Apply KNN on Amazon Fine Food Reviews

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

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

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

Attribute Information:

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

Objective:

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

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

[Ans] We could use 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.

*** I am applying brute KNN on 50k points because of low system configration(4GB RAM and 1.3GHz processor) and automatic system shut down issue due to heat up.**

*** I am applying kd-tree KNN on 20k points due to above issue and kd-tree takes more amount of time and memory as compair to brute KNN.**

----->>> Impertant note-->> I have applied K-NN on 50k on brute and 20k on kd-tree seperately, from 50k points that i filtered data earlier i have sample out 20k points and apply kd-tree on it.

[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 [1]:

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


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os
```

```
C:\Users\HIMANSHU NEGI\Anaconda3\lib\site-packages\gensim\utils.py:1212: Use
rWarning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [158]:

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

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 50000"""

# 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 (50000, 10)
```

Out[158]:

	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 [159]:

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

In [160]:

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

(80668, 7)

Out[160]:

	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 [161]:

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

Out[161]:

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

In [162]:

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

Out[162]:

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 [163]:

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

Out[163]:

	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 [164]:

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

In [165]:

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

Out[165]:

```
(46072, 10)
```

In [166]:

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

Out[166]:

```
92.144
```

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 [167]:

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

display.head()
```

Out[167]:

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

In [168]:

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

In [169]:

```
#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()
```

```
(46071, 10)
```

Out[169]:

```
1    38479
0     7592
Name: Score, dtype: int64
```

note: From here you will see the code is written one after the another cell for Kd-tree version and brute version it is only for redability steps of code but 1st we have run all steps for brute version of KNN and then we did it for kd-T ree.

Here we have done sampling of points we have taken 7592 -ve and 12408 +ve points from 50k points for brute version.

In [274]:

```
data_pos =final[final["Score"] == 1].sample(n = 12408)
data_neg = final[final["Score"] == 0].sample(n = 7592)
final = pd.concat([data_pos, data_neg])
final.shape
```

Out[274]:

```
(20000, 10)
```

[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 [275]:

```
# 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 coffee has my favorite characteristics: medium roast and extra bold.  B
y the way, the ones I got are extra bold. I don't see that in the descriptio
n or on the label here. It has a bit of an aftertaste, not a bad one though.
==================================================
Great taste and good nutrional value.  Worth the money. Has a lot of necessa
ry minerals and doesn't hurt the body like refined salt.
==================================================
Refreshing orange juice drink with a good kick to it. If you drink this firs
t thing in the morning with breakfast, it's sure to be a real eye-opener! I
t's a "juice" drink, not soda pop. But it is carbonated, so it has the same
bubbly, zesty fizz. The tangerine taste is better than Switch's black cherr
y. I would definitely buy a whole case of this orange flavor. Highly recomme
nded!!
==================================================
We have always used "Bags on Board" for our pet clean up.  Unfortunately, th
is time, the container that hangs on the leash broke off and we had to purch
ase a new one.  I can't say how it happened because my husband was walking t
he dog.  However, I would not let that keep me from purchasing this product.
It is lightweight, easily dispensed and, having them on the leash, makes it
so simple to clean up after your dog.  Everyone should be required to carry
something as a courtesy to walkers.  You can use other types of plastic bags
from groceries or other purchases, but these are the most compact and easies
t to carry around and use.
==================================================
```

In [276]:

```
# 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 coffee has my favorite characteristics: medium roast and extra bold.  B
y the way, the ones I got are extra bold. I don't see that in the descriptio
n or on the label here. It has a bit of an aftertaste, not a bad one though.
```

In [278]:

```
# 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 coffee has my favorite characteristics: medium roast and extra bold.  B
y the way, the ones I got are extra bold. I don't see that in the descriptio
n or on the label here. It has a bit of an aftertaste, not a bad one though.
==================================================
Great taste and good nutrional value.  Worth the money. Has a lot of necessa
ry minerals and doesn't hurt the body like refined salt.
==================================================
Refreshing orange juice drink with a good kick to it. If you drink this firs
t thing in the morning with breakfast, it's sure to be a real eye-opener! I
t's a "juice" drink, not soda pop. But it is carbonated, so it has the same
bubbly, zesty fizz. The tangerine taste is better than Switch's black cherr
y. I would definitely buy a whole case of this orange flavor. Highly recomme
nded!!
==================================================
We have always used "Bags on Board" for our pet clean up.  Unfortunately, th
is time, the container that hangs on the leash broke off and we had to purch
ase a new one.  I can't say how it happened because my husband was walking t
he dog.  However, I would not let that keep me from purchasing this product.
It is lightweight, easily dispensed and, having them on the leash, makes it
so simple to clean up after your dog.  Everyone should be required to carry
something as a courtesy to walkers.  You can use other types of plastic bags
from groceries or other purchases, but these are the most compact and easies
t to carry around and use.
```

In [279]:

```
# 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 [280]:

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

```
Refreshing orange juice drink with a good kick to it. If you drink this firs
t thing in the morning with breakfast, it is sure to be a real eye-opener! I
t is a "juice" drink, not soda pop. But it is carbonated, so it has the same
bubbly, zesty fizz. The tangerine taste is better than Switch is black cherr
y. I would definitely buy a whole case of this orange flavor. Highly recomme
nded!!
==================================================
```

In [281]:

```
#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 coffee has my favorite characteristics: medium roast and extra bold.  B
y the way, the ones I got are extra bold. I don't see that in the descriptio
n or on the label here. It has a bit of an aftertaste, not a bad one though.
```

In [282]:

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

```
Refreshing orange juice drink with a good kick to it If you drink this first
thing in the morning with breakfast it is sure to be a real eye opener It is
a juice drink not soda pop But it is carbonated so it has the same bubbly ze
sty fizz The tangerine taste is better than Switch is black cherry I would d
efinitely buy a whole case of this orange flavor Highly recommended
```

In [177]:

```
# 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 [178]:

```
#code for BRUTE version
# 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%|██████████████████████████████████████████████████
███| 46071/46071 [00:30<00:00, 1532.32it/s]
```

In [283]:

```
#code for kd-TREE version
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%|████████████████████████████████████████████████████████████████
███| 20000/20000 [00:13<00:00, 1507.63it/s]
```

In [284]:

```
preprocessed_reviews[1500]
```

Out[284]:

```
'refreshing orange juice drink good kick drink first thing morning breakfast
sure real eye opener juice drink not soda pop carbonated bubbly zesty fizz t
angerine taste better switch black cherry would definitely buy whole case or
ange flavor highly recommended'
```

[5.1] Applying KNN brute force

In [179]:

```
#spilliting data only for brute-KNN
from sklearn.model_selection import train_test_split

# split the data set into train and test
X_1, X_test, y_1, y_test = train_test_split(preprocessed_reviews, final['Score'], test_size

# split the train data set into cross validation train and cross validation test
X_tr, X_cv, y_tr, y_cv = train_test_split(X_1, y_1, test_size=0.3)
```

In [285]:

```
# ============================== loading libraries ========================================
from sklearn.metrics import roc_curve,auc
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_validate
# ==========================================================================================
```

[5.1.1] Applying KNN brute force on BOW, SET 1

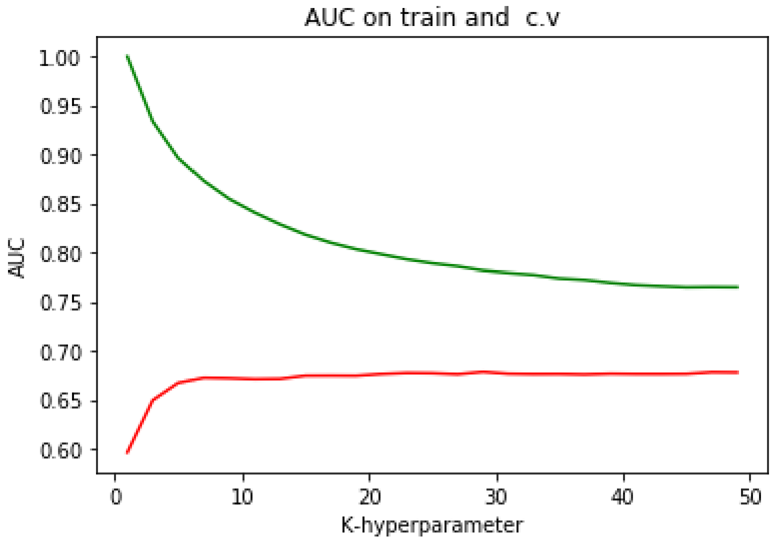
In [267]:

```
#code for BRUTE version
count_vect = CountVectorizer(min_df = 10)
Xbow_tr = count_vect.fit_transform(X_tr)
Xbow_test = count_vect.transform(X_test)
Xbow_cv = count_vect.transform(X_cv)
print("the type of count vectorizer :",type(X_tr))
print("the shape of out text BOW vectorizer : ",Xbow_tr.get_shape())
print("the number of unique words :", Xbow_tr.get_shape()[1])
```

```
the type of count vectorizer : <class 'list'>
the shape of out text BOW vectorizer :  (22574, 5755)
the number of unique words : 5755
```

In [182]:

```
myList = list(range(0,50))
neighbors = list(filter(lambda x: x % 2 != 0, myList))

auc1=[]
auc2=[]

for i in neighbors:
    # instantiate learning model (k = 50)
    knn = KNeighborsClassifier(n_neighbors=i,algorithm = 'brute')

    # fitting the model on crossvalidation train
    knn.fit(Xbow_tr, y_tr)

    probs = knn.predict_proba(Xbow_tr)
    preds = probs[:,1]
    roc_auc1=metrics.roc_auc_score(y_tr, preds)
    auc1.append(roc_auc1)

    #knn.fit(Xbow_cv, y_cv)
    probs = knn.predict_proba(Xbow_cv)
    preds = probs[:,1]
    roc_auc2=metrics.roc_auc_score(y_cv, preds)
    auc2.append(roc_auc2)

plt.title('AUC on train and  c.v')
plt.plot(neighbors, auc1,'g',label ='Train')
plt.plot(neighbors, auc2,'r',label ='C.V')

plt.ylabel('AUC')
plt.xlabel('K-hyperparameter')
plt.show()
```



Train curve with green and C.V curve with red color above

In [16]:

```
# ============================== KNN with k = optimal_k ===================================
# instantiate learning model k = optimal_k
knn_brute = KNeighborsClassifier(n_neighbors=9,algorithm = 'brute')

# fitting the model
knn_brute.fit(Xbow_tr, y_tr)

# predict the response
pred = knn_brute.predict(Xbow_test)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (9, 83.063233))
```

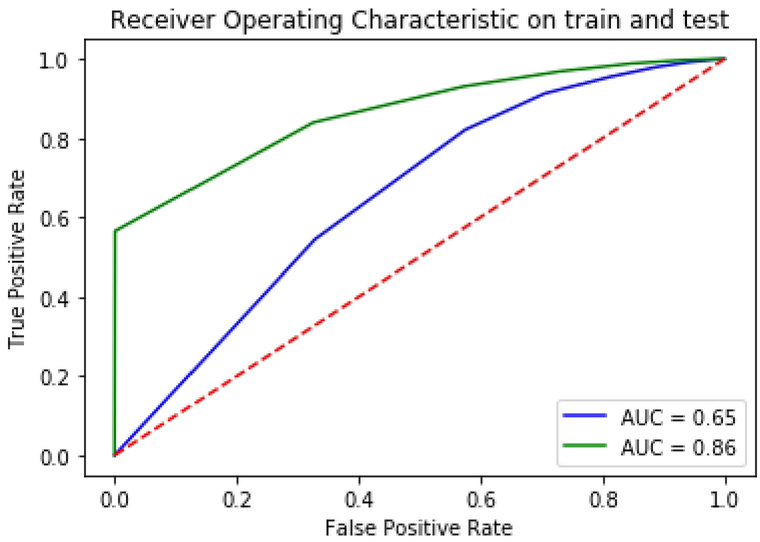
```
The accuracy of the knn classifier for k = 9 is 83.063233%
```

In [269]:

```
probs2 = knn_brute.predict_proba(Xbow_tr)
preds2 = probs2[:,1]
fpr2, tpr2, threshold2 = metrics.roc_curve(y_tr, preds2)
roc_auc2 = metrics.auc(fpr2, tpr2)


probs1 = knn_brute.predict_proba(Xbow_test)
preds1 = probs1[:,1]
fpr1, tpr1, threshold1 = metrics.roc_curve(y_test, preds1)
roc_auc1 = metrics.auc(fpr1, tpr1)
```

In [270]:

```
plt.title('Receiver Operating Characteristic on train and test')
plt.plot(fpr1, tpr1, 'b', label = 'AUC = %0.2f' % roc_auc1)
plt.plot(fpr2, tpr2, 'g', label = 'AUC = %0.2f' % roc_auc2)
#plt.plot(neighbors, auc1,'g')
#plt.plot(neighbors, auc2,'r')
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Train curve with green and test curve with blue color above

In [272]:

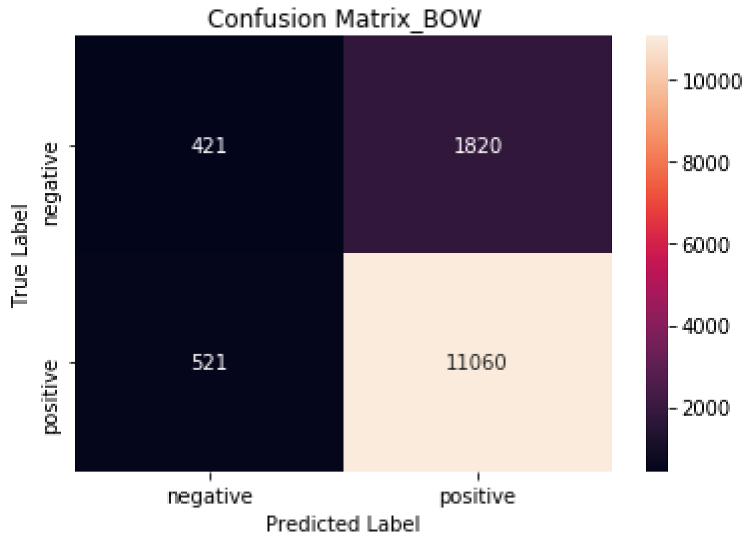
```
from sklearn.metrics import confusion_matrix
knn_brute.fit(Xbow_tr, y_tr)
cm = confusion_matrix(y_test, pred)
cm
```

Out[272]:

```
array([[  421,  1820],
       [  521, 11060]], dtype=int64)
```

In [273]:

```
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix_BOW")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")

plt.show()
```



[5.1.2] Applying KNN brute force on TFIDF, SET 2

In [230]:

```
#code for BRUTE version

count_vect  = TfidfVectorizer(ngram_range=(1,2), min_df=10)

Xbow_tr = count_vect.fit_transform(X_tr)
Xbow_test = count_vect.transform(X_test)
Xbow_cv = count_vect.transform(X_cv)
print("the type of count vectorizer :",type(X_tr))
print("the shape of out text BOW vectorizer : ",Xbow_tr.get_shape())
print("the number of unique words :", Xbow_tr.get_shape()[1])
```

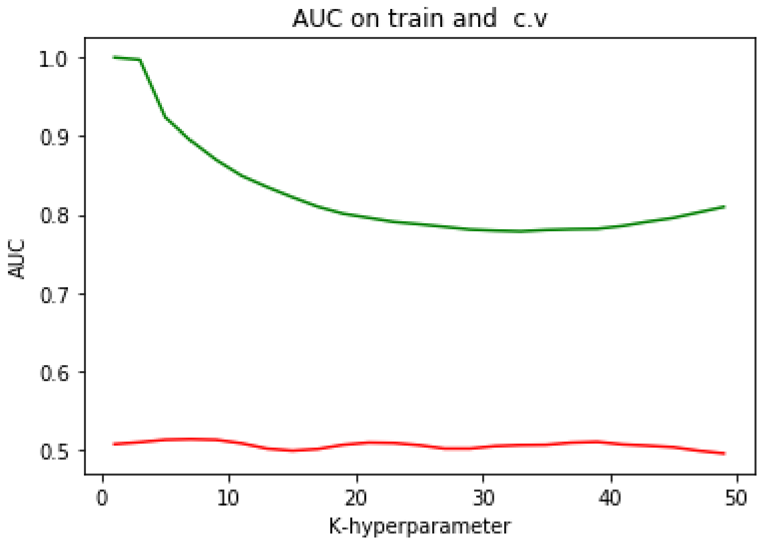
```
the type of count vectorizer : <class 'list'>
the shape of out text BOW vectorizer :  (22574, 13489)
the number of unique words : 13489
```

In [216]:

```
myList = list(range(0,50))
neighbors = list(filter(lambda x: x % 2 != 0, myList))

auc1=[]
auc2=[]

for i in neighbors:
    # instantiate learning model (k = 50)
    knn = KNeighborsClassifier(n_neighbors=i,algorithm = 'brute')

    # fitting the model on crossvalidation train
    knn.fit(Xbow_tr, y_tr)

    probs = knn.predict_proba(Xbow_tr)
    preds = probs[:,1]
    roc_auc1=metrics.roc_auc_score(y_tr, preds)
    auc1.append(roc_auc1)

    #knn.fit(Xbow_cv, y_cv)
    probs = knn.predict_proba(Xbow_cv)
    preds = probs[:,1]
    roc_auc2=metrics.roc_auc_score(y_cv, preds)
    auc2.append(roc_auc2)
```

In [217]:

```

plt.title('AUC on train and  c.v')
plt.plot(neighbors, auc1,'g',label = 'Train')
plt.plot(neighbors, auc2,'r',label ='C.V' )

plt.ylabel('AUC')
plt.xlabel('K-hyperparameter')
plt.show()
```



Train curve with green and cv curve with red color above

In [218]:

```
print(auc1)

print(auc2)
```

```
[0.9996020164499868, 0.9966919076819256, 0.9234669062022297, 0.8940851030838
417, 0.8695025025733306, 0.8493627501365234, 0.8350862622201901, 0.822102347
8383988, 0.8100335592935525, 0.8011179376614482, 0.795874043197529, 0.790640
413132533, 0.7876771764000093, 0.7843771827279584, 0.7809586017391349, 0.779
5216564355418, 0.7785707286573341, 0.7803582296388011, 0.7813037536166134,
0.781796599965475, 0.7854771100729449, 0.7908855312115681, 0.795633510588803
3, 0.802747063552996, 0.8097120261154528]
[0.5080060525843455, 0.5104343657947286, 0.5135509093638538, 0.5142752637129
401, 0.5134926032907902, 0.508986320996919, 0.5022536293089672, 0.4996570759
3598624, 0.5017339515365231, 0.5071053741094167, 0.5099250370573561, 0.50940
04386116329, 0.5066338486397775, 0.502317129426048, 0.5023055306961753, 0.50
55137315683685, 0.5067232018180561, 0.5070917446254922, 0.509858139332535,
0.5108628158466925, 0.5076040413880173, 0.5060107976755052, 0.50409751493502
44, 0.4995927166537297, 0.4961546109910969]
```

In [231]:

```
# ============================== KNN with k = optimal_k ==================================
# instantiate learning model k = optimal_k
knn_brute = KNeighborsClassifier(n_neighbors=5,algorithm = 'brute')

# fitting the model
knn_brute.fit(Xbow_tr, y_tr)

# predict the response
pred = knn_brute.predict(Xbow_test)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (5, acc))
```

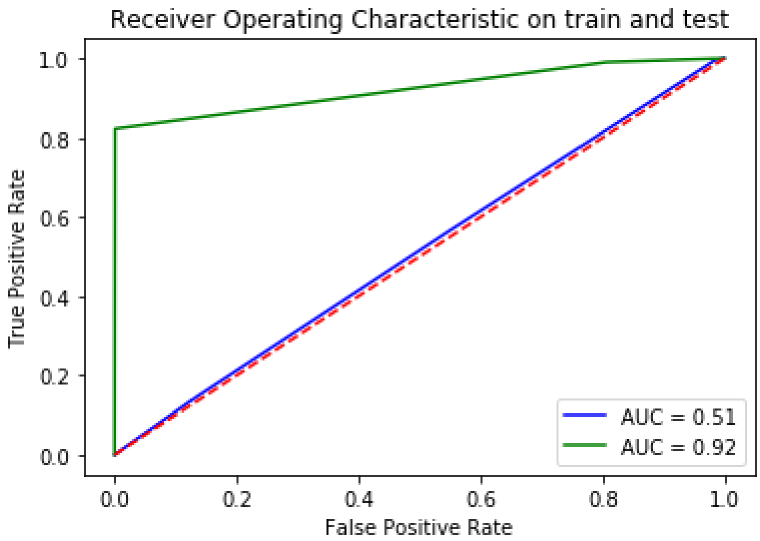
```
The accuracy of the knn classifier for k = 5 is 83.772247%
```

In [232]:

```
#knn.fit(Xbow_tr, y_tr)
probs2 = knn_brute.predict_proba(Xbow_tr)
preds2 = probs2[:,1]
fpr2, tpr2, threshold2 = metrics.roc_curve(y_tr, preds2)
roc_auc2 = metrics.auc(fpr2, tpr2)

#knn.fit(Xbow_test, y_test)
probs1 = knn_brute.predict_proba(Xbow_test)
preds1 = probs1[:,1]
fpr1, tpr1, threshold1 = metrics.roc_curve(y_test, preds1)
roc_auc1 = metrics.auc(fpr1, tpr1)
```

In [233]:

```
plt.title('Receiver Operating Characteristic on train and test')
plt.plot(fpr1, tpr1, 'b', label = 'AUC = %0.2f' % roc_auc1)
plt.plot(fpr2, tpr2, 'g', label = 'AUC = %0.2f' % roc_auc2)
#plt.plot(neighbors, auc1,'g')
#plt.plot(neighbors, auc2,'r')
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')

plt.show()
```



Train curve with green and test curve with blue color above

In [234]:

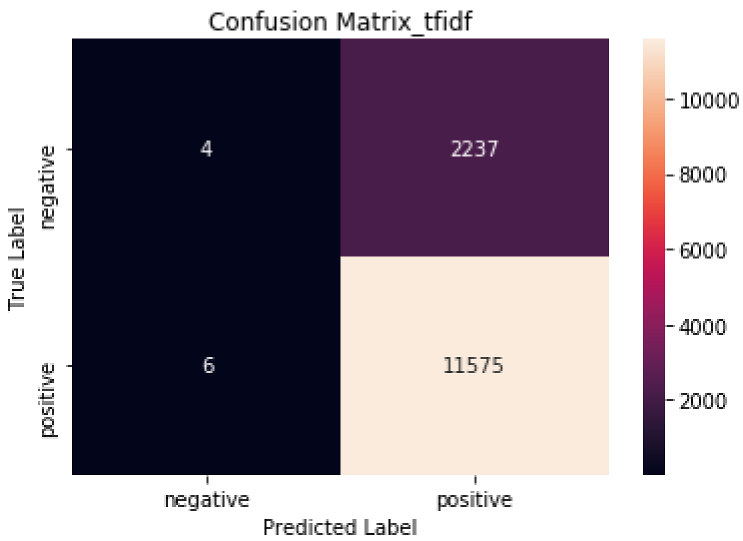
```
from sklearn.metrics import confusion_matrix
knn_brute.fit(Xbow_tr, y_tr)
cm = confusion_matrix(y_test, pred)
cm
```

Out[234]:

```
array([[    4,  2237],
       [    6, 11575]], dtype=int64)
```

In [235]:

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



[5.1.3] Applying KNN brute force on AVG W2V, SET 3

In [236]:

```
# List of sentence in X_train text
sent_of_train=[]
for sent in X_tr:
    sent_of_train.append(sent.split())

# List of sentence in X_est text
sent_of_test=[]
for sent in X_test:
    sent_of_test.append(sent.split())

    sent_of_cv=[]
for sent in X_cv:
    sent_of_cv.append(sent.split())

# Train your own Word2Vec model using your own train text corpus
# min_count = 5 considers only words that occured atleast 5 times
w2v_model=Word2Vec(sent_of_train,min_count=5,size=50, workers=4)

w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
```

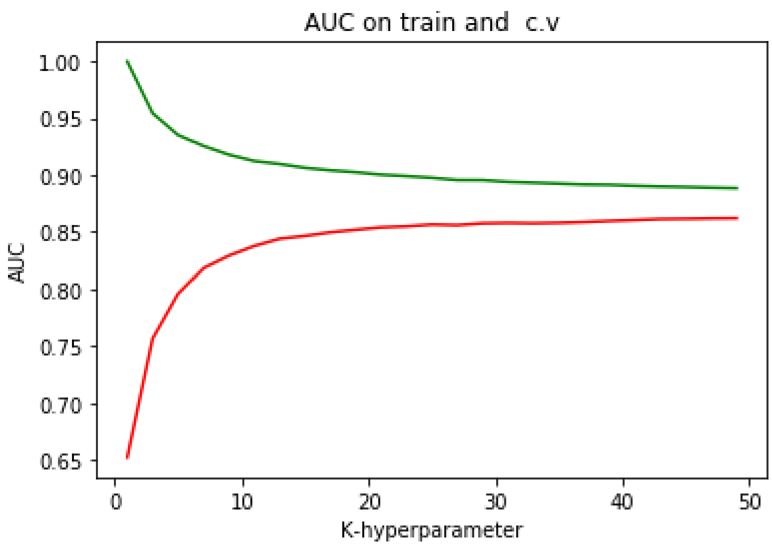
```
number of words that occured minimum 5 times  9081
```

In [238]:

```
# compute average word2vec for each review for X_train .
Xbow_tr = [];
for sent in sent_of_train:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    Xbow_tr.append(sent_vec)

# compute average word2vec for each review for X_test .
Xbow_test = [];
for sent in sent_of_test:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    Xbow_test.append(sent_vec)
  #gdfghsdgfsdgfhsdgfdhsgfhgdhgfhdghfg


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

In [239]:

```
myList = list(range(0,50))
neighbors = list(filter(lambda x: x % 2 != 0, myList))

auc1=[]
auc2=[]

for i in neighbors:
    # instantiate learning model (k = 50)
    knn = KNeighborsClassifier(n_neighbors=i,algorithm = 'brute')

    # fitting the model on crossvalidation train
    knn.fit(Xbow_tr, y_tr)

    probs = knn.predict_proba(Xbow_tr)
    preds = probs[:,1]
    roc_auc1=metrics.roc_auc_score(y_tr, preds)
    auc1.append(roc_auc1)

    #knn.fit(Xbow_cv, y_cv)
    probs = knn.predict_proba(Xbow_cv)
    preds = probs[:,1]
    roc_auc2=metrics.roc_auc_score(y_cv, preds)
    auc2.append(roc_auc2)


plt.title('AUC on train and  c.v')
plt.plot(neighbors, auc1,'g',label ='Train')
plt.plot(neighbors, auc2,'r',label ='C.V')

plt.ylabel('AUC')
plt.xlabel('K-hyperparameter')
plt.show()
```



Train curve with green and C.V curve with red color above

In [ ]:

```
print(auc1)

print('-----------------------------------------------------------------------------------
print('-----------------------------------------------------------------------------------
print(auc2)
```

In [17]:

```
# ============================== KNN with k = optimal_k ==================================
# instantiate learning model k = optimal_k
knn_brute = KNeighborsClassifier(n_neighbors=9,algorithm = 'brute')

# fitting the model
knn_brute.fit(Xbow_tr, y_tr)

# predict the response
pred = knn_brute.predict(Xbow_test)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (9, 85.646071))
```

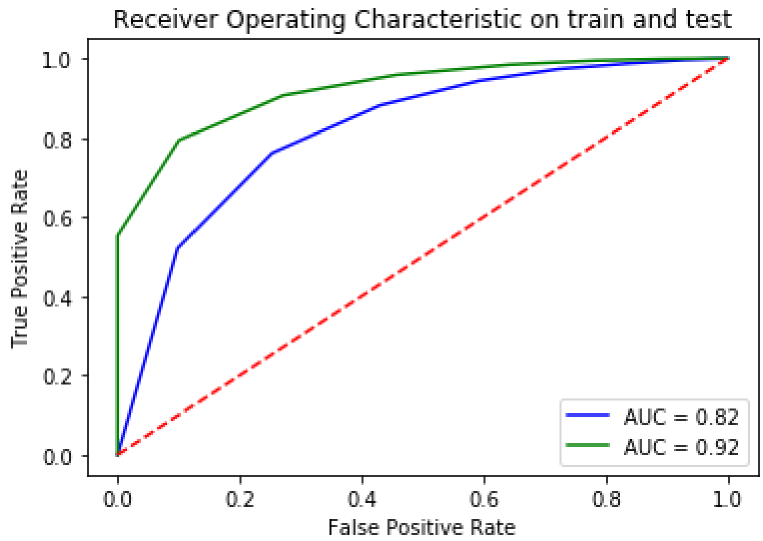
```
The accuracy of the knn classifier for k = 9 is 85.646071%
```

In [247]:

```
#knn.fit(Xbow_tr, y_tr)
probs2 = knn_brute.predict_proba(Xbow_tr)
preds2 = probs2[:,1]
fpr2, tpr2, threshold2 = metrics.roc_curve(y_tr, preds2)
roc_auc2 = metrics.auc(fpr2, tpr2)

#knn.fit(Xbow_test, y_test)
probs1 = knn_brute.predict_proba(Xbow_test)
preds1 = probs1[:,1]
fpr1, tpr1, threshold1 = metrics.roc_curve(y_test, preds1)
roc_auc1 = metrics.auc(fpr1, tpr1)
```

In [248]:

```
plt.title('Receiver Operating Characteristic on train and test')
plt.plot(fpr1, tpr1, 'b', label = 'AUC = %0.2f' % roc_auc1)
plt.plot(fpr2, tpr2, 'g', label = 'AUC = %0.2f' % roc_auc2)
#plt.plot(neighbors, auc1,'g')
#plt.plot(neighbors, auc2,'r')
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')

plt.show()
```



Train curve with green and test curve with blue color above

In [249]:

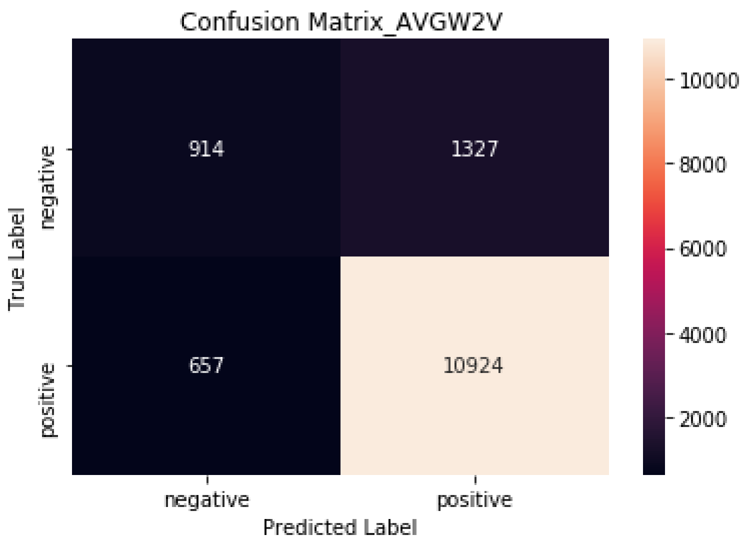
```
from sklearn.metrics import confusion_matrix
knn_brute.fit(Xbow_tr, y_tr)
cm = confusion_matrix(y_test, pred)
cm
```

Out[249]:

```
array([[  914,  1327],
       [  657, 10924]], dtype=int64)
```

In [250]:

```
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix_AVGW2V")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



[5.1.4] Applying KNN brute force on TFIDF W2V, SET 4

In [251]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
model.fit(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 [252]:

```
# 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 = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in sent_of_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.append(sent_vec)
    row += 1
```

In [253]:

```
Xbow_tr=tfidf_sent_vectors
```

In [254]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
model.fit(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 [255]:

```
# 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

Xbow_cv = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in sent_of_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
    Xbow_cv.append(sent_vec)
    row += 1
```

In [114]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
model.fit(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 [256]:

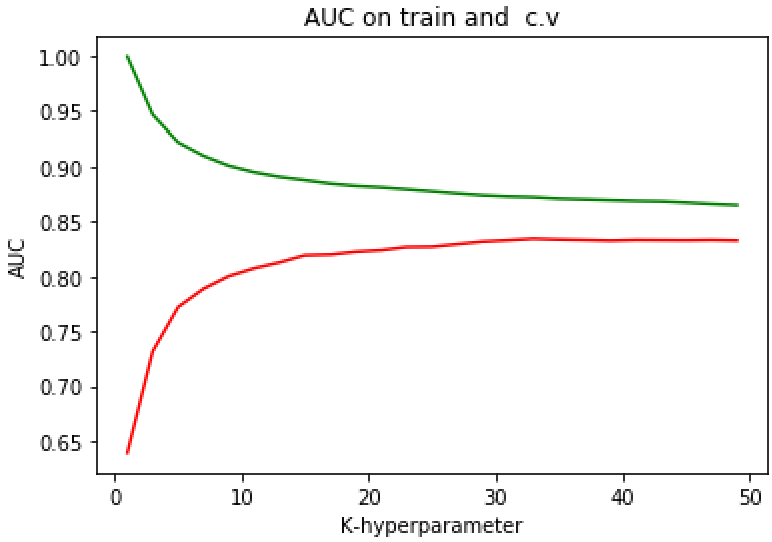
```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

Xbow_test = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in sent_of_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
    Xbow_test.append(sent_vec)
    row += 1
```

In [258]:

```
myList = list(range(0,50))
neighbors = list(filter(lambda x: x % 2 != 0, myList))

auc1=[]
auc2=[]

for i in neighbors:
    # instantiate learning model (k = 50)
    knn = KNeighborsClassifier(n_neighbors=i,algorithm = 'brute')

    # fitting the model on crossvalidation train
    knn.fit(Xbow_tr, y_tr)

    probs = knn.predict_proba(Xbow_tr)
    preds = probs[:,1]
    roc_auc1=metrics.roc_auc_score(y_tr, preds)
    auc1.append(roc_auc1)

   # knn.fit(Xbow_cv, y_cv)
    probs = knn.predict_proba(Xbow_cv)
    preds = probs[:,1]
    roc_auc2=metrics.roc_auc_score(y_cv, preds)
    auc2.append(roc_auc2)
```

In [260]:

```
plt.title('AUC on train and  c.v')
plt.plot(neighbors, auc1,'g',label ='Train')
plt.plot(neighbors, auc2,'r',label ='C.V')

plt.ylabel('AUC')
plt.xlabel('K-hyperparameter')
plt.show()
```



Train curve with green and C.V curve with red color above

In [261]:

```
print(auc1)

print('-----------------------------------------------------------------------------------
print('-----------------------------------------------------------------------------------
print(auc2)
```

```
[0.9996020164499868, 0.9467830421406838, 0.9219624839393903, 0.9097507415940
04, 0.9007731949490128, 0.8949097893371448, 0.8907190292573465, 0.8877762931
043909, 0.8846423992760883, 0.8825180708093968, 0.881262484383828, 0.8794666
801738162, 0.8775989264073637, 0.8756538263386452, 0.8739074252802904, 0.872
8044221429118, 0.8722026955651941, 0.8708359432301844, 0.8701854060846651,
0.8694546443724394, 0.8687708026033337, 0.8684185891015787, 0.86729489491124
96, 0.8660144199637549, 0.8649610033968458]
----------------------------------------------------------------------------
----------------
----------------------------------------------------------------------------
--------------
[0.639063147546935, 0.7316093350951947, 0.7719779919372816, 0.78864267210992
07, 0.8001959443342197, 0.8073410353065338, 0.8127235098678245, 0.8192347712
582068, 0.8198607121417066, 0.8224745659770903, 0.8238831282297776, 0.826731
3779462921, 0.8269755370680566, 0.8293323052510769, 0.831708053173889, 0.832
9405646714716, 0.8343052704472328, 0.833605207040843, 0.8331689073434098, 0.
8326163079235492, 0.8331750777114902, 0.8330447189225506, 0.832923889056469
4, 0.8331708599915365, 0.8327297567797114]
```

In [18]:

```
# ============================== KNN with k = optimal_k ==================================
# instantiate learning model k = optimal_k
knn_brute = KNeighborsClassifier(n_neighbors=11,algorithm = 'brute')

# fitting the model
knn_brute.fit(Xbow_tr, y_tr)

# predict the response
pred = knn_brute.predict(Xbow_test)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (11, 85.523079))
```

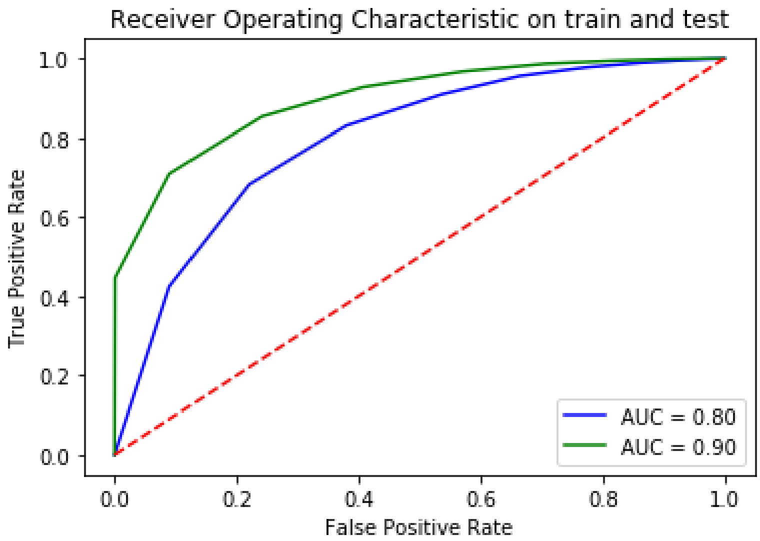
```

The accuracy of the knn classifier for k = 11 is 85.523079%
```

In [265]:

```
probs2 = knn_brute.predict_proba(Xbow_tr)
preds2 = probs2[:,1]
fpr2, tpr2, threshold2 = metrics.roc_curve(y_tr, preds2)
roc_auc2 = metrics.auc(fpr2, tpr2)

#knn.fit(Xbow_test, y_test)
probs1 = knn_brute.predict_proba(Xbow_test)
preds1 = probs1[:,1]
fpr1, tpr1, threshold1 = metrics.roc_curve(y_test, preds1)
roc_auc1 = metrics.auc(fpr1, tpr1)
```

In [266]:

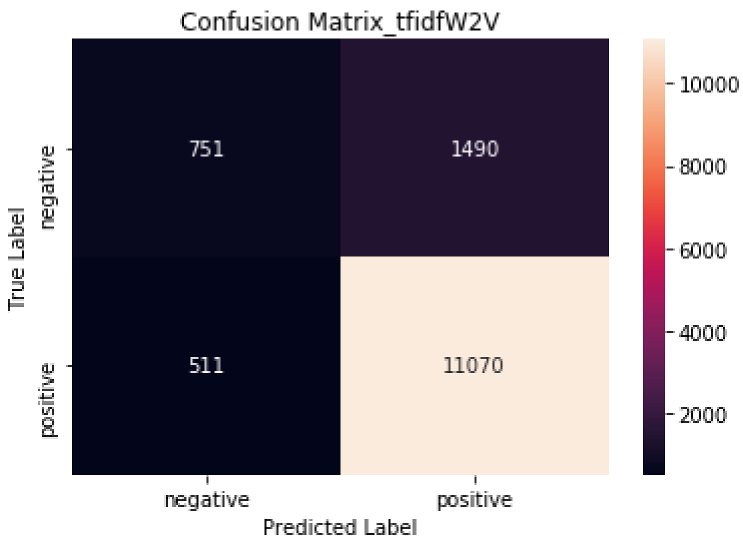
```
plt.title('Receiver Operating Characteristic on train and test')
plt.plot(fpr1, tpr1, 'b', label = 'AUC = %0.2f' % roc_auc1)
plt.plot(fpr2, tpr2, 'g', label = 'AUC = %0.2f' % roc_auc2)
#plt.plot(neighbors, auc1,'g')
#plt.plot(neighbors, auc2,'r')
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')

plt.show()
```



Train curve with green and test curve with blue color above

In [264]:

```
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix_tfidfW2V")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



In [263]:

```
cm = confusion_matrix(y_test, pred)
cm
```

Out[263]:

```
array([[  751,  1490],
       [  511, 11070]], dtype=int64)
```

[5.2] Applying KNN kd-tree

[5.2.1] Applying KNN kd-tree on BOW, SET 5

In [286]:

```
# spilliting data for kd-Tree version of KNN
from sklearn.model_selection import train_test_split

# split the data set into train and test
X_1, X_test, y_1, y_test = train_test_split(preprocessed_reviews, final['Score'], test_size

# split the train data set into cross validation train and cross validation test
X_tr, X_cv, y_tr, y_cv = train_test_split(X_1, y_1, test_size=0.3)
```

In [346]:

```
#code for kd-TREE version
count_vect = CountVectorizer(min_df = 10, max_features=500)
Xbow_tr = count_vect.fit_transform(X_tr)
Xbow_test = count_vect.transform(X_test)
Xbow_cv = count_vect.transform(X_cv)
print("the type of count vectorizer :",type(X_tr))
print("the shape of out text BOW vectorizer : ",Xbow_tr.get_shape())
print("the number of unique words :", Xbow_tr.get_shape()[1])
```

```
the type of count vectorizer : <class 'list'>
the shape of out text BOW vectorizer :  (9800, 500)
the number of unique words : 500
```

In [347]:

```
type(Xbow_tr)
```

Out[347]:

```
scipy.sparse.csr.csr_matrix
```

In [348]:

```
Xbow_tr = Xbow_tr.todense()
```

In [349]:

```
type(Xbow_tr)
```

Out[349]:

```
numpy.matrixlib.defmatrix.matrix
```

In [350]:

```
print(Xbow_tr.shape)
```

```
(9800, 500)
```

In [351]:

```
Xbow_test = Xbow_test.todense()
```

In [352]:

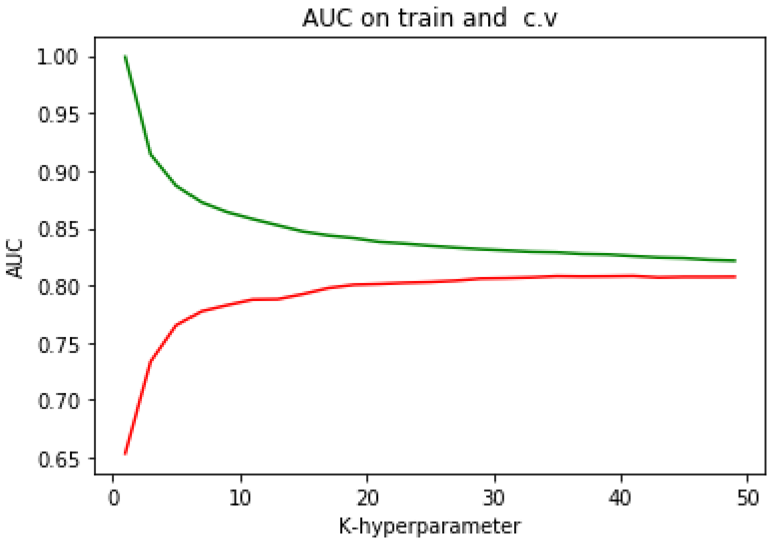
```
Xbow_cv = Xbow_cv.todense()
```

In [294]:

```
myList = list(range(0,50))
neighbors = list(filter(lambda x: x % 2 != 0, myList))

auc1=[]
auc2=[]

for i in neighbors:
    # instantiate learning model (k = 50)
    knn = KNeighborsClassifier(n_neighbors=i,algorithm = 'kd_tree')

    # fitting the model on crossvalidation train
    knn.fit(Xbow_tr, y_tr)

    probs = knn.predict_proba(Xbow_tr)
    preds = probs[:,1]
    roc_auc1=metrics.roc_auc_score(y_tr, preds)
    auc1.append(roc_auc1)

    #knn.fit(Xbow_cv, y_cv)
    probs = knn.predict_proba(Xbow_cv)
    preds = probs[:,1]
    roc_auc2=metrics.roc_auc_score(y_cv, preds)
    auc2.append(roc_auc2)
```

In [324]:

```
plt.title('AUC on train and  c.v')
plt.plot(neighbors, auc1,'g',label ='Train')
plt.plot(neighbors, auc2,'r',label ='C.V')

plt.ylabel('AUC')
plt.xlabel('K-hyperparameter')
plt.show()
```



Train curve with green and C.V curve with red color above

In [297]:

```
print(auc1)

print('-------------------------------------------------------------------------------------
print('-------------------------------------------------------------------------------------
print(auc2)
```

```
[0.9991005761292283, 0.9054449645067689, 0.8782976355867036, 0.8663309188048
325, 0.8554742102320381, 0.8487192613111334, 0.841970714616496, 0.8347582510
469418, 0.8302507947652552, 0.8301507822097782, 0.8289759959196479, 0.825798
7355247443, 0.8243646146108762, 0.821696019984905, 0.8184068095501298, 0.817
4091070601619, 0.8168191685855561, 0.8153481681807931, 0.8150711607449631,
0.8137899819030404, 0.8132014883753356, 0.8120448639562485, 0.81058360027060
07, 0.808553227575669, 0.8068619728033428]
----------------------------------------------------------------------------
----------------
----------------------------------------------------------------------------
--------------
[0.6355500333135253, 0.6989693815689169, 0.7216756403099659, 0.7306251538124
479, 0.7422321862676247, 0.7471711114712574, 0.7536517025912519, 0.757752807
6399018, 0.7608019255817862, 0.7656996740676715, 0.7689832471983625, 0.76758
70803546239, 0.7706462823906506, 0.7729707861391726, 0.7722480927256465, 0.7
755350272210518, 0.7792831889747237, 0.7797819914885443, 0.7796214863234474,
0.7813528292486749, 0.781739146093314, 0.7839053055540549, 0.784637482818024
1, 0.7858702633269107, 0.7855125180823413]
```

In [19]:

```
# ============================== KNN with k = optimal_k ===================================
# instantiate learning model k = optimal_k
knn_brute = KNeighborsClassifier(n_neighbors=11,algorithm = 'kd_tree')

# fitting the model
knn_brute.fit(Xbow_tr, y_tr)

# predict the response
pred = knn_brute.predict(Xbow_test)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (11, 69.816667))
```

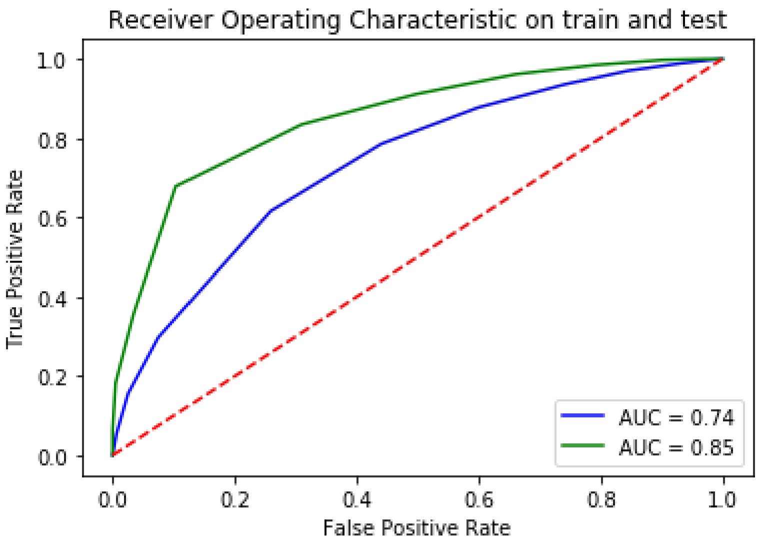
```
The accuracy of the knn classifier for k = 11 is 69.816667%
```

In [326]:

```
probs2 = knn_brute.predict_proba(Xbow_tr)
preds2 = probs2[:,1]
fpr2, tpr2, threshold2 = metrics.roc_curve(y_tr, preds2)
roc_auc2 = metrics.auc(fpr2, tpr2)

#knn.fit(Xbow_test, y_test)
probs1 = knn_brute.predict_proba(Xbow_test)
preds1 = probs1[:,1]
fpr1, tpr1, threshold1 = metrics.roc_curve(y_test, preds1)
roc_auc1 = metrics.auc(fpr1, tpr1)
```

In [300]:

```
plt.title('Receiver Operating Characteristic on train and test')
plt.plot(fpr1, tpr1, 'b', label = 'AUC = %0.2f' % roc_auc1)
plt.plot(fpr2, tpr2, 'g', label = 'AUC = %0.2f' % roc_auc2)
#plt.plot(neighbors, auc1,'g')
#plt.plot(neighbors, auc2,'r')
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Train curve with green and test curve with blue color above

In [301]:

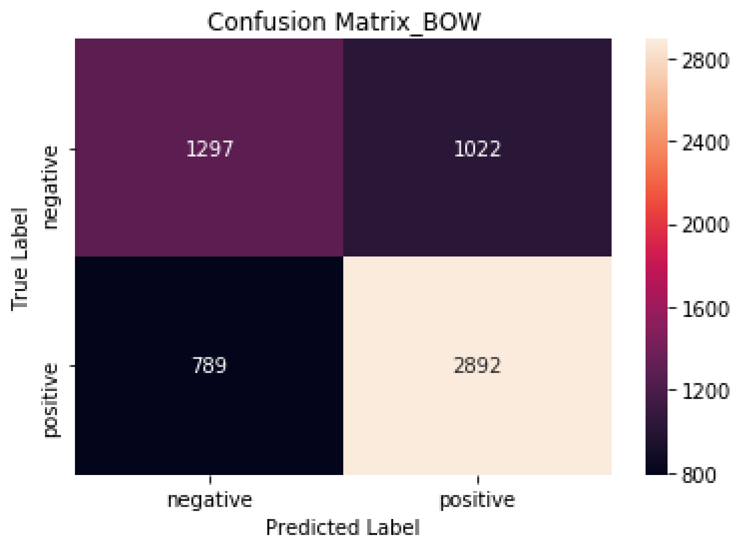
```
cm = confusion_matrix(y_test, pred)
cm
```

Out[301]:

```
array([[1297, 1022],
       [ 789, 2892]], dtype=int64)
```

In [302]:

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



[5.2.2] Applying KNN kd-tree on TFIDF, SET 6

In [303]:

```
count_vect  = TfidfVectorizer(ngram_range=(1,2), min_df=10, max_features=500)

Xbow_tr = count_vect.fit_transform(X_tr)
Xbow_test = count_vect.transform(X_test)
Xbow_cv = count_vect.transform(X_cv)
print("the type of count vectorizer :",type(X_tr))
print("the shape of out text BOW vectorizer : ",Xbow_tr.get_shape())
print("the number of unique words :", Xbow_tr.get_shape()[1])
```

```
the type of count vectorizer : <class 'list'>
the shape of out text BOW vectorizer :  (9800, 500)
the number of unique words : 500
```

In [304]:

```
Xbow_tr = Xbow_tr.todense()
```

In [305]:

```
Xbow_test = Xbow_test.todense()
```

In [306]:

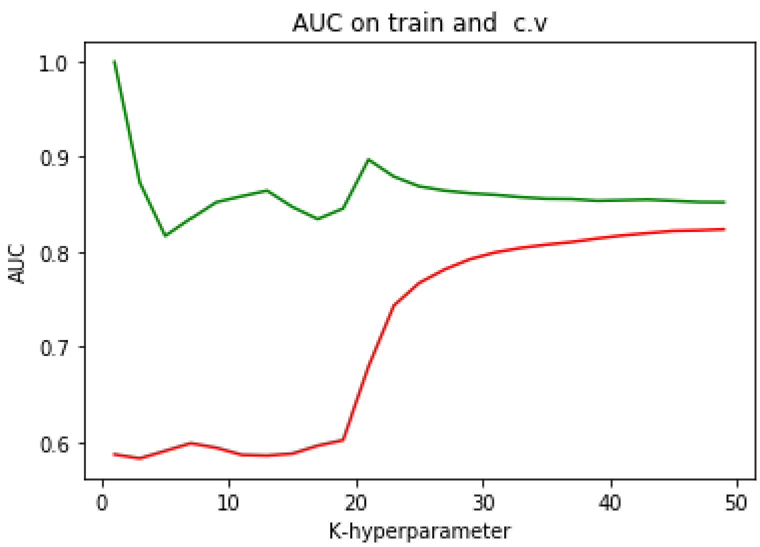
```
Xbow_cv = Xbow_cv.todense()
```

In [307]:

```
myList = list(range(0,50))
neighbors = list(filter(lambda x: x % 2 != 0, myList))

auc1=[]
auc2=[]

for i in neighbors:
    # instantiate learning model (k = 50)
    knn = KNeighborsClassifier(n_neighbors=i,algorithm = 'kd_tree')

    # fitting the model on crossvalidation train
    knn.fit(Xbow_tr, y_tr)

    probs = knn.predict_proba(Xbow_tr)
    preds = probs[:,1]
    roc_auc1=metrics.roc_auc_score(y_tr, preds)
    auc1.append(roc_auc1)

    #knn.fit(Xbow_cv, y_cv)
    probs = knn.predict_proba(Xbow_cv)
    preds = probs[:,1]
    roc_auc2=metrics.roc_auc_score(y_cv, preds)
    auc2.append(roc_auc2)
```

In [308]:

```
plt.title('AUC on train and  c.v')
plt.plot(neighbors, auc1,'g',label ='Train')
plt.plot(neighbors, auc2,'r',label ='C.V')

plt.ylabel('AUC')
plt.xlabel('K-hyperparameter')
plt.show()
```



Train curve with green and C.V curve with red color above

In [309]:

```
print(auc1)

print('-----------------------------------------------------------------------------
print('-----------------------------------------------------------------------------
print(auc2)
```

```
[0.9991005761292283, 0.87278807528449, 0.8168072088712092, 0.835030367893262
2, 0.852226525348192, 0.8584542242600272, 0.864394000829444, 0.8472200733268
316, 0.8344460758265451, 0.8456032444348648, 0.897051045661389, 0.8791834769
566091, 0.8689066144600682, 0.8644176090388054, 0.8616340855593518, 0.860000
2285239099, 0.8575319924581775, 0.855840715455899, 0.8555459907480717, 0.853
7126198905788, 0.8543238546595047, 0.8549712797908037, 0.8537173548704223,
0.8523942058785242, 0.8521865781238773]
----------------------------------------------------------------------------
----------------
----------------------------------------------------------------------------
--------------
[0.5876253759026164, 0.5834638863378532, 0.5913762987773036, 0.5993454942706
74, 0.5946286592356497, 0.587201963997383, 0.5864485381064712, 0.58850713389
6362, 0.5968963799302517, 0.6027006164502788, 0.6795247270391779, 0.74353663
58741647, 0.7673149298615237, 0.7814878840809371, 0.7922358477541882, 0.7994
223254641384, 0.8040937820755227, 0.8075186525729446, 0.8103072647494882, 0.
8140687519132768, 0.8172702117059526, 0.8196960365908558, 0.822014777999868,
0.8226417925677825, 0.823715748593929]
```

In [311]:

```
# ============================== KNN with k = optimal_k ==================================
# instantiate learning model k = optimal_k
knn_brute = KNeighborsClassifier(n_neighbors=23,algorithm = 'kd_tree')

# fitting the model
knn_brute.fit(Xbow_tr, y_tr)

# predict the response
pred = knn_brute.predict(Xbow_test)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (23, acc))
```

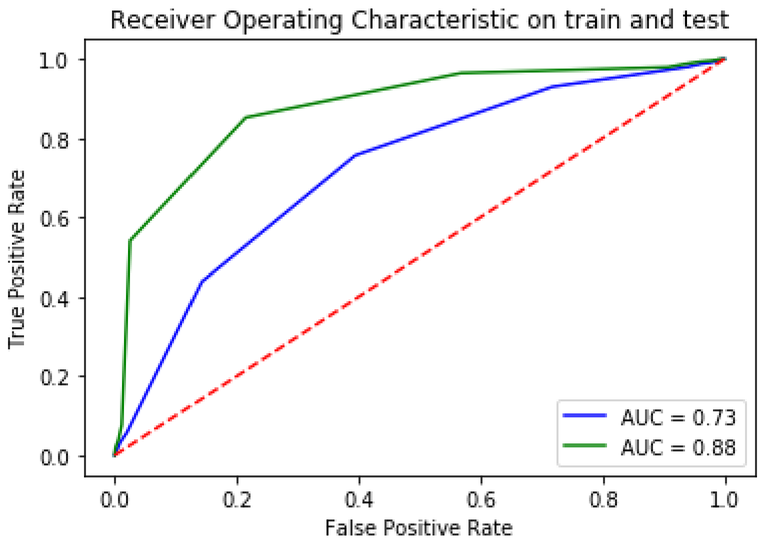
```
The accuracy of the knn classifier for k = 23 is 61.633333%
```

In [312]:

```
probs2 = knn_brute.predict_proba(Xbow_tr)
preds2 = probs2[:,1]
fpr2, tpr2, threshold2 = metrics.roc_curve(y_tr, preds2)
roc_auc2 = metrics.auc(fpr2, tpr2)

#knn.fit(Xbow_test, y_test)
probs1 = knn_brute.predict_proba(Xbow_test)
preds1 = probs1[:,1]
fpr1, tpr1, threshold1 = metrics.roc_curve(y_test, preds1)
roc_auc1 = metrics.auc(fpr1, tpr1)
```

In [313]:

```
plt.title('Receiver Operating Characteristic on train and test')
plt.plot(fpr1, tpr1, 'b', label = 'AUC = %0.2f' % roc_auc1)
plt.plot(fpr2, tpr2, 'g', label = 'AUC = %0.2f' % roc_auc2)
#plt.plot(neighbors, auc1,'g')
#plt.plot(neighbors, auc2,'r')
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Train curve with green and test curve with blue color above

In [314]:

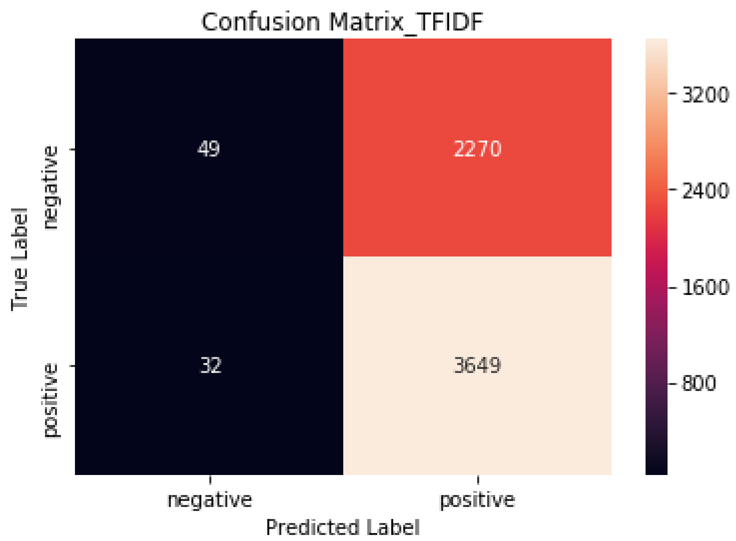
```
cm = confusion_matrix(y_test, pred)
cm
```

Out[314]:

```
array([[  49, 2270],
       [  32, 3649]], dtype=int64)
```

In [315]:

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



[5.2.3] Applying KNN kd-tree on AVG W2V, SET 7

In [3]:

```
# Please write all the code with proper documentation
```

In [316]:

```
# List of sentence in X_train text
sent_of_train=[]
for sent in X_tr:
    sent_of_train.append(sent.split())

# List of sentence in X_est text
sent_of_test=[]
for sent in X_test:
    sent_of_test.append(sent.split())

    sent_of_cv=[]
for sent in X_cv:
    sent_of_cv.append(sent.split())

# Train your own Word2Vec model using your own train text corpus
# min_count = 5 considers only words that occured atleast 5 times
w2v_model=Word2Vec(sent_of_train,min_count=5,size=50, workers=4)

w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
```

```
number of words that occured minimum 5 times  6071
```

In [317]:

```
# compute average word2vec for each review for X_train .
Xbow_tr = [];
for sent in sent_of_train:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    Xbow_tr.append(sent_vec)

# compute average word2vec for each review for X_test .
Xbow_test = [];
for sent in sent_of_test:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    Xbow_test.append(sent_vec)
  #gdfghsdgfsdgfhsdgfdhsgfhgdhgfhdghfg


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

In [321]:

```
myList = list(range(0,50))
neighbors = list(filter(lambda x: x % 2 != 0, myList))

auc1=[]
auc2=[]

for i in neighbors:
    # instantiate learning model (k = 50)
    knn = KNeighborsClassifier(n_neighbors=i,algorithm = 'kd_tree')

    # fitting the model on crossvalidation train
    knn.fit(Xbow_tr, y_tr)

    probs = knn.predict_proba(Xbow_tr)
    preds = probs[:,1]
    roc_auc1=metrics.roc_auc_score(y_tr, preds)
    auc1.append(roc_auc1)

    #knn.fit(Xbow_cv, y_cv)
    probs = knn.predict_proba(Xbow_cv)
    preds = probs[:,1]
    roc_auc2=metrics.roc_auc_score(y_cv, preds)
    auc2.append(roc_auc2)
```

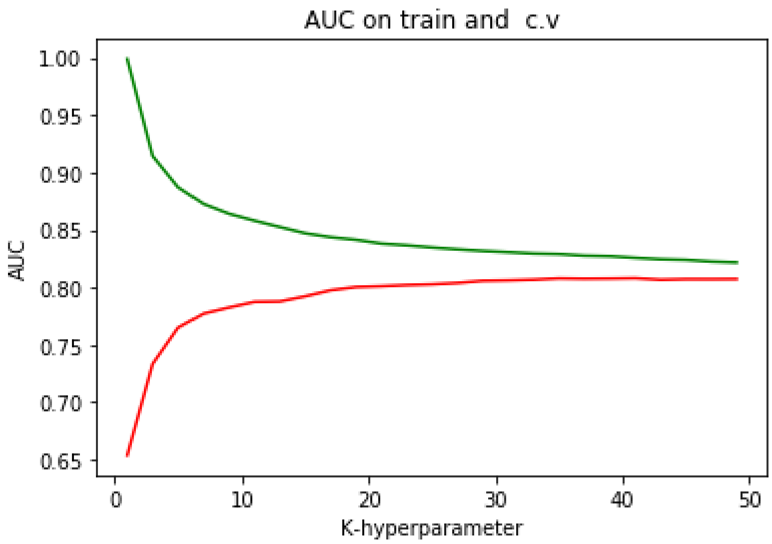
In [322]:

```
print(auc1)

print('-------------------------------------------------------------------------------
print('-------------------------------------------------------------------------------
print(auc2)
```

```
[0.9989399869536856, 0.9144433157556655, 0.8871416442766586, 0.8725335645604
053, 0.8640012864918005, 0.8579838162390334, 0.8524471798548972, 0.846996751
2258485, 0.8435960575802003, 0.841320066142111, 0.8380031349567956, 0.836511
7941456621, 0.8346259383707473, 0.8330073755424997, 0.8316098229001944, 0.83
05131748924605, 0.8293425011435087, 0.8288619118043536, 0.8274726953941673,
0.8269992418697061, 0.8255885735555333, 0.8243600574706511, 0.82379546114168
77, 0.8223335527874226, 0.8215689091175526]
----------------------------------------------------------------------------
----------------
----------------------------------------------------------------------------
--------------
[0.6538760736859165, 0.7334677879218964, 0.765112875827586, 0.77744752369462
01, 0.7827524775058674, 0.7876312582908661, 0.7879921248026699, 0.7924721036
740917, 0.7977460849104737, 0.8005569061038782, 0.8012624325476142, 0.802268
2008895612, 0.8029373525651413, 0.804091381100727, 0.8059891115793012, 0.806
3643839398796, 0.806990438117876, 0.8080840821373477, 0.8078013673551464, 0.
807947226573989, 0.8082526305680106, 0.8069635472001633, 0.8074252546533892,
0.8074169712903438, 0.8075181723779854]
```

In [323]:

```
plt.title('AUC on train and  c.v')
plt.plot(neighbors, auc1,'g',label ='Train')
plt.plot(neighbors, auc2,'r',label ='C.V')

plt.ylabel('AUC')
plt.xlabel('K-hyperparameter')
plt.show()
```



Train curve with green and C.V curve with red color above

In [20]:

```
# ============================== KNN with k = optimal_k ==================================
# instantiate learning model k = optimal_k
knn_brute = KNeighborsClassifier(n_neighbors=11,algorithm = 'kd_tree')

# fitting the model
knn_brute.fit(Xbow_tr, y_tr)

# predict the response
pred = knn_brute.predict(Xbow_test)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (11, 71.933333))
```

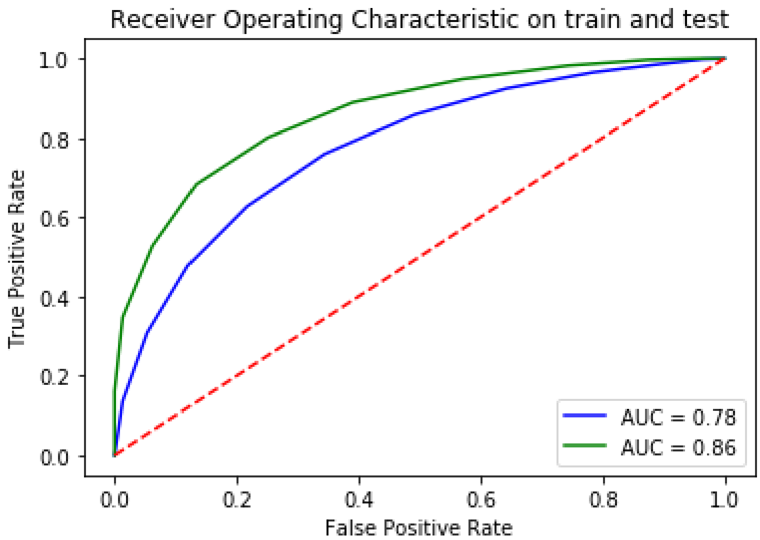
```
The accuracy of the knn classifier for k = 11 is 71.933333%
```

In [328]:

```
probs2 = knn_brute.predict_proba(Xbow_tr)
preds2 = probs2[:,1]
fpr2, tpr2, threshold2 = metrics.roc_curve(y_tr, preds2)
roc_auc2 = metrics.auc(fpr2, tpr2)

#knn.fit(Xbow_test, y_test)
probs1 = knn_brute.predict_proba(Xbow_test)
preds1 = probs1[:,1]
fpr1, tpr1, threshold1 = metrics.roc_curve(y_test, preds1)
roc_auc1 = metrics.auc(fpr1, tpr1)
```

In [329]:

```
plt.title('Receiver Operating Characteristic on train and test')
plt.plot(fpr1, tpr1, 'b', label = 'AUC = %0.2f' % roc_auc1)
plt.plot(fpr2, tpr2, 'g', label = 'AUC = %0.2f' % roc_auc2)
#plt.plot(neighbors, auc1,'g')
#plt.plot(neighbors, auc2,'r')
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Train curve with green and test curve with blue color above

In [330]:

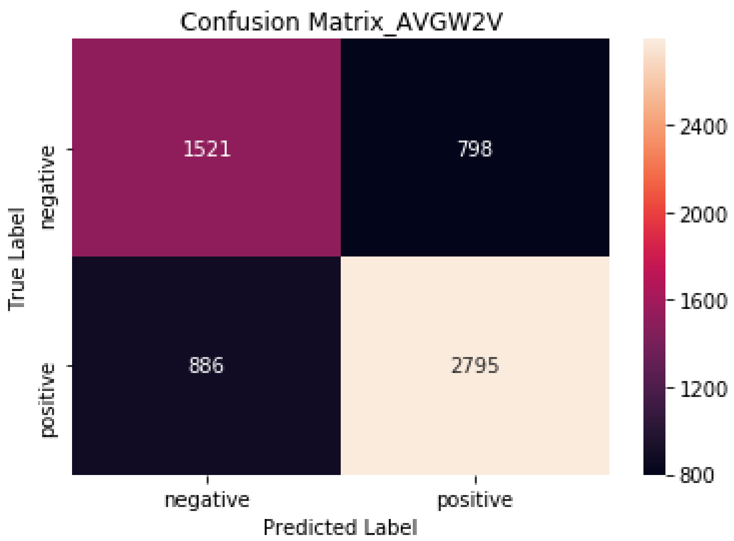
```
cm = confusion_matrix(y_test, pred)
cm
```

Out[330]:

```
array([[1521,  798],
       [ 886, 2795]], dtype=int64)
```

In [331]:

```
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix_AVGW2V")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



[5.2.4] Applying KNN kd-tree on TFIDF W2V, SET 8

In [332]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
model.fit(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 [333]:

```
# 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

Xbow_tr = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in sent_of_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
    Xbow_tr.append(sent_vec)
    row += 1
```

In [334]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
model.fit(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 [335]:

```
# 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

Xbow_test = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in sent_of_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
    Xbow_test.append(sent_vec)
    row += 1
```

In [336]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
model.fit(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 [337]:

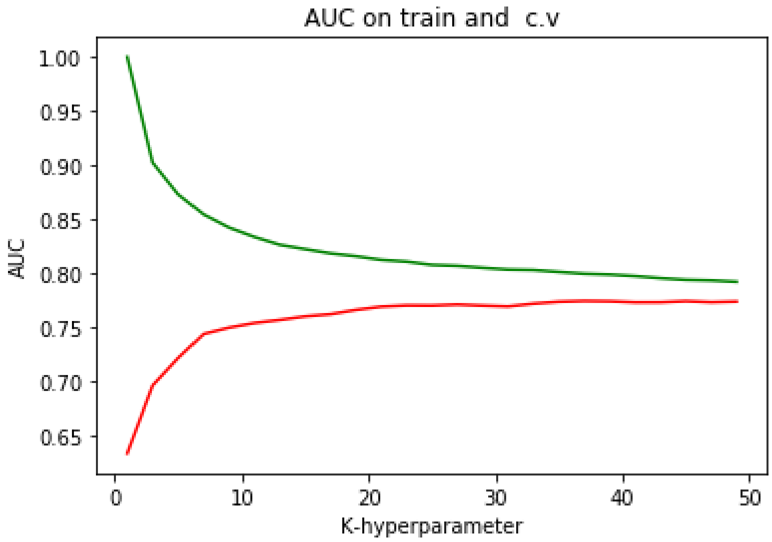
```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

Xbow_cv = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in sent_of_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
    Xbow_cv.append(sent_vec)
    row += 1
```

In [338]:

```
myList = list(range(0,50))
neighbors = list(filter(lambda x: x % 2 != 0, myList))

auc1=[]
auc2=[]

for i in neighbors:
    # instantiate learning model (k = 50)
    knn = KNeighborsClassifier(n_neighbors=i,algorithm = 'kd_tree')

    # fitting the model on crossvalidation train
    knn.fit(Xbow_tr, y_tr)

    probs = knn.predict_proba(Xbow_tr)
    preds = probs[:,1]
    roc_auc1=metrics.roc_auc_score(y_tr, preds)
    auc1.append(roc_auc1)

    #knn.fit(Xbow_cv, y_cv)
    probs = knn.predict_proba(Xbow_cv)
    preds = probs[:,1]
    roc_auc2=metrics.roc_auc_score(y_cv, preds)
    auc2.append(roc_auc2)
```

In [339]:

```
plt.title('AUC on train and  c.v')
plt.plot(neighbors, auc1,'g',label ='Train')
plt.plot(neighbors, auc2,'r',label ='C.V')

plt.ylabel('AUC')
plt.xlabel('K-hyperparameter')
plt.show()
```



Train curve with green and C.V curve with red color above

In [340]:

```
print(auc1)

print('-----------------------------------------------------------------------------------
print('-----------------------------------------------------------------------------------
print(auc2)
```

```
[0.9997273718647764, 0.9024551470698078, 0.8731571591828198, 0.8544397838608
411, 0.842313433791377, 0.8336363053534704, 0.8263768698946691, 0.8223508699
202781, 0.8184520919630007, 0.8158185095119299, 0.8124290864521068, 0.810807
5225802963, 0.8077396780107002, 0.806888648746124, 0.8050804866545593, 0.803
4677925337237, 0.8030259055415536, 0.8012633148522401, 0.7995986471028858,
0.7988177088779671, 0.7973951475393043, 0.7954806595858044, 0.79396357648988
7, 0.7934001805783486, 0.7921633282613475]
----------------------------------------------------------------------------
----------------
----------------------------------------------------------------------------
--------------
[0.6338483424270254, 0.6961842508058271, 0.7214671156489535, 0.7438814158548
37, 0.7497391941128098, 0.7540056062761482, 0.7568502812141731, 0.7601354149
784812, 0.762148632344732, 0.7661335302132667, 0.7691478340206123, 0.7703090
654805851, 0.7701699289911703, 0.7710814590723833, 0.7701220295439948, 0.769
3342697135036, 0.7719228806895599, 0.7737081254989525, 0.774403327751067, 0.
7741280559907323, 0.773148578322799, 0.7732017599145252, 0.7742324983943482,
0.7732686270625875, 0.7738590267648666]
```

In [21]:

```
# ============================== KNN with k = optimal_k ==================================
# instantiate learning model k = optimal_k
knn_brute = KNeighborsClassifier(n_neighbors=9,algorithm = 'kd_tree')

# fitting the model
knn_brute.fit(Xbow_tr, y_tr)

# predict the response
pred = knn_brute.predict(Xbow_test)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (9, 68.25))
```

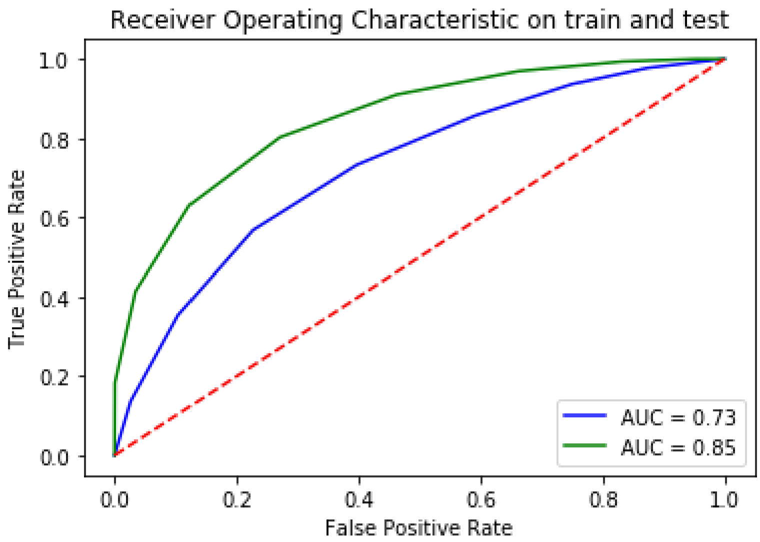
```

The accuracy of the knn classifier for k = 9 is 68.250000%
```

In [342]:

```
probs2 = knn_brute.predict_proba(Xbow_tr)
preds2 = probs2[:,1]
fpr2, tpr2, threshold2 = metrics.roc_curve(y_tr, preds2)
roc_auc2 = metrics.auc(fpr2, tpr2)

#knn.fit(Xbow_test, y_test)
probs1 = knn_brute.predict_proba(Xbow_test)
preds1 = probs1[:,1]
fpr1, tpr1, threshold1 = metrics.roc_curve(y_test, preds1)
roc_auc1 = metrics.auc(fpr1, tpr1)
```

In [343]:

```
plt.title('Receiver Operating Characteristic on train and test')
plt.plot(fpr1, tpr1, 'b', label = 'AUC = %0.2f' % roc_auc1)
plt.plot(fpr2, tpr2, 'g', label = 'AUC = %0.2f' % roc_auc2)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Train curve with green and train curve with blue color above

In [344]:

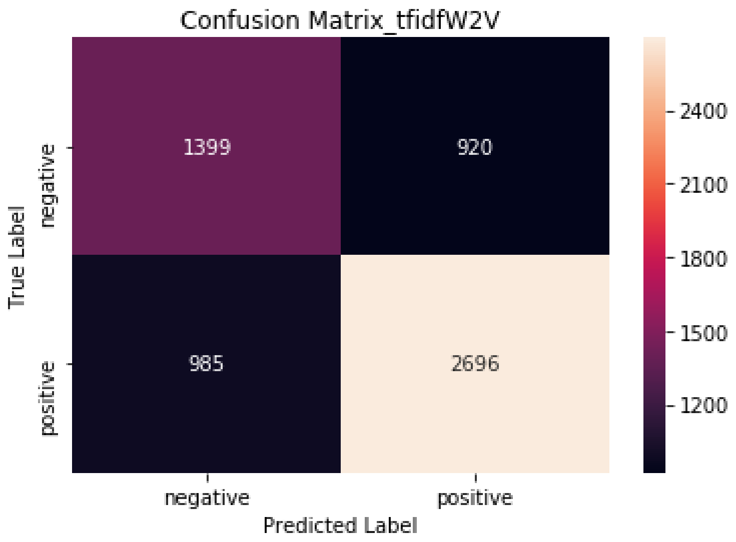
```
cm = confusion_matrix(y_test, pred)
cm
```

Out[344]:

```
array([[1399,  920],
       [ 985, 2696]], dtype=int64)
```

In [345]:

```
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix_tfidfW2V")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



[6] Conclusions

In [22]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vactorizer", "Model", "Hyperparameter(K)", "AUC%"]

x.add_row(["BOW", "BRUTE", 9, 83.063233])
x.add_row(["TFIDF", "BRUTE", 5, 83.772247])
x.add_row(["AVGW2V", "BRUTE", 9,85.646071])
x.add_row(["TFIDFW2V","BRUTE", 11,85.523079])
x.add_row(["BOW", "KD-TREE", 11, 69.816667])
x.add_row(["TFIDF","KD-TREE", 23,61.633333])
x.add_row(["AVGW2V", "KD-TREE", 11,71.933333])
x.add_row(["TFIDFW2V","KD-TREE", 9,68.250000])
print(x)
```

```
+------------+---------+-------------------+-----------+
| Vactorizer |  Model  | Hyperparameter(K) |    AUC%   |
+------------+---------+-------------------+-----------+
|    BOW     |  BRUTE  |         9         | 83.063233 |
|   TFIDF    |  BRUTE  |         5         | 83.772247 |
|   AVGW2V   |  BRUTE  |         9         | 85.646071 |
|  TFIDFW2V  |  BRUTE  |         11        | 85.523079 |
|    BOW     | KD-TREE |         11        | 69.816667 |
|   TFIDF    | KD-TREE |         23        | 61.633333 |
|   AVGW2V   | KD-TREE |         11        | 71.933333 |
|  TFIDFW2V  | KD-TREE |         9         |   68.25   |
+------------+---------+-------------------+-----------+
```

#------>>> my system configration is 4GB RAM and 1.3 GHz processor ------>>> We have done all below steps saparately for KNN BRUTE knn and KNN kd-Tree version. ------>>> WE have taken 50k points for brute and 20k points for kd-tree version

STEP 1 :- Data cleaning (removing duplication)

STEP 2 :- Text Preprocessing

STEP 3:- Featurization on text reviews i.e BOW,TFIDF,avgW2V,TFIDF-W2V.

STEP 4:- Using AUC as a metric and plot curve for train(predected value on itself) and C.V predected value on train VS for each odd values of k 1 to 50.

STEP 5:- Plot "AUC VS K" to analise overfitting and underfitting.

STEP 6:- Once , we analise optimal value of K then train KNN again with this analised optimal K and make predictions on test_data. ------->Here we take odd value of K to avoid the ambiguity on K-NN.

STEP 7:- Compute test accuracy using predicted values of test_data.

STEP 8:- Plot ROC curve for train and test on K-NN model.

STEP 9:- Plot Seaborn Heatmap for representation of Confusion Matrix on k-NN model.

Repeat from STEP 4 to STEP 9 for each of these four vectorizers : Bag Of Words(BoW), TFIDF , Avg Word2Vec and TFIDF Word2Vec saparately on brute and kd-tree version of KNN as we have taken different no. of points for different versions.

AT THE END WE MAKE A TABLE TO COMPAIR OUR RESULTS OF KNN WITH DIFFERENT VECTORIZERS AND DIFFERENT VERSIONS.