```
was way to hot for my blood, took a bite and did a jig  lol
==================================================
```

In [351]:

```
#remove words with numbers python: https://stackoverflow.com/a/18082370/40840
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying i
t anymore.  Its very hard to find any chicken products made in the USA but they
are out there, but this one isnt.  Its too bad too because its a good product b
ut I wont take any chances till they know what is going on with the china impor
ts.
```

In [352]:

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

```
was way to hot for my blood took a bite and did a jig lol
```

In [353]:
```
# 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 1s

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours',
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'i
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', '
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'beca
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', '
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'th
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "shoul
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn',
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", '
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shou
            'won', "won't", 'wouldn', "wouldn't"])
```

In [354]:
```
# 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
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 87773/87773 [00:50<00:00, 1738.45it/s]
```

In [355]:
```
preprocessed_reviews[1500]
```

Out[355]:
```
'way hot blood took bite jig lol'
```

[3.2] Preprocessing Review Summary

In [356]:

```
## Similartly you can do preprocessing for review summary also.
## Similartly you can do preprocessing for review summary also.
# Combining all the above stundents
from tqdm import tqdm
preprocessed_summary = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Summary'].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
    preprocessed_summary.append(sentance.strip())
```

```
100%|██████████| 87773/87773 [00:37<00:00, 2339.59it/s]
```

[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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) 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.
   - 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

   NOTE: Do sparsity and multicollinearity for any one of the vectorizers. Bow or tf-idf is recommended.

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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/)](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points. Please visualize your confusion matrices using [seaborn heatmaps. (https://seaborn.pydata.org/generated/seaborn.heatmap.html)](https://seaborn.pydata.org/generated/seaborn.heatmap.html)

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 (http://zetcode.com/python/prettytable/)](http://zetcode.com/python/prettytable/)

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)](https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf)

Some utility functions

In [357]:

```
def logistic_results(c,input_penalty,X_train,X_test,y_train,y_test):
    # roc curve and auc
    from sklearn.metrics import roc_curve
    from sklearn.metrics import roc_auc_score
    from matplotlib import pyplot
    # ============================== Logistic regression====================
    clf = LogisticRegression(C=c,penalty=input_penalty)

    # fitting the model
    clf.fit(X_train, y_train)

    # predict the response
    pred = clf.predict(X_test)

    # evaluate accuracy
    acc = accuracy_score(y_test, pred) * 100
    print('\nThe accuracy of the Logistic Regession classifier for c = %d is

    probs = clf.predict_proba(X_test)
    probs = probs[:, 1]
    # calculate AUC
    auc = roc_auc_score(y_test, probs)
    print('AUC: %.3f' % auc)
    # calculate roc curve
    fpr, tpr, thresholds = roc_curve(y_test, probs)
    # plot no skill
    pyplot.plot([0, 1], [0, 1], linestyle='--')
    # plot the roc curve for the model
    pyplot.plot(fpr, tpr, marker='.')
    # show the plot
    pyplot.show()
    from sklearn.metrics import confusion_matrix
    con_mat = confusion_matrix(y_test, pred, [0, 1])
    return con_mat,clf
```

In [358]:

```
def showHeatMap(con_mat):
    class_label = ["negative", "positive"]
    df_cm = pd.DataFrame(con_mat, index = class_label, columns = class_label)
    sns.heatmap(df_cm, annot = True, fmt = "d")
    plt.title("Confusion Matrix")
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()
```

In [359]:

```
## Some utility functions

def check_trade_off(X_train,X_test,y_train,y_test):

    from sklearn.metrics import roc_curve
    from sklearn.metrics import roc_auc_score

    [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]

    C_range1 = ['0.00001','0.001','1','100','10000',]
    C_range = [0.00001,0.001,1,100,10000]
    dummy_range = [1,2,3,4,5]

    auc_scores =[]
    auc_train_scores = []

    i = 0
    for i in C_range:
        clf =clf = LogisticRegression(C=i)

        # fitting the model on crossvalidation train
        clf.fit(X_train, y_train)


        #evaluate AUC score.
        probs = clf.predict_proba(X_test)
        probs = probs[:, 1]
        # calculate AUC
        auc = roc_auc_score(y_test, probs)
        print('AUC: %.3f' % auc)
        auc_scores.append(auc)

    print('#######################################################')
    print('AUC from train data ###################################')
    i = 0
    for i in C_range:
        clf = LogisticRegression(C=i)

        # fitting the model on crossvalidation train
        clf.fit(X_train, y_train)

        #evaluate AUC score.
        probs = clf.predict_proba(X_train)
        probs = probs[:, 1]
        # calculate AUC
        auc = roc_auc_score(y_train, probs)
        print('AUC: %.3f' % auc)
        auc_train_scores.append(auc)

    plt.plot(dummy_range, auc_scores,'r')
    plt.plot(dummy_range, auc_train_scores,'b')
    plt.xticks(dummy_range, C_range1, rotation='vertical')
    for xy in zip(dummy_range, auc_scores):
        plt.annotate('(%f, %f)' % xy, xy=xy, textcoords='data')
    for xy in zip(dummy_range, auc_train_scores):
        plt.annotate('(%f, %f)' % xy, xy=xy, textcoords='data')
```

```
    plt.xlabel('C-Values')
    plt.ylabel('auc_scores')
    plt.show()
```

Applying Logistic Regression

[5.1] Logistic Regression on BOW, SET 1

In [360]:

```
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import cross_validation
from sklearn.grid_search import GridSearchCV
import warnings
warnings.filterwarnings("ignore")
```

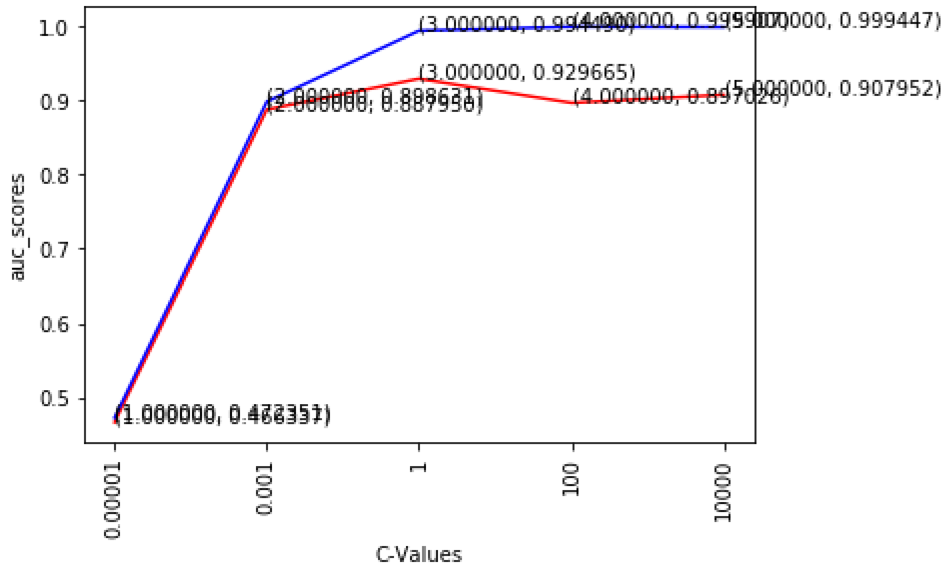
In [361]:

```
X_1, X_test, y_1, y_test = cross_validation.train_test_split(preprocessed_rev
```

In [362]:

```
count_vect = CountVectorizer()
final_counts = count_vect.fit_transform(X_1)
final_test_count = count_vect.transform(X_test)

# split the train data set into cross validation train and cross validation t
X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(X_1, y_1, test_siz

final_counts_tr_cv = count_vect.transform(X_tr)
final_test_count_cv = count_vect.transform(X_cv)


tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]

#Using GridSearchCV
model = GridSearchCV(LogisticRegression(), tuned_parameters, scoring = 'roc_a
model.fit(final_counts_tr_cv, y_tr)

print(model.best_estimator_)
print(model.score(final_test_count_cv, y_cv))

check_trade_off(final_counts_tr_cv,final_test_count_cv,y_tr,y_cv)
```

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
0.9296648957262242
AUC: 0.466
AUC: 0.888
AUC: 0.930
AUC: 0.897
AUC: 0.908
#######################################################
AUC from train data ###################################
AUC: 0.472
AUC: 0.899
AUC: 0.994
AUC: 1.000
AUC: 0.999
```

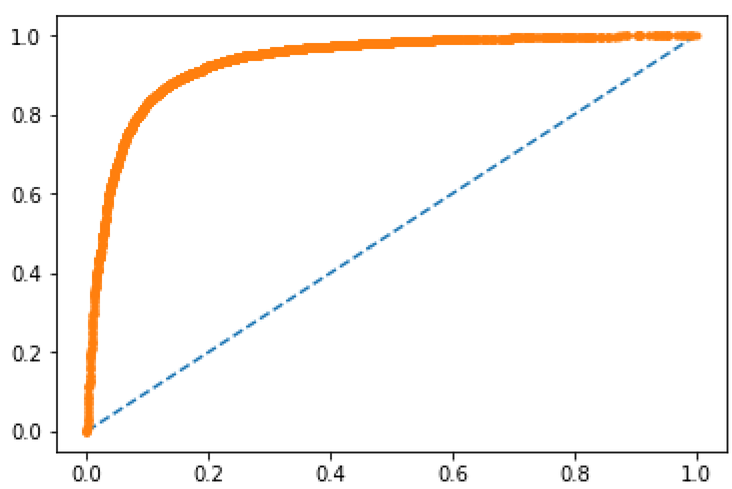


Observation: Optimal C is 1

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

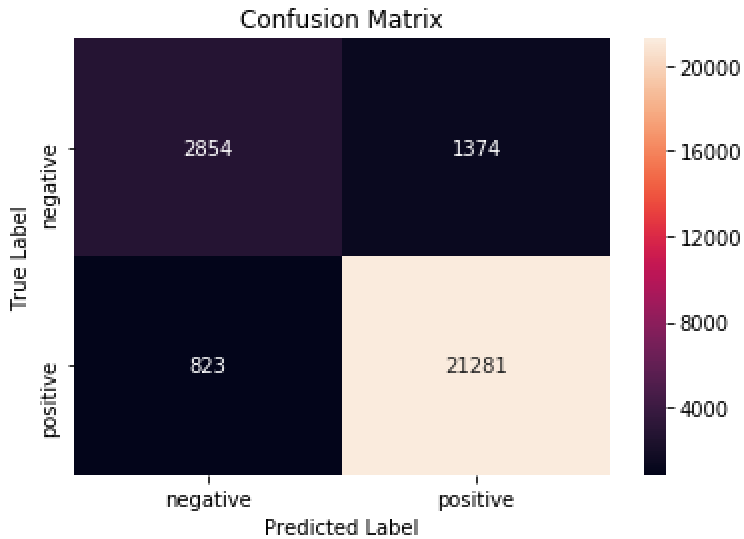
In [363]:
```
con_mat,clf = logistic_results(1,'l1',final_counts,final_test_count,y_1,y_tes
```

```
The accuracy of the Logistic Regession classifier for c = 1 is 91.656540%
AUC: 0.935
```



Observation: AUC = 0.935 is very good for this model

In [364]:
```
showHeatMap(con_mat)
```



Observation: My model predicte 823 + 1374 points wrongly.

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

In [365]:
```
w = clf.coef_
print(np.count_nonzero(w))
```

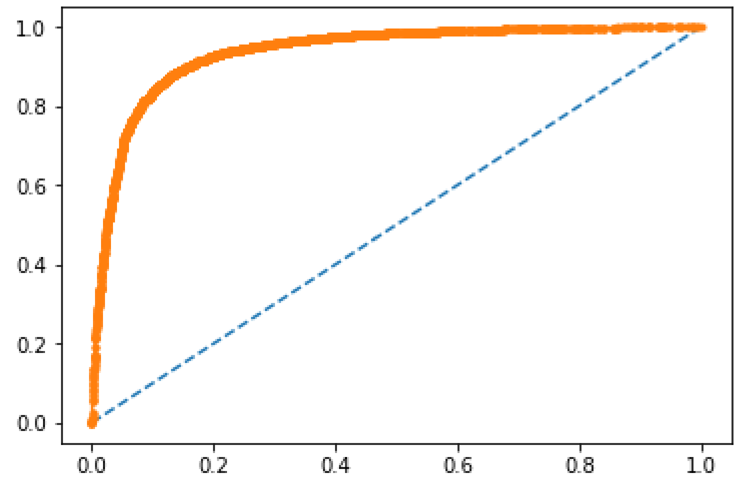
```
4655
```

Observation: 4655 features are non zero mean they are important for the model building.

[5.1.2] Applying Logistic Regression with L2 regularization on BOW, SET 1

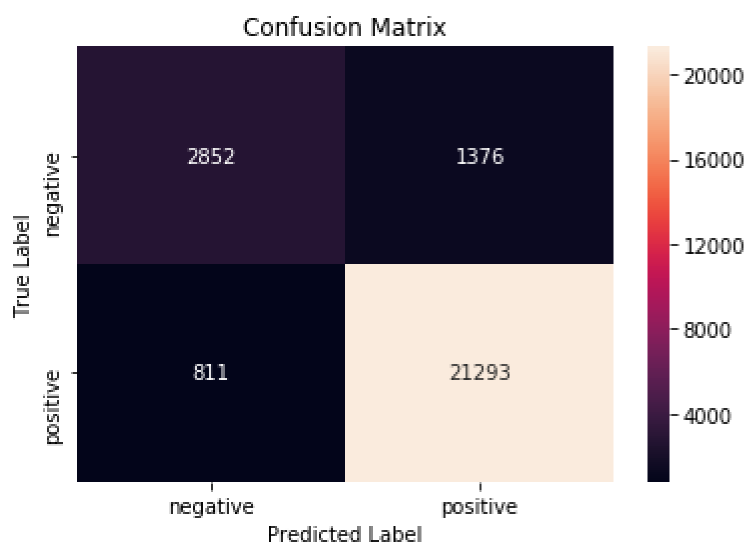
In [366]:
```
con_mat,clf = logistic_results(1,'l2',final_counts,final_test_count,y_1,y_tes
```

```
The accuracy of the Logistic Regession classifier for c = 1 is 91.694516%
AUC: 0.937
```



Observation: With AUC 0.937 both the models perform almost similarly with L1 and L2 regularizer.

In [367]:
```
showHeatMap(con_mat)
```



Observation: My model pedicted 811 + 1376 points wrongly.

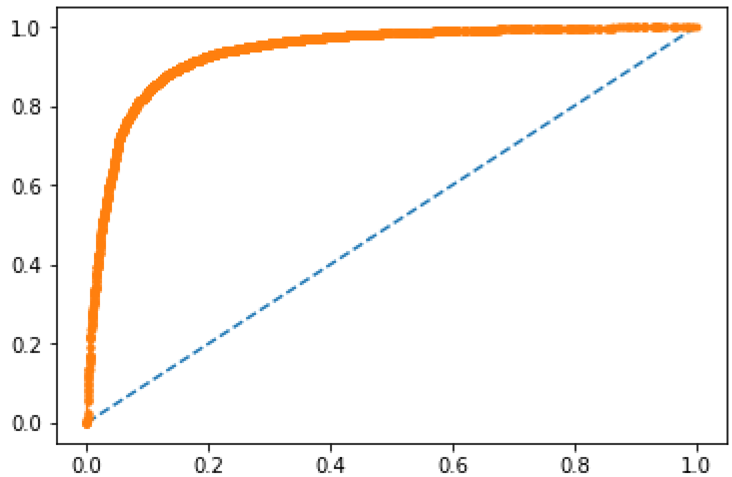
[5.1.2.1] Performing pertubation test (multicollinearity check) on BOW, SET 1

In [368]:
```
w = clf.coef_
```

In [369]:
```
e=np.random.normal(0,0.01)
final_counts.data = final_counts.data + e
```

In [370]:
```
con_mat,clf = logistic_results(1,'l2',final_counts,final_test_count,y_1,y_tes
```

```
The accuracy of the Logistic Regession classifier for c = 1 is 91.694516%
AUC: 0.937
```



In [371]:
```
w_dash = clf.coef_
```

In [372]:
```
# to avoid divide by zero
w = w + 10**-6
```

In [373]:
```
w_dash = w_dash + 10**-6
```

In [374]:
```
percentageChange = abs((w - w_dash) / w)*100
percentageChange.shape
```

Out[374]:
```
(1, 46446)
```

In [375]:
```
percentile_list = [0,10,20,30,40,50,60,70,80,90,100]

perentile_output = []

for i in percentile_list:
    p = np.percentile(percentageChange, i)
    perentile_output.append(p)
print(perentile_output)

```

```
[4.977845774948689e-05, 0.04276306216484363, 0.08809617114810528, 0.13754680775
741213, 0.19302171768105322, 0.25848840352156394, 0.34889138289043586, 0.486780
2664309642, 0.707673942817454, 1.1898960005129449, 4034.459062208871]
```

Observation: There is stiff change in percentage from 90th percentile to 100th percentile

In [376]:
```
p99 = np.percentile(percentageChange, 99)

print(p99)
```

```
13.614716538774443
```

In [377]:
```
p100 = np.percentile(percentageChange, 100)
print(p100)
```

```
4034.459062208871
```

In [378]:
```
percentile99_list = [99.1,99.2,99.3,99.4,99.5,99.6,99.7,99.8,99.9]
perentile99_output = []

for i in percentile99_list:
    p = np.percentile(percentageChange, i)
    perentile99_output.append(p)

```

In [379]:
```
print(perentile99_output)
```

```
[15.346042571173982, 16.549074088493587, 17.999667496954586, 21.65854652790269
3, 28.298062371468376, 33.503149013539904, 58.488352866694086, 160.272686847987
3, 600.1414102057831]
```

In [380]:
```
feature_names= count_vect.get_feature_names()
```

In [381]:
```
dictionary = dict(zip(feature_names, percentageChange[0]))
len(dictionary)
dict1_cond = {k:v for (k,v) in dictionary.items() if (v>578.8582227781285).ar
dict1_cond
```

Out[381]:
```
{'admiral': 600.1414102057831,
 'aft': 600.1414102057831,
 'algernon': 600.1414102057831,
 'angered': 600.1414102057831,
 'bellowed': 600.1414102057831,
 'brocoli': 623.0547158458185,
 'buccaneers': 600.1414102057831,
 'cacophony': 600.1414102057831,
 'callous': 600.1414102057831,
 'caverdish': 600.1414102057831,
 'chuckled': 600.1414102057831,
 'clink': 600.1414102057831,
 'commandeered': 600.1414102057831,
 'controlled': 1984.46573487145,
 'cowardice': 600.1414102057831,
 'decreed': 600.1414102057831,
 'devious': 600.1414102057831,
 'digress': 1308.3580885728638,
 'dutchies': 600.1414102057831,
 'eendracht': 600.1414102057831,
 'fiddych': 600.1414102057831,
 'giggled': 600.1414102057831,
 'governor': 600.1414102057831,
 'grappling': 600.1414102057831,
 'gruff': 600.1414102057831,
 'gruffed': 600.1414102057831,
 'householder': 600.1414102057831,
 'kneeled': 600.1414102057831,
 'knelt': 600.1414102057831,
 'leeward': 600.1414102057831,
 'lieutenant': 600.1414102057831,
 'lqqking': 633.1303416178392,
 'maarten': 600.1414102057831,
 'marque': 600.1414102057831,
 'midshipman': 600.1414102057831,
 'mitford': 600.1414102057831,
 'muskets': 600.1414102057831,
 'nancies': 600.1414102057831,
 'olivier': 600.1414102057831,
 'onesies': 600.1414102057831,
 'patrolled': 600.1414102057831,
 'pillars': 600.1414102057831,
 'portsmouth': 600.1414102057831,
 'privateering': 600.1414102057831,
 'privateers': 600.1414102057831,
 'requisitioned': 600.1414102057831,
 'restraints': 600.1414102057831,
 'resupply': 600.1414102057831,
 'riches': 600.1414102057831,
 'rotterdam': 600.1414102057831,
 'sailed': 600.1414102057831,
 'sanctioned': 600.1414102057831,
```

```
 'seeded': 4034.459062208871,
 'siege': 600.1414102057831,
 'sleepwalk': 600.1414102057831,
 'sullen': 600.1414102057831,
 'swashbuckling': 600.1414102057831,
 'tactician': 600.1414102057831,
 'treachery': 600.1414102057831,
 'tromp': 600.1414102057831,
 'twas': 600.1414102057831,
 'unshaven': 600.1414102057831,
 'unsheathed': 600.1414102057831,
 'volleyed': 600.1414102057831,
 'whirrrrr': 600.1414102057831,
 'windward': 600.1414102057831,
 'yorkie': 1579.8715415096058}
```

Observation: These are the feature more than threshold 578.8582227781285

[5.1.3] Feature Importance on BOW, SET 1

In [382]:
```
feature_names = np.array(count_vect.get_feature_names())
sorted_coef_index = clf.coef_[0].argsort()
```

[5.1.3.1] Top 10 important features of positive class from SET 1

In [383]:
```
print('Top 10 positive features: \n{}\n'.format(feature_names[sorted_coef_ind
```

```
Top 10 positive features:
['pleasantly' 'hooked' 'worried' 'complaint' 'resist' 'satisfied' 'beat'
 'excellent' 'wonderful' 'delicious']
```

[5.1.3.2] Top 10 important features of negative class from SET 1

In [384]:
```
print('Top 10 negative features: \n{}\n'.format(feature_names[sorted_coef_ind
```

```
Top 10 negative features:
['worst' 'disappointing' 'undrinkable' 'cancelled' 'rip' 'terrible'
 'disappointment' 'hopes' 'canceled' 'tasteless']
```

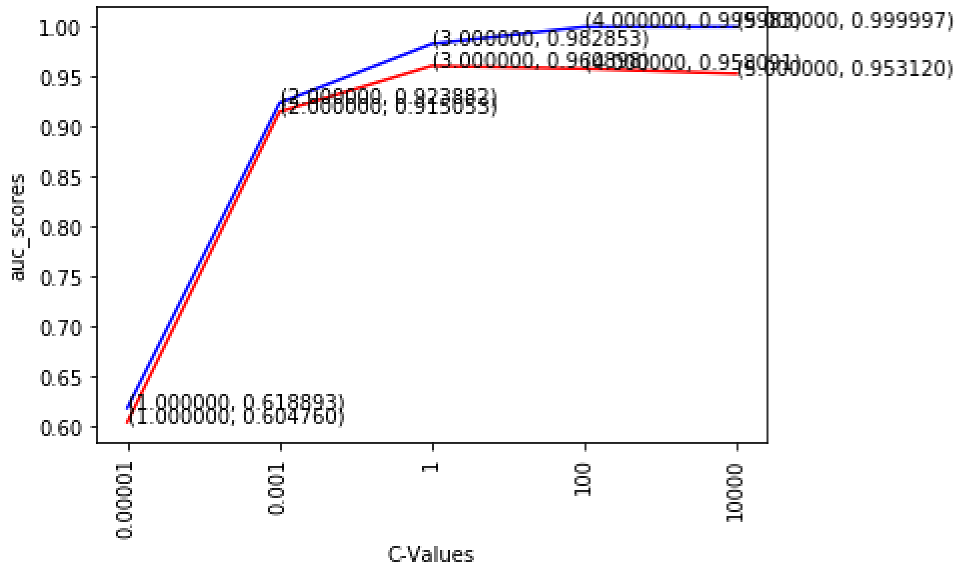
[5.2] Logistic Regression on TFIDF, SET 2

In [385]:
```
X_1, X_test, y_1, y_test = cross_validation.train_test_split(preprocessed_rev
```

In [386]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(X_1)
final_tf_idf = tf_idf_vect.transform(X_1)
final_test_count = tf_idf_vect.transform(X_test)

# split the train data set into cross validation train and cross validation t
X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(X_1, y_1, test_siz

final_counts_tr_cv = tf_idf_vect.transform(X_tr)
final_test_count_cv = tf_idf_vect.transform(X_cv)

tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]

#Using GridSearchCV
model = GridSearchCV(LogisticRegression(), tuned_parameters, scoring = 'roc_a
model.fit(final_counts_tr_cv, y_tr)

print(model.best_estimator_)
print(model.score(final_test_count_cv, y_cv))

check_trade_off(final_counts_tr_cv,final_test_count_cv,y_tr,y_cv)
```

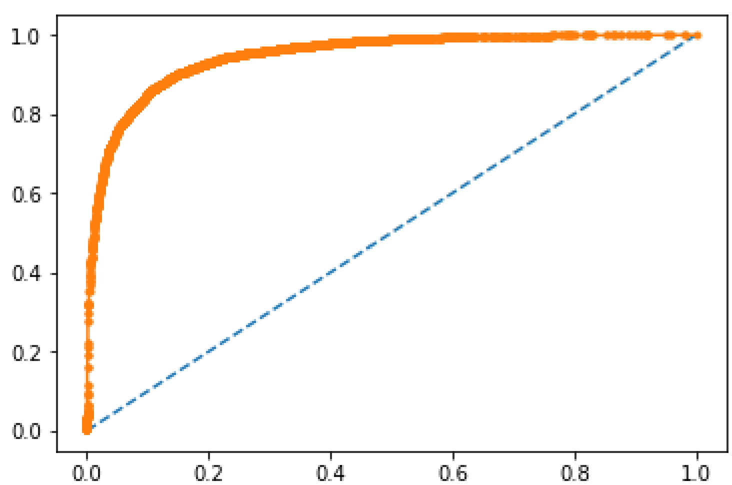
```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
0.9608978149597611
AUC: 0.605
AUC: 0.915
AUC: 0.961
AUC: 0.958
AUC: 0.953
#######################################################
AUC from train data ###################################
AUC: 0.619
AUC: 0.924
AUC: 0.983
AUC: 1.000
AUC: 1.000
```



[5.2.1] Applying Logistic Regression with L1 regularization on TFIDF, SET 2

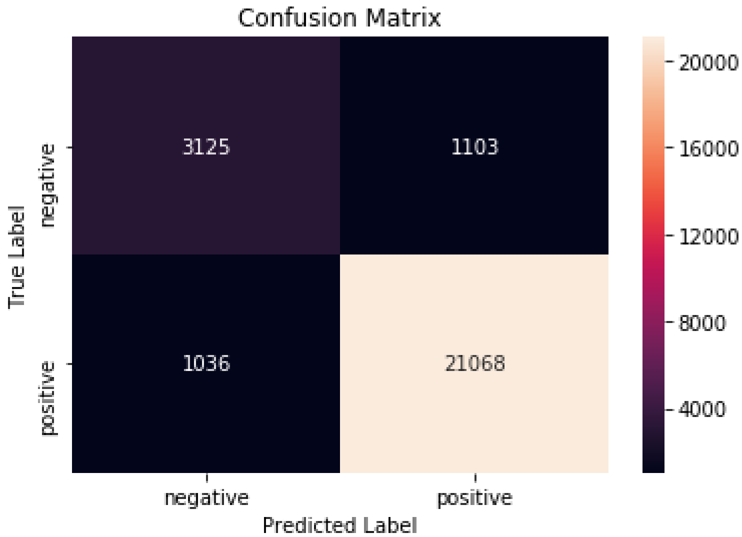
In [387]:
```
con_mat,clf = logistic_results(100,'l1',final_tf_idf,final_test_count,y_1,y_t
```

```
The accuracy of the Logistic Regession classifier for c = 100 is 91.876804%
AUC: 0.948
```



Observation: Auc value 0.948 is a very good prediction.

In [388]:
```
1 showHeatMap(con_mat)
```

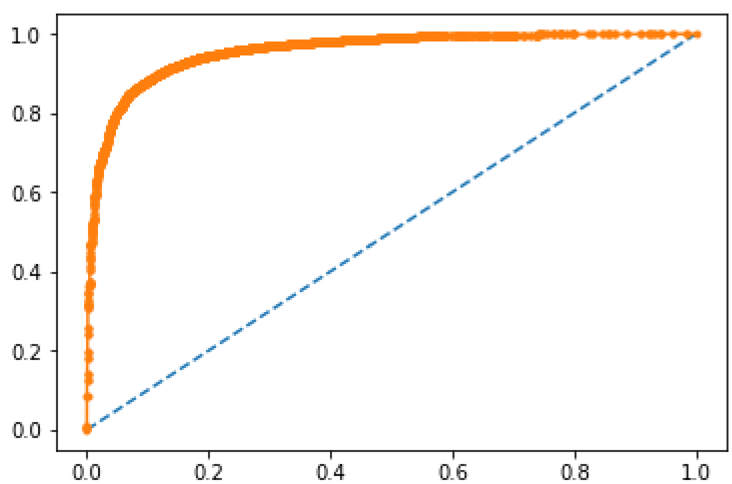


Observation: My model predicted 1036 + 1103 points wrongly.

[5.2.2] Applying Logistic Regression with L2 regularization on TFIDF, SET 2

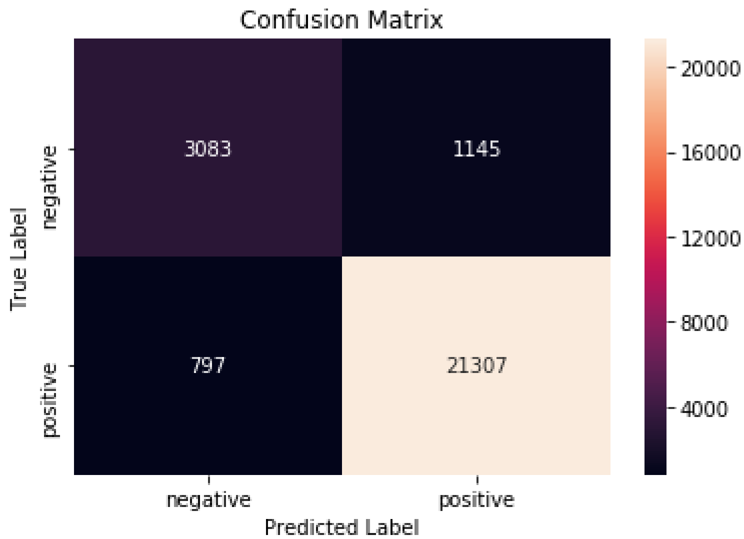
In [390]:
```
1 con_mat,clf = logistic_results(100,'l2',final_tf_idf,final_test_count,y_1,y_t
```

```
The accuracy of the Logistic Regession classifier for c = 100 is 92.624943%
AUC: 0.957
```



Observation: L2 regularizer has neglible impact over L1 regularizer.

In [391]:
```
showHeatMap(con_mat)
```



Observation: My model predicted 797 + 1145 points wrongly.

[5.2.3] Feature Importance on TFIDF, SET 2

In [392]:
```
feature_names = np.array(tf_idf_vect.get_feature_names())
sorted_coef_index = clf.coef_[0].argsort()
```

[5.2.3.1] Top 10 important features of positive class from SET 2

In [393]:
```
print('Top 10 positive features: \n{}\n'.format(feature_names[sorted_coef_ind
```

```
Top 10 positive features:
['great' 'not disappointed' 'delicious' 'best' 'good' 'perfect'
 'wonderful' 'yummy' 'nice' 'complaint']
```

[5.2.3.2] Top 10 important features of negative class from SET 2

In [394]:
```
print('Top 10 negative features: \n{}\n'.format(feature_names[sorted_coef_ind
```

```
Top 10 negative features:
['worst' 'not recommend' 'not worth' 'disappointing' 'disappointed'
 'terrible' 'not good' 'disappointment' 'not impressed' 'two stars']
```

[5.3] Logistic Regression on AVG W2V, SET 3

In [395]:
```
X_train, X_test, y_1, y_test = cross_validation.train_test_split(preprocessed
```

In [396]:

```
i=0
list_of_sentance=[]
for sentance in X_train:
    list_of_sentance.append(sentance.split())

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

# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this l
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you migh
    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.append(sent_vec)
print(sent_vectors[0])



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

test_sent_vectors = [];

for sent in tqdm(list_of_test_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you migh
    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
    test_sent_vectors.append(sent_vec)
print(test_sent_vectors[0])

```

```
100%|██████████| 61441/61441 [03:44<00:00, 274.17it/s]

[ 0.74045644  0.60303211  0.12341589 -0.45794731  0.32432638  0.11831686
 -0.12156923 -0.52800487 -0.92545938  0.01349999 -0.55728007 -0.33659766
  0.53924885 -0.30885776  0.46957641  0.42749843  0.12095882 -0.00679701
 -0.1448173  -0.68314885  0.418051   -0.35123651  0.6618213  -0.70897275
 -0.74302584  0.52285688  0.69474762 -0.24266879  1.07457848  0.53164221
  0.5222707  -0.29674824  0.34685098  0.99548314 -0.10032255  0.30168726
  0.48121558  0.35064653  0.0025968   0.25674252  0.24919124  0.09882337
```

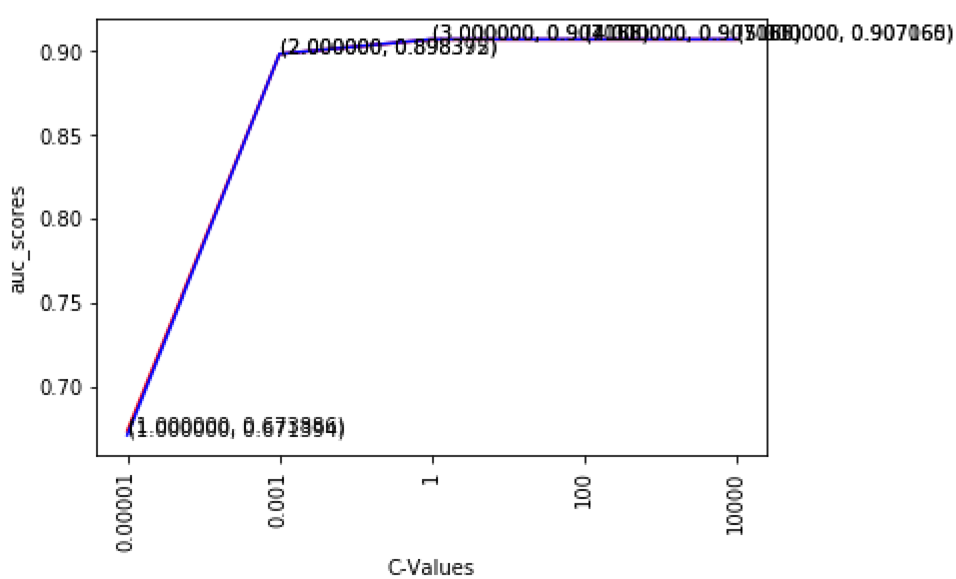
```
  0.81942444  0.38134465 -0.90479087 -0.21129748  0.30605643 -0.00329594
  0.36646091  0.69998252]

100%|██████████| 26332/26332 [01:27<00:00, 301.14it/s]

[ 0.98208966 -0.00353566  1.0263749  -1.25980652 -0.03089801 -0.13802817
 -0.67412848  0.76422948 -0.68406282 -0.01215545 -0.3506007   0.0543738
  0.38830186  0.0422269   1.08450957  0.92130454  0.98330136  0.59649227
  0.14995765 -0.52200646 -0.42841324 -0.48182835 -0.69027534  0.37838847
  0.27869663  0.27796    -0.25707956  0.14310502 -0.18219045 -0.48930866
  1.08867794  0.1090177  -0.23210589 -0.24454332 -0.16841162 -1.11224413
  0.19355719  0.43011621 -0.21351859 -0.38531899 -0.8526959   0.03386442
  0.37350017  0.02895028  0.04147604  0.48510124 -0.07853465 -0.26035498
 -0.47163333  0.11295018]
```

In [399]:

```
# split the train data set into cross validation train and cross validation t
X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(X_train, y_1, test

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

cv_train_sent_vectors = [];

for sent in tqdm(list_of_cv_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you migh
    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
    cv_train_sent_vectors.append(sent_vec)
print(cv_train_sent_vectors[0])

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

cv_test_sent_vectors = [];

for sent in tqdm(list_of_cv_test_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you migh
    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
    cv_test_sent_vectors.append(sent_vec)
print(cv_test_sent_vectors[0])

tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]

#Using GridSearchCV
model = GridSearchCV(LogisticRegression(), tuned_parameters, scoring = 'roc_a
model.fit(cv_train_sent_vectors, y_tr)

print(model.best_estimator_)
print(model.score(cv_test_sent_vectors, y_cv))

check_trade_off(cv_train_sent_vectors,cv_test_sent_vectors,y_tr,y_cv)
```

```
100%|██████████| 43008/43008 [02:27<00:00, 291.14it/s]
```

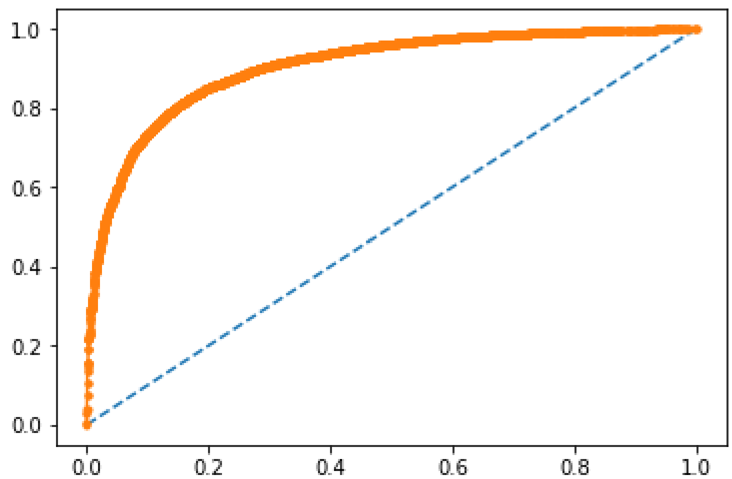
```
[ 0.37361057  0.31626734  0.54422138 -0.38241403  0.40141402 -0.02619578
 -0.35063599 -0.98078779 -0.99473731  0.09431426 -0.24607888 -0.02253204
  0.07713559 -0.17152592 -0.06503122  0.56621434 -0.05171528  0.35966107
 -0.54602919 -0.5155081   0.93328671 -0.10365933  0.19818859  0.04914908
 -0.36811468  0.50337239 -0.34957303  0.18935331  0.6428792   0.48135398
  0.43446666  0.13928255  0.08969751 -0.84231885 -0.04207609  0.80123565
  0.44986954  0.44091471 -0.38318165  0.38986817 -1.14287725 -0.35971191
  0.74201355  0.10782631  0.05853157 -0.16336035 -0.09909254  0.41024937
 -0.10741118 -0.27210346]

100%|██████████| 18433/18433 [01:13<00:00, 251.76it/s]

[ 0.63816541  0.3986908   0.46477161 -0.57556484 -0.06937366  0.13951594
 -0.42554119 -0.51502702 -0.87926801 -0.22107712 -0.15586036 -0.07928071
 -0.16015537 -0.63207406  0.68287121  0.81767482  0.1516807   0.27963698
 -0.70797576 -0.0283674   0.75852527 -0.46727257  0.51651693 -0.40529962
 -0.41706849  0.16574075  0.9508073   0.35727982  0.5304499   0.16506671
  0.50069449 -0.39542366  0.57839521  0.04590331 -0.1277163  -0.09088829
  0.61001984  0.41065946 -0.00129317  0.35397229  0.03127426 -0.18227117
  0.85830291  0.32050966 -0.81701327  0.38167497  0.51377736  0.21926436
  0.03697683  0.4978388 ]
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
0.9070581687867151
AUC: 0.674
AUC: 0.898
AUC: 0.907
AUC: 0.907
AUC: 0.907
#######################################################
AUC from train data ###################################
AUC: 0.671
AUC: 0.898
AUC: 0.907
AUC: 0.907
AUC: 0.907
```



[5.3.1] Applying Logistic Regression with L1 regularization on AVG W2V SET 3

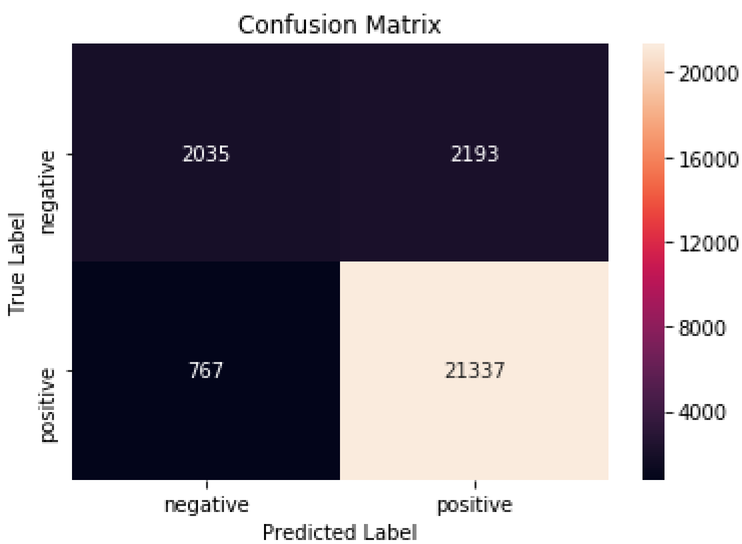
In [401]:
```
con_mat,clf = logistic_results(1,'l1',sent_vectors,test_sent_vectors,y_1,y_te
```

```
The accuracy of the Logistic Regession classifier for c = 1 is 88.758925%
AUC: 0.905
```



Observation: My model predicted 88% accurately wity AUC 0.905

In [402]:
```
showHeatMap(con_mat)
```

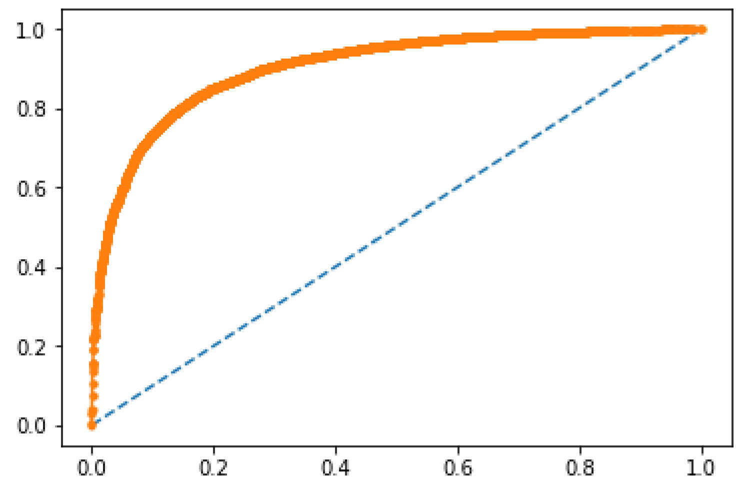


Observation: My model predicted 767 + 2193 points wrongly.

[5.3.2] Applying Logistic Regression with L2 regularization on AVG W2V, SET 3

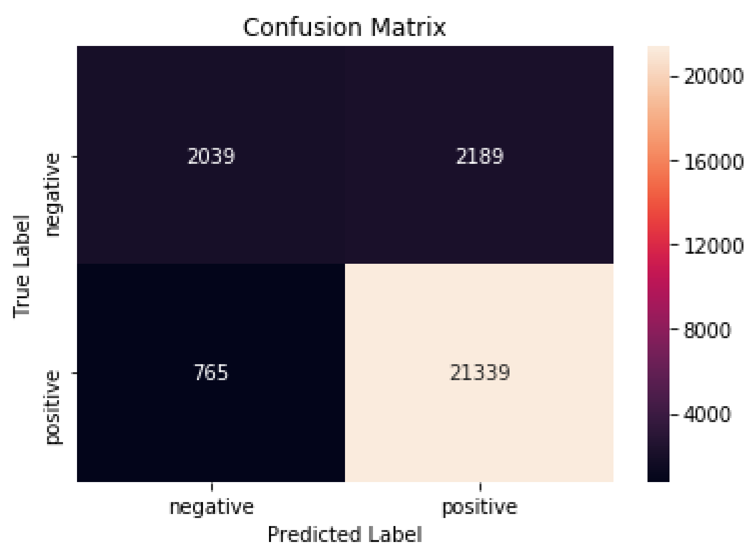
In [426]:
```
1 con_mat,clf = logistic_results(1,'l2',sent_vectors,test_sent_vectors,y_1,y_te
```

```
The accuracy of the Logistic Regession classifier for c = 1 is 88.770317%
AUC: 0.905
```



Observation: L2 regularizer has neglible impact over L1 regularizer.

In [404]:
```
1 showHeatMap(con_mat)
```



Observation: My model predicted 765 + 2189 points wrongly

[5.4] Logistic Regression on TFIDF W2V, SET 4

In [405]:
```
1 X_train, X_test, y_1, y_test = cross_validation.train_test_split(preprocessed
```

In [406]:

```
model = TfidfVectorizer()
X_train_transformed = model.fit_transform(X_train)

dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [407]:

```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentance in X_train:
    list_of_sentance.append(sentance.split())
```

In [408]:

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

In [409]:

```
# 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_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored i
row=0;
for sent in tqdm(list_of_sentance): # 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.append(sent_vec)
    row += 1
```

```
100%|██████████| 61441/61441 [53:27<00:00, 19.16it/s]
```

In [410]:

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

In [411]:

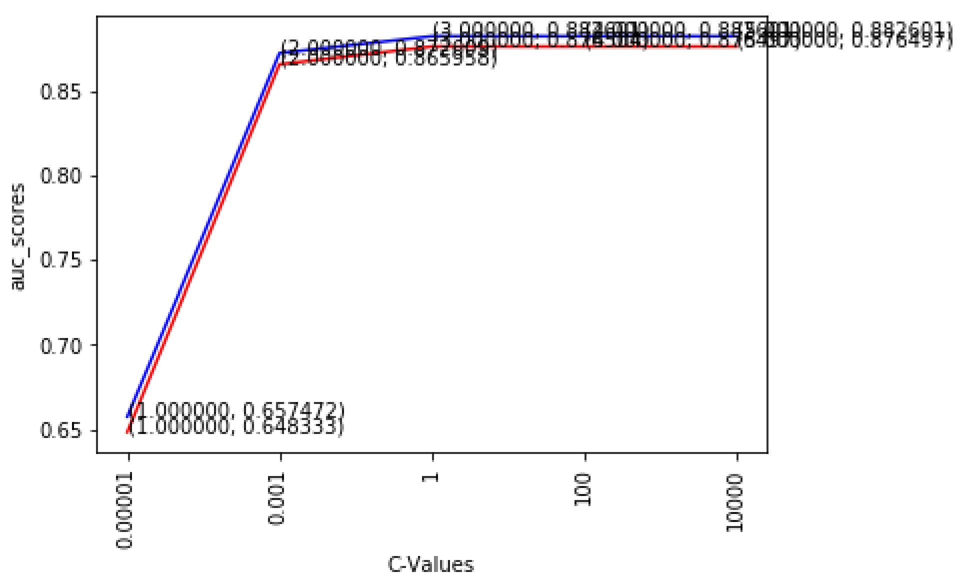
```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val

tfidf_test_sent_vectors = []; # the tfidf-w2v for each sentence/review is sto
row=0;
for sent in tqdm(list_of_test_sentance): # 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_test_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 26332/26332 [21:12<00:00, 20.69it/s]
```

In [412]:
```
X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(X_train, y_1, test

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


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_cv_sent_vectors = []; # the tfidf-w2v for each sentence/review is store
row=0;
for sent in tqdm(list_of_cv_sentance): # 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_cv_sent_vectors.append(sent_vec)
    row += 1

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


tfidf_cv_test_sent_vectors = []; # the tfidf-w2v for each sentence/review is
row=0;
for sent in tqdm(list_of_cv_test_sentance): # 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_cv_test_sent_vectors.append(sent_vec)
    row += 1
```

```


#Using GridSearchCV
model = GridSearchCV(LogisticRegression(), tuned_parameters, scoring = 'roc_a
model.fit(tfidf_cv_sent_vectors, y_tr)

print(model.best_estimator_)
print(model.score(tfidf_cv_test_sent_vectors, y_cv))

check_trade_off(tfidf_cv_sent_vectors,tfidf_cv_test_sent_vectors,y_tr,y_cv)

```

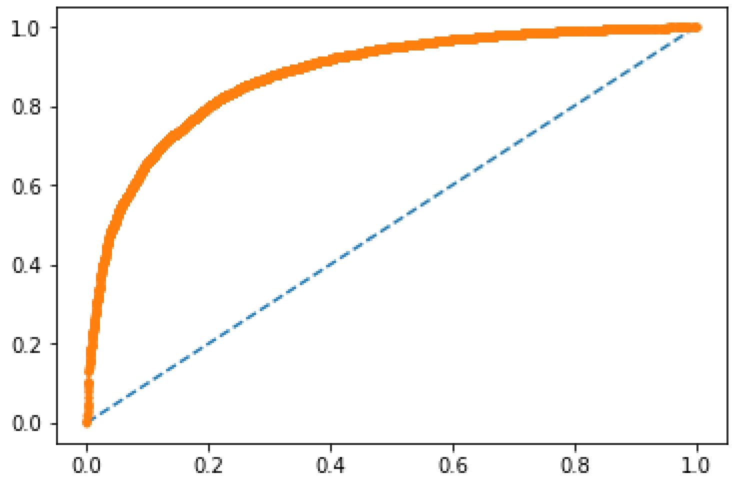
```
100%|██████████| 43008/43008 [34:27<00:00, 20.80it/s]
100%|██████████| 18433/18433 [17:13<00:00, 17.83it/s]

LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
0.8765040082504795
AUC: 0.648
AUC: 0.866
AUC: 0.877
AUC: 0.876
AUC: 0.876
#######################################################
AUC from train data ###################################
AUC: 0.657
AUC: 0.873
AUC: 0.883
AUC: 0.883
AUC: 0.883
```



[5.4.1] Applying Logistic Regression with L1 regularization on TFIDF W2V, SET 4

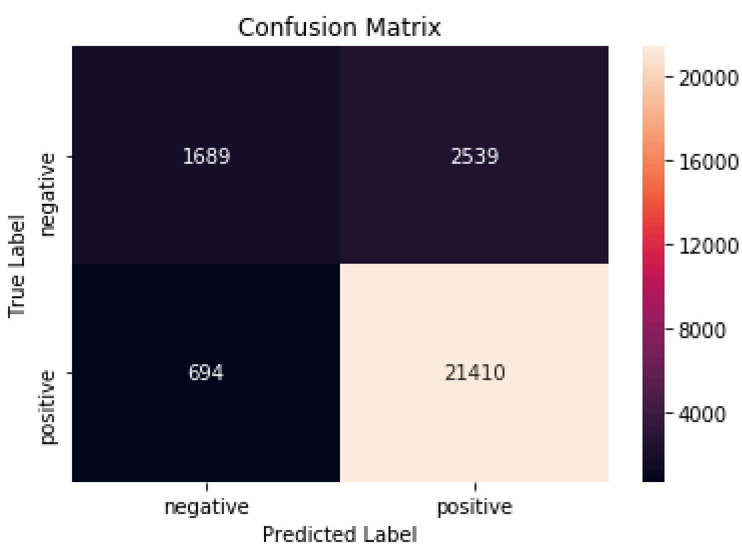
In [427]:
```
1 con_mat,clf = logistic_results(1,'l1',tfidf_sent_vectors,tfidf_test_sent_vect
```

```
The accuracy of the Logistic Regession classifier for c = 1 is 87.710770%
AUC: 0.879
```



Observation: My model predicted with accuracy 87 % with AUC score : 0.879

In [414]:
```
1 showHeatMap(con_mat)
```

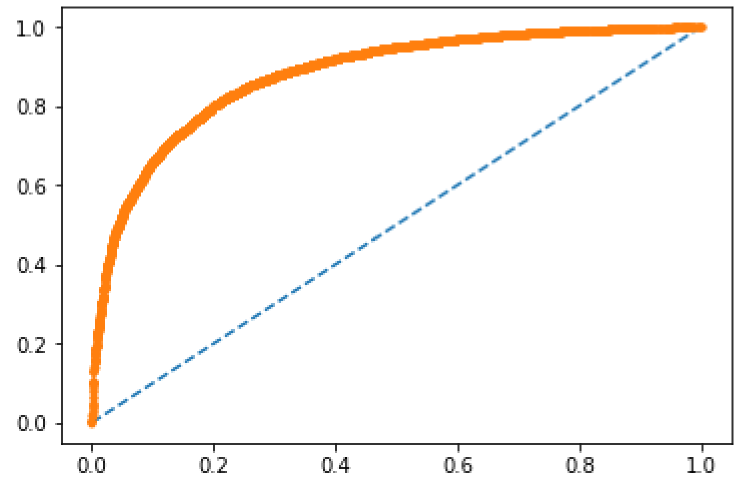


Observation: My model predicted 694 + 2539 points wrongly

[5.4.2] Applying Logistic Regression with L2 regularization on TFIDF W2V, SET 4

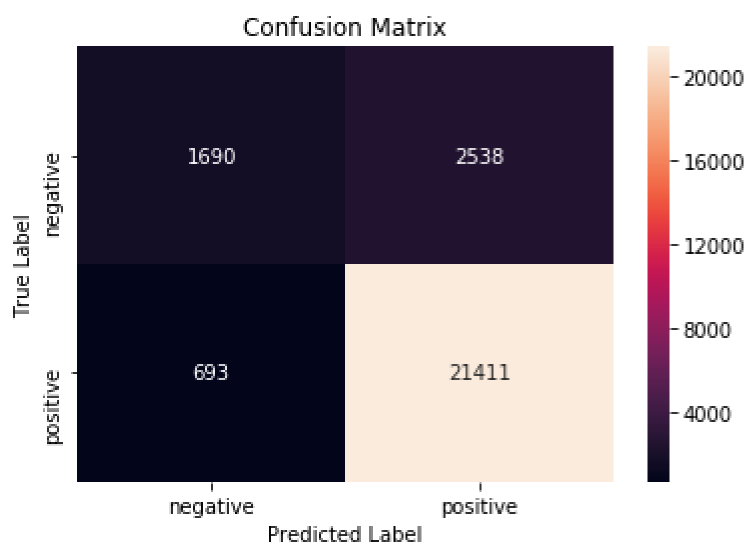
In [428]:
```
con_mat,clf = logistic_results(1,'l2',tfidf_sent_vectors,tfidf_test_sent_vect
```

```
The accuracy of the Logistic Regession classifier for c = 1 is 87.718365%
AUC: 0.879
```



Observation: L2 regularizer has no impact over L1 regularizer.

In [416]:
```
showHeatMap(con_mat)
```



Observation: My model predicted 693 + 2538 points wrongly.

Repeat with extra features

In [417]:
```
mylen = np.vectorize(len)
newarr = mylen(preprocessed_summary)
```

In [418]:
```
newproce_reviews = np.asarray(preprocessed_reviews)
```

In [419]:
```
newproce_summary = np.asanyarray(preprocessed_summary)
```

In [420]:
```
df = pd.DataFrame({'desc':newproce_reviews, 'summary':newproce_summary,'len':
```

In [421]:
```
df.head()
```

Out[421]:

	desc	summary	len
0	dogs loves chicken product china wont buying a...	made china	10
1	dogs love saw pet store tag attached regarding...	dog lover delites	17
2	infestation fruitflies literally everywhere fl...	one fruitfly stuck	18
3	worst product gotten long time would rate no s...	not work not waste money	24
4	wish would read reviews making purchase basica...	big rip	7

In [422]:
```
X_1, X_test, y_1, y_test = cross_validation.train_test_split(df, final['Score
```

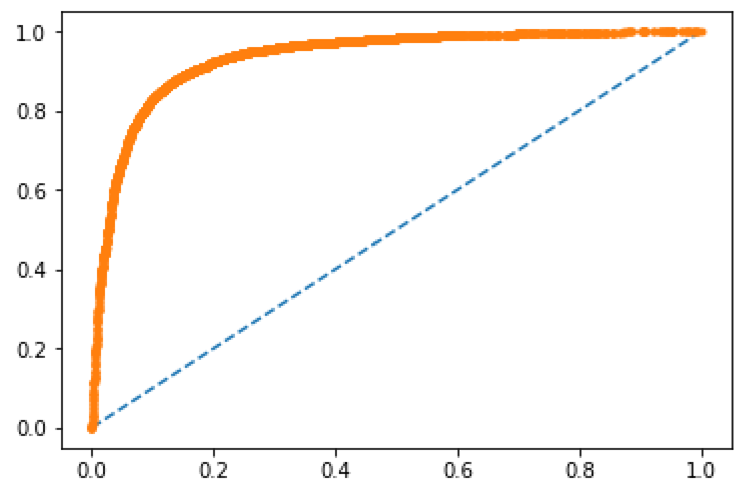
In [423]:
```
import scipy
count_vect = CountVectorizer()
final_counts = count_vect.fit_transform(X_1['desc'])
final_test_count = count_vect.transform(X_test['desc'])

# split the train data set into cross validation train and cross validation t
X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(X_1, y_1, test_siz

final_counts_tr_cv = count_vect.transform(X_tr['desc'])
final_test_count_cv = count_vect.transform(X_cv['desc'])

from scipy.sparse import csr_matrix, issparse

#################Adding len as feature#########################
#if issparse(final_counts_tr_cv):
    #print('sparse matrix')
len_sparse = scipy.sparse.coo_matrix(X_tr['len'])
len_sparse = len_sparse.transpose()

final_counts_tr_cv = scipy.sparse.hstack([final_counts_tr_cv, len_sparse])
print(final_counts_tr_cv.shape)

len_test_sparse = scipy.sparse.coo_matrix(X_cv['len'])
len_test_sparse = len_test_sparse.transpose()
final_test_count_cv = scipy.sparse.hstack([final_test_count_cv,len_test_spars
print("final_counts_tr_cv.shape after length = ",final_counts_tr_cv.shape)

#################Adding summary as feature#########################
final_summary_count = count_vect.transform(X_tr['summary'])
final_test_summary_count_cv = count_vect.transform(X_cv['summary'])
columns=count_vect.get_feature_names()

print("sujet",final_summary_count[:,12].shape)
final_counts_tr_cv = scipy.sparse.hstack([final_counts_tr_cv, final_summary_c
print("final_counts_tr_cv.shape after f1= ",final_counts_tr_cv.shape)

final_test_count_cv = scipy.sparse.hstack([final_test_count_cv,final_test_sum


final_counts_tr_cv = scipy.sparse.hstack([final_counts_tr_cv, final_summary_c
print("final_counts_tr_cv.shape after f2= ",final_counts_tr_cv.shape)


final_test_count_cv = scipy.sparse.hstack([final_test_count_cv,final_test_sum

#############finding the new C ########################################

tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]

#Using GridSearchCV
model = GridSearchCV(LogisticRegression(), tuned_parameters, scoring = 'roc_a
model.fit(final_counts_tr_cv, y_tr)

print(model.best_estimator_)
print(model.score(final_test_count_cv, y_cv))

```

```
(43008, 46447)
final_counts_tr_cv.shape after length =  (43008, 46447)
sujet (43008, 1)
final_counts_tr_cv.shape after f1=  (43008, 46448)
final_counts_tr_cv.shape after f2=  (43008, 46449)
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
0.9315652418505183
```

In [424]:

```
con_mat,clf = logistic_results(1,'l1',final_counts,final_test_count,y_1,y_tes
```

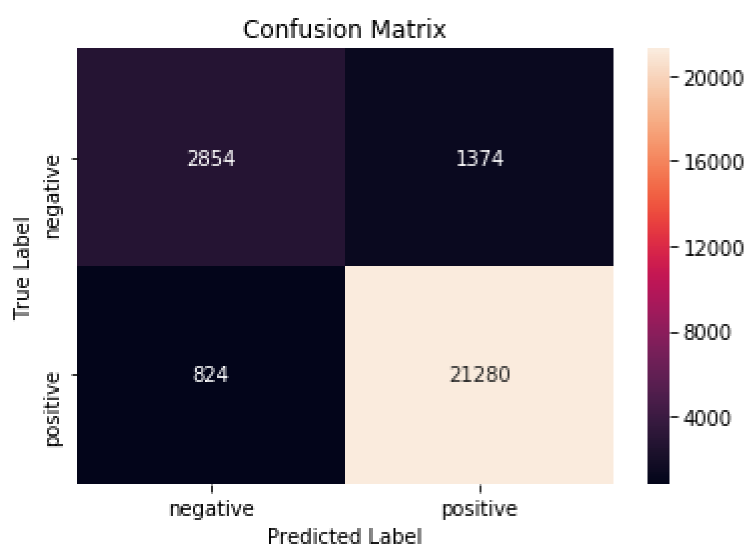
```
The accuracy of the Logistic Regession classifier for c = 1 is 91.652742%
AUC: 0.935
```



Observation: My model predicted with accuracy 91% with AUC: 0.935

In [425]:

```
showHeatMap(con_mat)
```



Observation : My model predicted 824 + 1374 points wrongly.

[6] Conclusions

Method	No of samples	C	accuray	AUC Score	Regulaizer
BOW	100000	1	91	0.935	l1
BOW	100000	1	91	0.937	l2
TFIDF	100000	1	91	0.948	l1
TFIDF	100000	1	91	0.957	l2
AVG W2VE	100000	1	88	0.905	l1
AVG W2VE	100000	1	88	0.905	l2
TFIDF W2VE	100000	1	87	0.879	l1
TFIDF W2VE	100000	1	87	0.879	l2
BOW1	100000	1	91	0.935	l1

In [ ]:

```
1
```