[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 [0]:
```
%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
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```python
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc, accuracy_score
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 [6]:

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

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 50
0000 data points
# you can change the number to any other number based on your computing
 power

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

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

# Give reviews with Score>3 a positive rating(1), and reviews with a sc
ore<3 a negative rating(0).
def partition(x):
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    if x < 3:
        return 0
    return 1

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

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

Out[6]:

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [0]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [18]:
```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[18]:

	UserId	ProductId	ProfileName	Time	Score	Text	COU
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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	UserId	ProductId	ProfileName	Time	Score	Text	COU
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

[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 [19]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[19]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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 [0]:
```
sample_data = filtered_data.head(50000)
```

In [0]:
```
#Sorting data according to ProductId in ascending order
sorted_data=sample_data.sort_values('ProductId', axis=0, ascending=True
, inplace=False, kind='quicksort', na_position='last')
```

In [9]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time"
,"Text"}, keep='first', inplace=False)
final.shape
```

Out[9]:
```
(46072, 10)
```

In [22]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[22]:
```
92.144
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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 [23]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[23]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2

In [0]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [10]:
```
#Before starting the next phase of preprocessing lets see the number of
 entries left
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
print(final.shape)

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

```
(46072, 10)
```

Out[10]:
```
1    38480
0     7592
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 [0]:
```
# https://stackoverflow.com/a/47091490/4084039
import re
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```python
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 [0]:

```python
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'no
t'
# <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', 'o
urs', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselve
s', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'it
s', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'th
is', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'h
ave', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between',
'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out',
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'h
ow', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 's
o', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should',
"should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't",
'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "is
n't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',
 "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [13]:

```
# Combining all the above stundents
from tqdm import tqdm
from bs4 import BeautifulSoup
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%|██████████| 46072/46072 [00:24<00:00, 1886.15it/s]
```

In [29]:

```
preprocessed_reviews[1500]
```

Out[29]:

```
'orange lemon peels make tea hippy despite initial oohing ahing pretty
blue flowers regrettable purchase hoping stronger bergamot component tw
inings earl grey instead got something seems herbal blech disagree posi
tive reviews'
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [0]:
```
final["Clean_text"] = preprocessed_reviews
```

[3.2] Preprocessing Review Summary

In [0]:
```
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

In [0]:
```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(preprocessed_reviews)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

final_counts = count_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_counts))
print("the shape of out text BOW vectorizer ",final_counts.get_shape())
print("the number of unique words ", final_counts.get_shape()[1])
```

```
some feature names  ['aa', 'aaa', 'aaaa', 'aaaaa', 'aaaaaaaaaaaa', 'aaa
aaaaaaaaaaaa', 'aaaaaaahhhhhh', 'aaaaaawwwwwwwwww', 'aaaaah', 'aaaand']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (46071, 39364)
the number of unique words  39364
```

[4.2] Bi-Grams and n-Grams.

In [0]:
```
#bi-gram, tri-gram and n-gram
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
#removing stop words like "not" should be avoided before building n-grams
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_bigram_counts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.3] TF-IDF

In [0]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()[0:10])
print('='*50)

final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf.get_shape()[1])
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
some sample features(unique words in the corpus) ['ability', 'able', 'a
ble find', 'able get', 'absolute', 'absolutely', 'absolutely deliciou
s', 'absolutely love', 'absolutely no', 'according']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.4] Word2Vec

In [0]:

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

In [0]:

```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as val
ues
# To use this code-snippet, download "GoogleNews-vectors-negative300.bi
n"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edi
t
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17
SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```python
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors
-negative300.bin', binary=True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_trai
n_w2v = True, to train your own w2v ")
```

```
[('excellent', 0.9928812980651855), ('snack', 0.9926465749740601), ('ob
vious', 0.9926168918609619), ('wonderful', 0.992343544960022), ('licori
ce', 0.9922345876693726), ('chewy', 0.9922112226486206), ('tasting', 0.
9920798540115356), ('overall', 0.9918702244758606), ('alternative', 0.9
917284250259399), ('think', 0.9916505813598633)]
==================================================
[('varieties', 0.9994438886642456), ('oatmeal', 0.9994245767593384),
('sticks', 0.9993307590484619), ('awful', 0.9993144869804382), ('clea
r', 0.9993140697479248), ('experience', 0.999302864074707), ('peanuts',
0.9993011951446533), ('comes', 0.9992982149124146), ('yes', 0.999292373
6572266), ('choice', 0.999289870262146)]
```

In [0]:

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

```
number of words that occured minimum 5 times  3817
sample words  ['tuna', 'charger', 'minimal', 'manufacturer', 'disguste
d', 'confection', 'expensive', 'packed', 'holidays', 'tolerate', 'stick
er', 'manner', 'snacking', 'approximately', 'oak', 'rotate', 'chew', 's
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
andwich', 'may', 'takes', 'states', 'apricot', 'scratching', 'golden',
'upgraded', 'fork', 'tip', 'apart', 'carrot', 'scrambled', 'thrived',
'safely', 'compromise', 'duncan', 'waited', 'cakes', 'sweetened', 'salt
y', 'talking', 'fiber', 'teaspoon', 'lime', 'spring', 'leaves', 'blis
s', 'baronet', 'trip', 'pocket', 'savings', 'el']
```

[4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

In [0]:

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in
 this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u might need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    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(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|██████████████████████████████████████████| 4986/4986 [00:14<00:00, 33
7.09it/s]
```

```
4986
50
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

[4.4.1.2] TFIDF weighted W2v

In [0]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(preprocessed_reviews)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [0]:
```
# 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 ce
ll_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is st
ored in this list
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/r
eview
    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%|██████████| 4986/4986 [00:52<00:00, 9
5.62it/s]
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

[5] Assignment 3: KNN

1. **Apply Knn(brute force version) 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. **Apply Knn(kd tree version) on these feature sets**
    NOTE: sklearn implementation of kd-tree accepts only dense matrices, you need to convert the sparse matrices of CountVectorizer/TfidfVectorizer into dense matices. You can convert sparse matrices to dense using .toarray() attribute. For more information please visit this link

    - SET 5:Review text, preprocessed one converted into vectors using (BOW) but with restriction on maximum features generated.

    ```
    count_vect = CountVectorizer(min_df=10, max_features=500)
    count_vect.fit(preprocessed_reviews)
    ```

    - SET 6:Review text, preprocessed one converted into vectors using (TFIDF) but with restriction on maximum features generated.

    ```
    tf_idf_vect = TfidfVectorizer(min_df=10, max_features=500)
    tf_idf_vect.fit(preprocessed_reviews)
    ```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

- SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
- SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)

3. **The hyper paramter tuning(find best K)**

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

4. **Representation of results**

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

5. **Conclusion**

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

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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.

In [0]:
```
# short final data based on time
final = final.sort_values('Time', axis=0, ascending=True, inplace=False
, kind='quicksort', na_position='last')
```

In [0]:
```
#split data in train, test and cv before using it to avoid data leakage
from sklearn.model_selection import train_test_split

X = final['Clean_text']
y = final['Score']

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=.5,rando
m_state=0)
X_cv,X_test,y_cv,y_test = train_test_split(X_test,y_test,test_size=.5,r
andom_state=0)
```

In [17]:
```
print("X_train shape: ",X_train.shape)
print("y_train shape: ",X_train.shape)
print("X_test shape: ",X_test.shape)
print("y_test shape: ",y_test.shape)
print("X_cv shape: ",X_cv.shape)
print("y_cv shape: ",y_cv.shape)
```

```
X_train shape:  (23036,)
y_train shape:  (23036,)
X_test shape:  (11518,)
y_test shape:  (11518,)
X_cv shape:  (11518,)
y_cv shape:  (11518,)
```

[5.1] Applying KNN brute force

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

In [0]:
```
# Please write all the code with proper documentation
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix

#function to find an optimal value of k,AUC,ROC,confusion matrix
def KNN_model(x_train,x_cv,x_test,y_train,y_cv,y_test,algo='brute'):

    neighbor = list(range(1,25,2))
    cv_score = [] #list to store values of k
    pred_cv = []
    pred_train = []
    max_k = 0
    max_auc = -1
    for k in tqdm(neighbor):
        knn = KNeighborsClassifier(n_neighbors = k,algorithm=algo)
        knn.fit(x_train,y_train)
        scores = cross_val_score(knn,x_train,y_train,cv=10,scoring='roc
_auc')
        cv_score.append(scores.mean())


        prob_cv = knn.predict_proba(x_cv)
        prob_tr = knn.predict_proba(x_train)
        prob_cv = prob_cv[:,1]
        prob_tr = prob_tr[:,1]
        auc_cv = roc_auc_score(y_cv,prob_cv)
        auc_tr = roc_auc_score(y_train,prob_tr)
        pred_cv.append(auc_cv)
        pred_train.append(auc_tr)
        if max_auc < auc_cv:
            max_auc = auc_cv
            max_k = k

    #AUC curve
    plt.plot(neighbor,pred_cv,'r',label='CV')
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    plt.plot(neighbor,pred_train,'g',label='train')
    plt.legend(loc='upper right')
    plt.title('K vs AUC Score')
    plt.ylabel('AUC Score')
    plt.xlabel('K')
    plt.show()

    #ROC curve
    fpr_tr,tpr_tr,thres_tr = roc_curve(y_train,prob_train)
    fpr,tpr,thres = roc_curve(y_test,prob_test)
    plt.plot([0,0],[1,1],linestyle='--')
    plt.plot(fpr,tpr,'r',marker='.',label='test')
    plt.plot(fpr_tr,tpr_tr,'b',marker='.',label='train')
    plt.legend(loc='upper right')
    plt.title("ROC curve")
    plt.show()
    print("The optimal value of k is {}".format(max_k))

    #error rate
    error = [1- x for x in cv_score]

    #choose best k with lowest value of error
    opt_k = neighbor[error.index(min(error))]
    print("optimal k: ",opt_k)

def test(x_train,x_cv,x_test,y_train,y_cv,y_test,max_k,algo='brute'):
    knn = KNeighborsClassifier(n_neighbors=max_k,algorithm=algo)
    knn.fit(x_train,y_train)
    prob_test = knn.predict_proba(x_test)
    prob_test = prob_test[:,1]

    prob_train = knn.predict_proba(x_train)
    prob_train = prob_train[:,1]
    print("AUC Score: {}".format(roc_auc_score(y_test,prob_test)))



    #confusion matrix fortrain and test
    print("Confusion matrix for train data")
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    predict_tr = knn.predict(x_train)
    confu_metrix_(y_train,predict_tr)

    print("Confusion matrix for test data")
    predict_te = knn.predict(x_test)
    confu_metrix_(y_test,predict_te)


def confu_metrix_(y,predict):
    confu_metrix = confusion_matrix(y,predict)
    confu_df = pd.DataFrame(confu_metrix,index=["-ve","+ve"],columns=[
"-ve","+ve"])
    sns.heatmap(confu_df,annot=True,fmt='d',cmap='viridis')
    plt.title("Confusion matrix")
    plt.xlabel("predicted label")
    plt.ylabel("True label")
    plt.show()
```

In [0]:

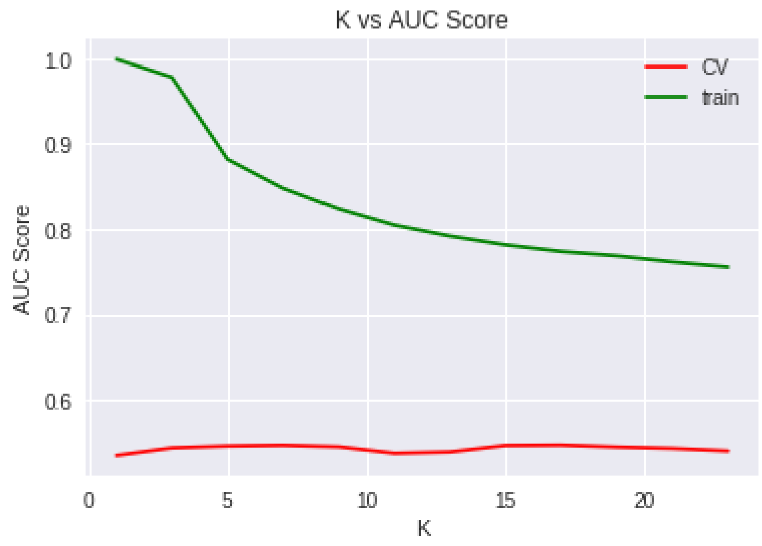
```
#BOW
count_vec = CountVectorizer()
bow_train = count_vec.fit_transform(X_train)
bow_test = count_vec.transform(X_test)
bow_cv = count_vec.transform(X_cv)

#normalize
from sklearn import preprocessing
bow_train=preprocessing.normalize(bow_train)
bow_cv=preprocessing.normalize(bow_cv)
bow_test=preprocessing.normalize(bow_test)
```

In [22]:

```
KNN_model(bow_train,bow_cv,bow_test,y_train,y_cv,y_test,algo='brute')
```

```
100%|██████████| 12/12 [11:14<00:00, 57.80s/it]
```

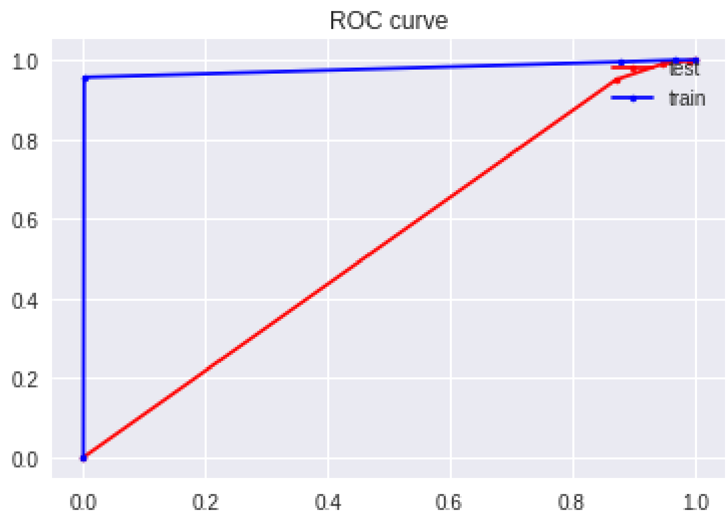
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



optimal k: 3

In [23]:
```
test(bow_train,bow_cv,bow_test,y_train,y_cv,y_test,3,algo='brute')
```

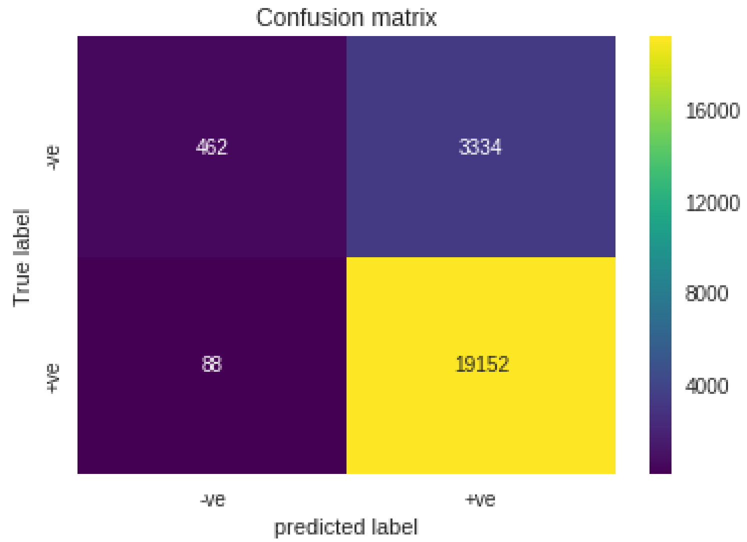
AUC Score: 0.5398782872672628



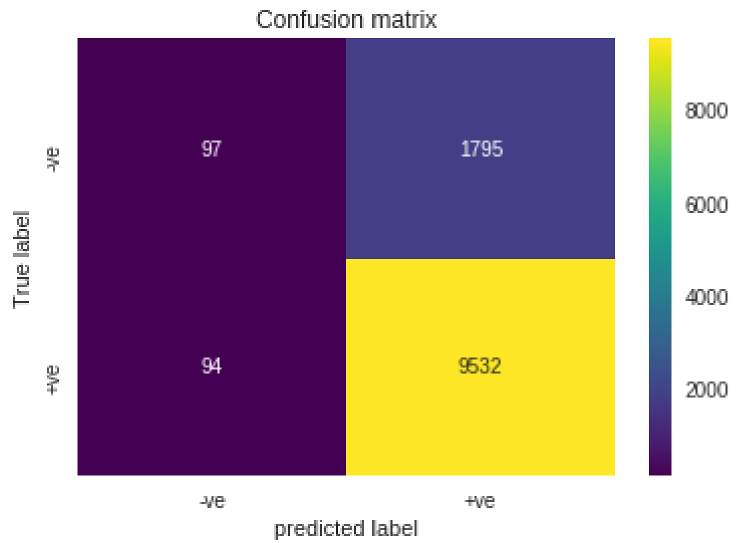
The optimal value of k is 3

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Confusion matrix for train data



Confusion matrix for test data



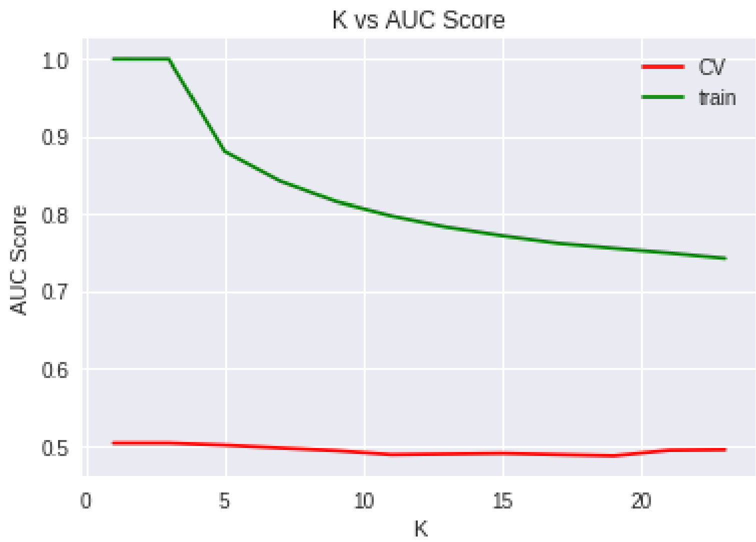
[5.1.2] Applying KNN brute force on TFIDF, SET 2

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [0]:
```
#TFIDF
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
tf_idf_train = preprocessing.normalize(tf_idf_vect.fit_transform(X_trai
n))
tf_idf_test = preprocessing.normalize(tf_idf_vect.transform(X_test))
tf_idf_cv = preprocessing.normalize(tf_idf_vect.transform(X_cv))
```

In [25]:
```
KNN_model(tf_idf_train,tf_idf_cv,tf_idf_test,y_train,y_cv,y_test,algo=
'brute')
```

```
100%|██████████| 12/12 [11:48<00:00, 60.48s/it]
```

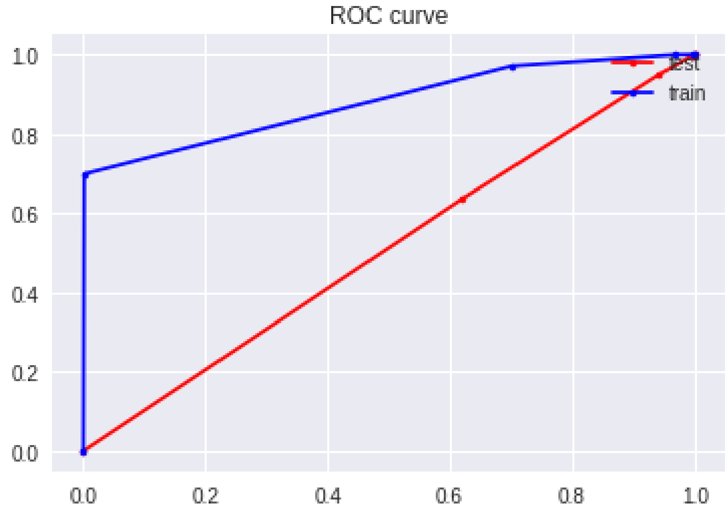



```
optimal k:  7
```

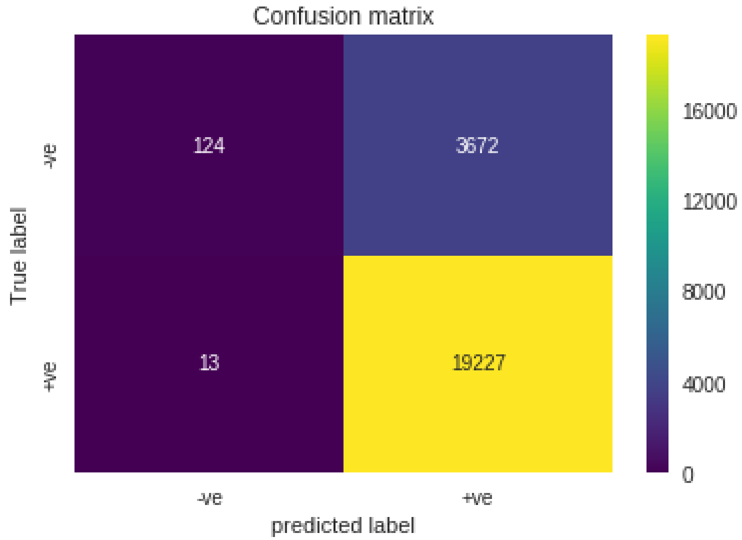
In [29]:
```
test(tf_idf_train,tf_idf_cv,tf_idf_test,y_train,y_cv,y_test,5,algo='bru
te')
```

```
AUC Score: 0.5090281386431832
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

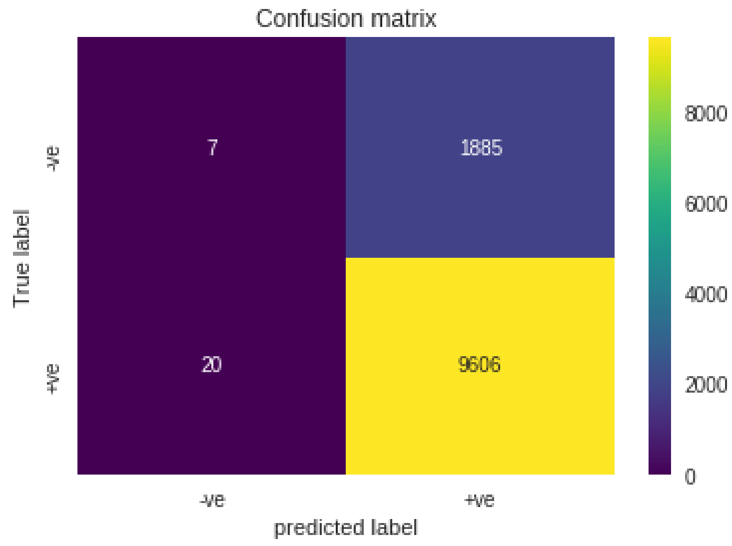



```
The optimal value of k is 5
Confusion matrix for train data
```




```
Confusion matrix for test data
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



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

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

In [31]:
```
is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors
-negative300.bin', binary=True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_trai
n_w2v = True, to train your own w2v ")
```

```
[('amazing', 0.847733736038208), ('awesome', 0.8419243693351746), ('exc
ellent', 0.809135377407074), ('wonderful', 0.7993621230125427), ('goo
d', 0.7772060632705688), ('fantastic', 0.7738646864891052), ('perfect',
0.7396727204322815), ('decent', 0.7238825559616089), ('satisfied', 0.71
47225737571716), ('terrific', 0.6963334083557129)]
==================================================
[('best', 0.811369776725769), ('ever', 0.7454642653465271), ('eaten',
0.7256593704223633), ('nastiest', 0.695431113243103), ('healthiest', 0.
695084273815155), ('hooked', 0.6913463473320007), ('tastiest', 0.676244
9145317078), ('greatest', 0.667126476764679), ('closest', 0.66476655006
40869), ('addicted', 0.66422438621521)]
```

In [32]:

```
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  9123
```

In [0]:

```
def vectorize_W2V(data):
    sent_vectors = []; # the avg-w2v for each sentence/review is stored
 in this list
    for sent in tqdm(data): # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length 5
0, you might need to change this to 300 if you use google's w2v
        cnt_words =0; # num of words with a valid vector in the sentenc
e/review
        for word in sent: # for each word in a review/sentence
            if word in w2v_words:
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

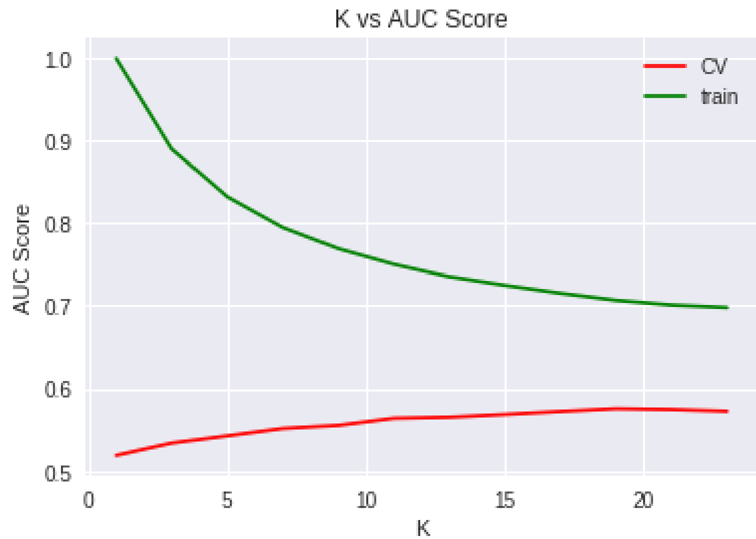
```
                vec = w2v_model.wv[word]
                sent_vec += vec
                cnt_words += 1
        if cnt_words != 0:
            sent_vec /= cnt_words
        sent_vectors.append(sent_vec)
    return sent_vectors
```

In [34]:
```
# vectorize all train,test and cv data
X_train_avgw2v = vectorize_W2V(X_train)
X_cv_avgw2v = vectorize_W2V(X_cv)
X_test_avgw2v = vectorize_W2V(X_test)
```

```
100%|██████████| 23036/23036 [15:30<00:00, 24.75it/s]
100%|██████████| 11518/11518 [07:40<00:00, 25.02it/s]
100%|██████████| 11518/11518 [07:49<00:00, 24.51it/s]
```

In [35]:
```
KNN_model(X_train_avgw2v,X_cv_avgw2v,X_test_avgw2v,y_train,y_cv,y_test,
algo='brute')
```

```
100%|██████████| 12/12 [05:39<00:00, 29.43s/it]
```



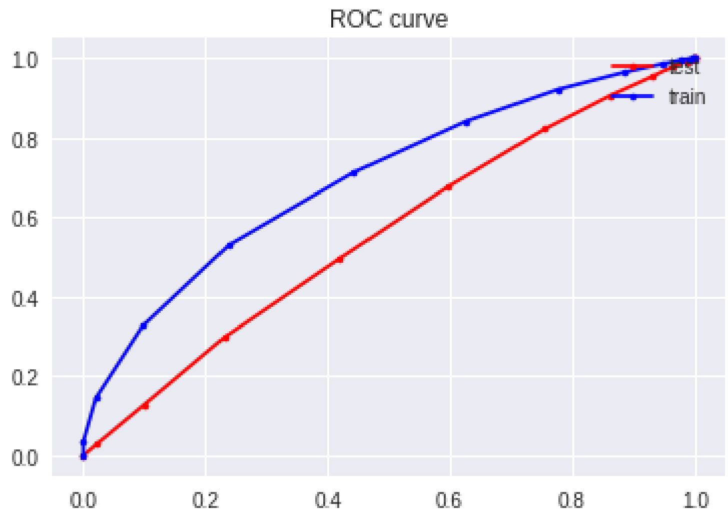

```
optimal k: 23
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

optimal k: 23

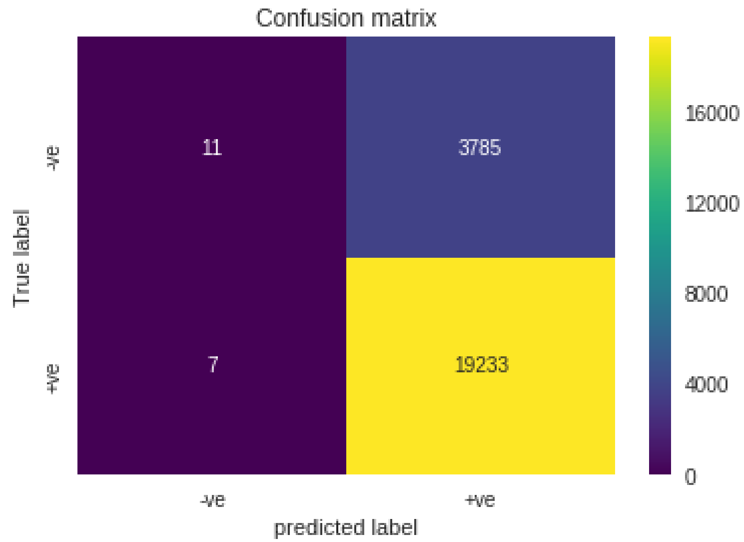
In [36]:
```
test(X_train_avgw2v,X_cv_avgw2v,X_test_avgw2v,y_train,y_cv,y_test,23,al
go='brute')
```

AUC Score: 0.5557054504427534

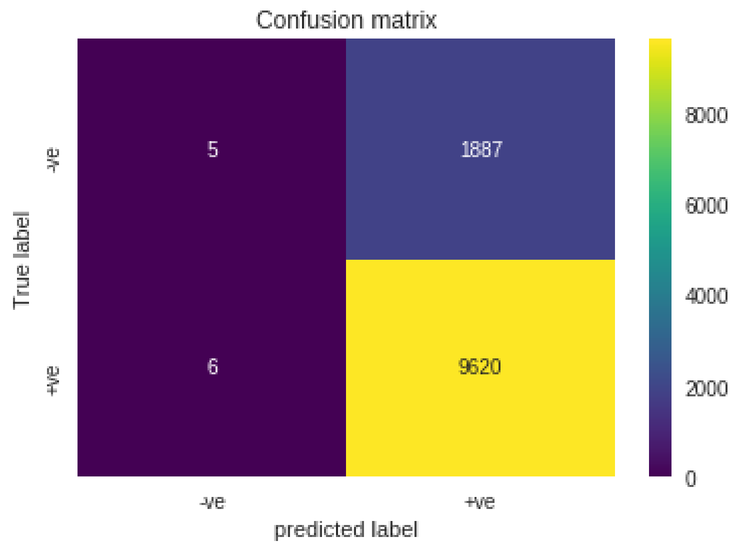


The optimal value of k is 23
Confusion matrix for train data

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




```
Confusion matrix for test data
```



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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [0]:
```
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_train)
# 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 [0]:
```
# 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
def vectorizer_W2V_tfidf(data):
    tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
    row=0;
    for sent in tqdm(data): # 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
    return tfidf_sent_vectors
```

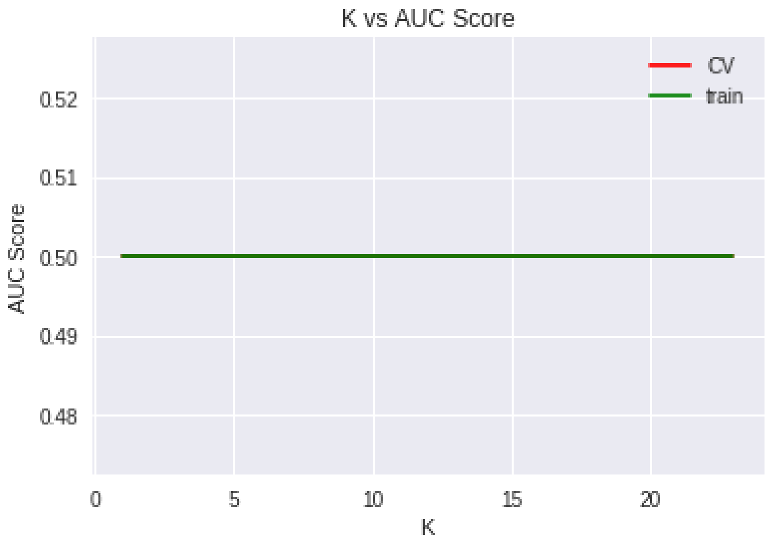
In [39]:
```
# vectorize all train, test and cv data
X_train_tfidfw2v = vectorizer_W2V_tfidf(X_train)
X_cv_tfidfw2v = vectorizer_W2V_tfidf(X_cv)
X_test_tfidfw2v = vectorizer_W2V_tfidf(X_test)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
100%|██████████| 23036/23036 [38:40<00:00,  9.93it/s]
100%|██████████| 11518/11518 [19:06<00:00, 11.48it/s]
100%|██████████| 11518/11518 [19:35<00:00,  9.30it/s]
```

In [40]:
```
KNN_model(X_train_tfidfw2v,X_cv_tfidfw2v,X_test_tfidfw2v,y_train,y_cv,y
_test,algo='brute')
```

```
100%|██████████| 12/12 [04:25<00:00, 22.10s/it]
```

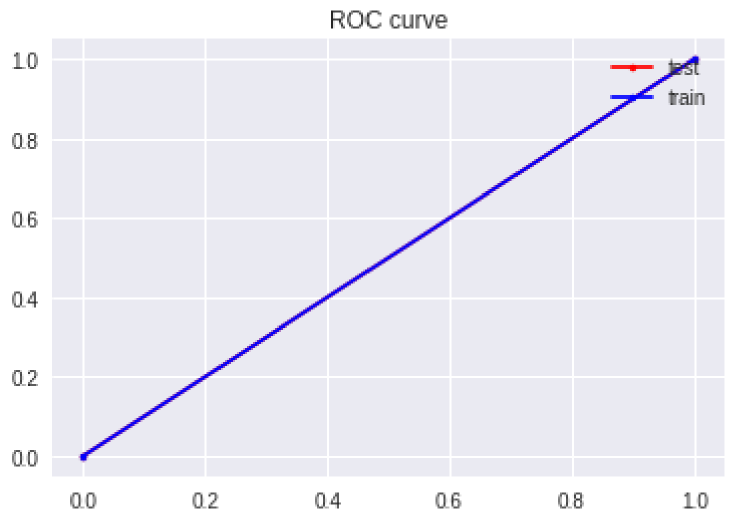



```
optimal k:  1
```

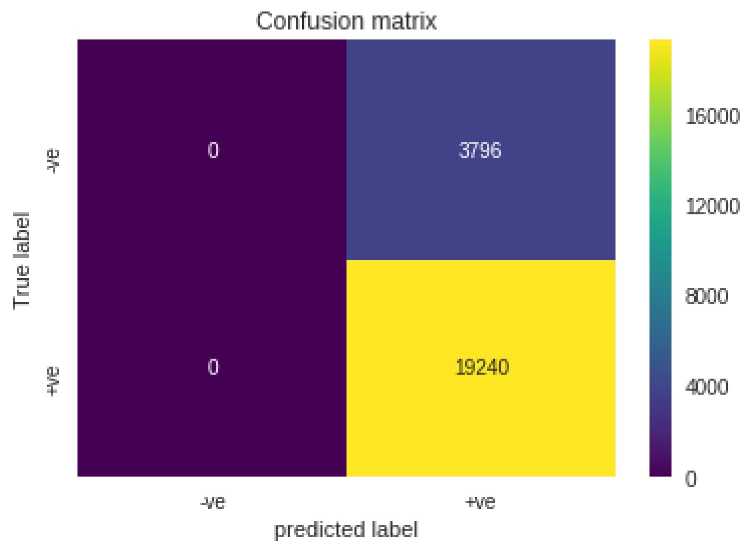
In [44]:
```
test(X_train_tfidfw2v,X_cv_tfidfw2v,X_test_tfidfw2v,y_train,y_cv,y_test
,1,algo='brute')
```

```
AUC Score: 0.5
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

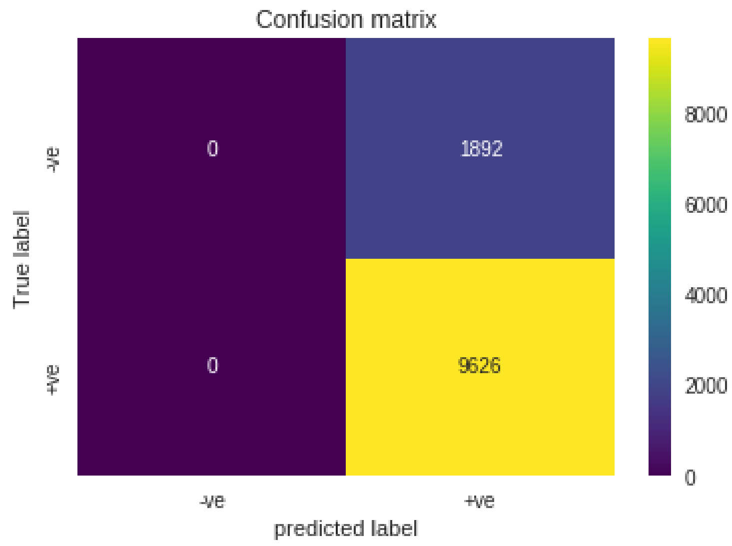



```
The optimal value of k is 1
Confusion matrix for train data
```




```
Confusion matrix for test data
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



[5.2] Applying KNN kd-tree

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

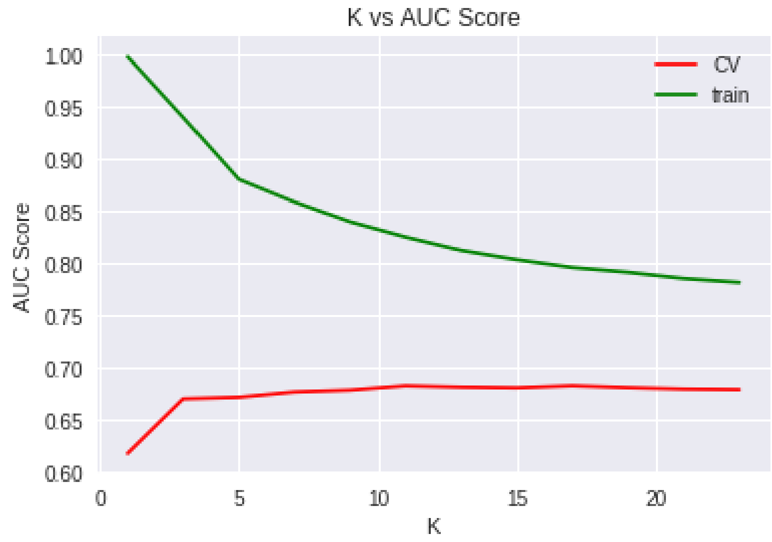
In [0]:

```
count_vect = CountVectorizer(min_df=10, max_features=500)
bow_train_kd = count_vect.fit_transform(X_train)
bow_test_kd = count_vect.transform(X_test)
bow_cv_kd = count_vect.transform(X_cv)

#normalize
from sklearn import preprocessing
bow_train_kd=preprocessing.normalize(bow_train_kd)
bow_cv_kd=preprocessing.normalize(bow_cv_kd)
bow_test_kd=preprocessing.normalize(bow_test_kd)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
In [46]: KNN_model(bow_train_kd,bow_cv_kd,bow_test_kd,y_train,y_cv,y_test,algo=
'kd_tree')
```

```
100%|██████████| 12/12 [11:22<00:00, 58.18s/it]
```

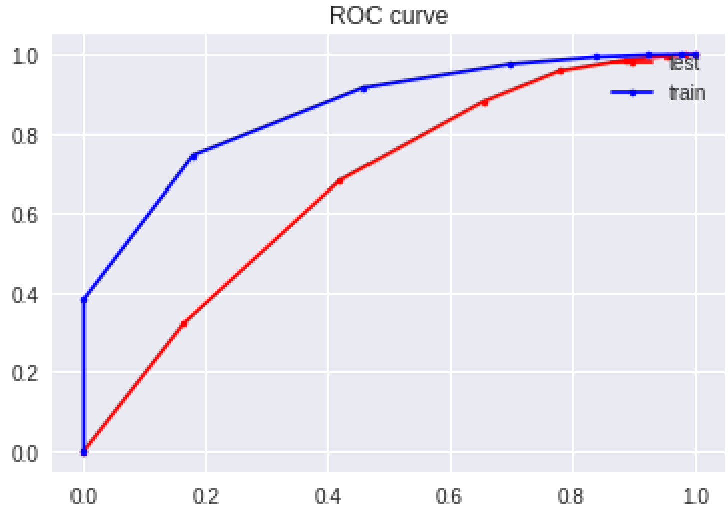



```
optimal k:  7
```

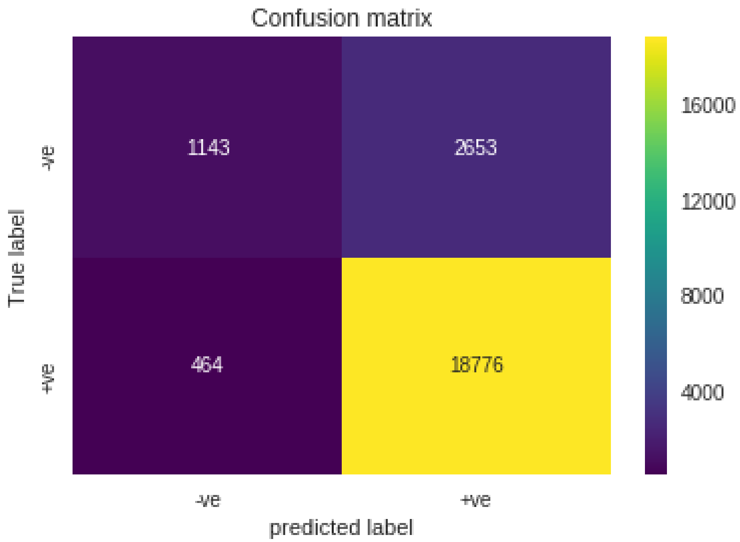
```
In [48]: test(bow_train_kd,bow_cv_kd,bow_test_kd,y_train,y_cv,y_test,7,algo='kd_
tree')
```

```
AUC Score: 0.6703097813840159
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

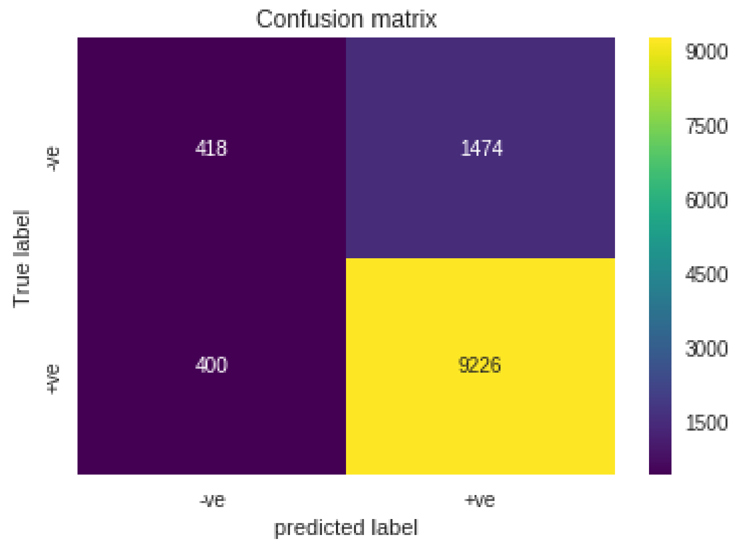



```
The optimal value of k is 7
Confusion matrix for train data
```




```
Confusion matrix for test data
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API
PDFCROWD



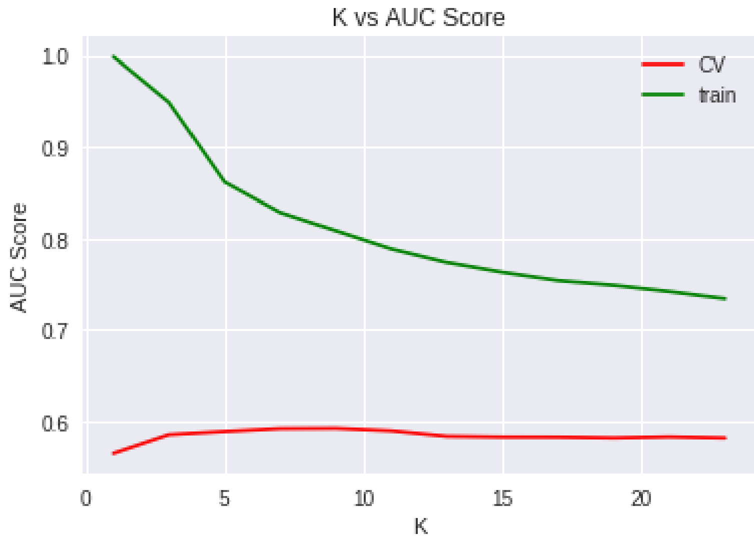
[5.2.2] Applying KNN kd-tree on TFIDF, SET 6

In [0]:
```
#TFIDF
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10,max_features
=500)
tf_idf_train_kd = preprocessing.normalize(tf_idf_vect.fit_transform(X_t
rain))
tf_idf_test_kd = preprocessing.normalize(tf_idf_vect.transform(X_test))
tf_idf_cv_kd = preprocessing.normalize(tf_idf_vect.transform(X_cv))
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [50]:
```
KNN_model(tf_idf_train_kd,tf_idf_cv_kd,tf_idf_test_kd,y_train,y_cv,y_te
st,algo='kd_tree')
```

```
100%|██████████| 12/12 [11:15<00:00, 57.55s/it]
```

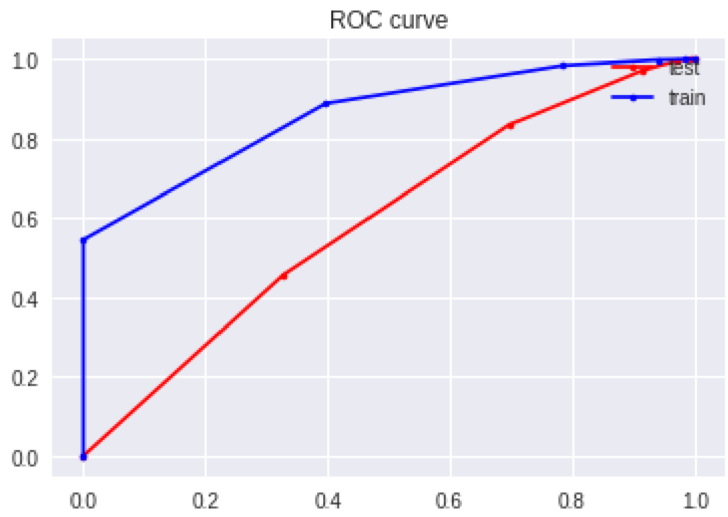



```
optimal k:  5
```

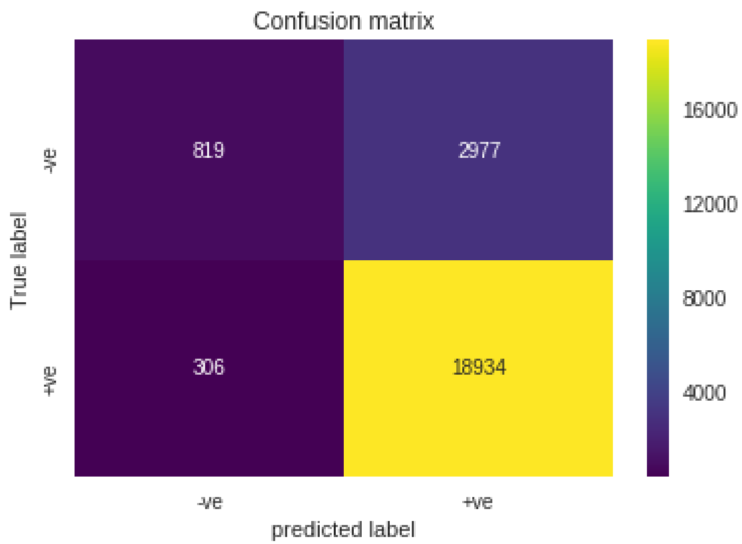
In [51]:
```
test(tf_idf_train_kd,tf_idf_cv_kd,tf_idf_test_kd,y_train,y_cv,y_test,5,
algo='kd_tree')
```

```
AUC Score: 0.5942960704996906
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

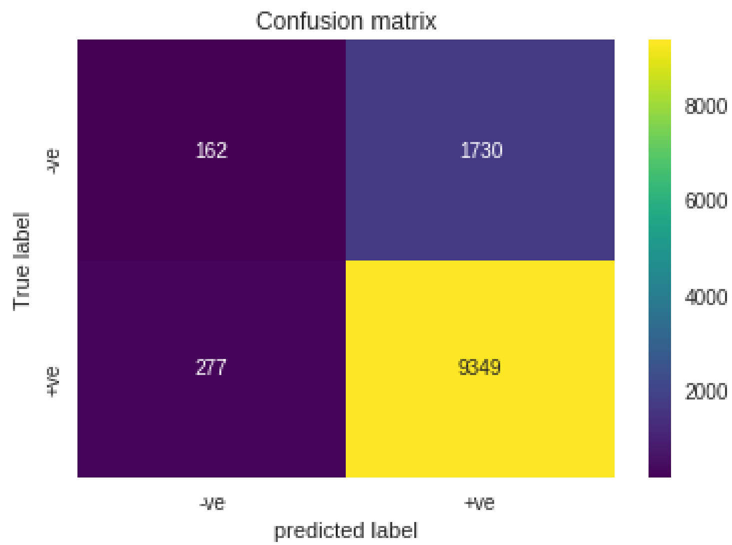



```
The optimal value of k is 5
Confusion matrix for train data
```




```
Confusion matrix for test data
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

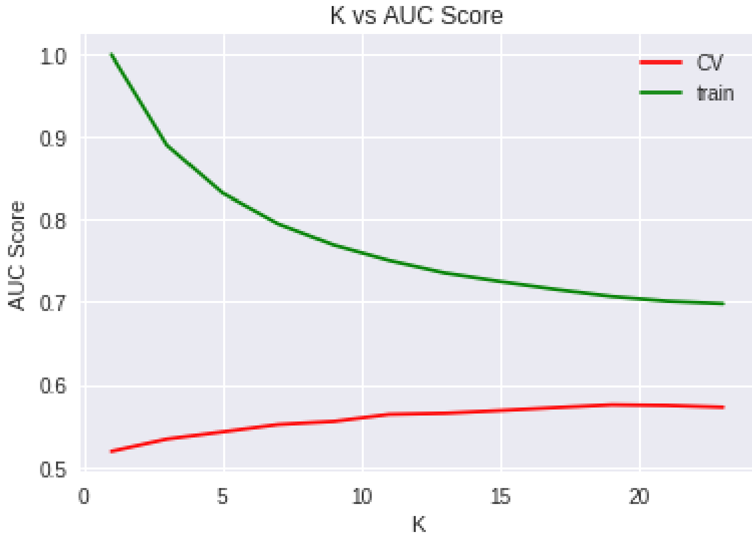


[5.2.3] Applying KNN kd-tree on AVG W2V, SET 3

In [52]:
```
KNN_model(X_train_avgw2v,X_cv_avgw2v,X_test_avgw2v,y_train,y_cv,y_test,
algo='kd_tree')
```

```
100%|██████████| 12/12 [05:32<00:00, 31.74s/it]
```

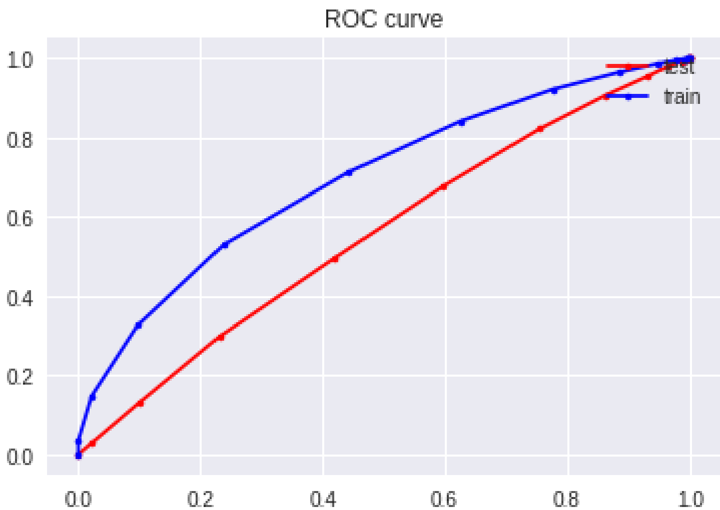
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



optimal k: 23

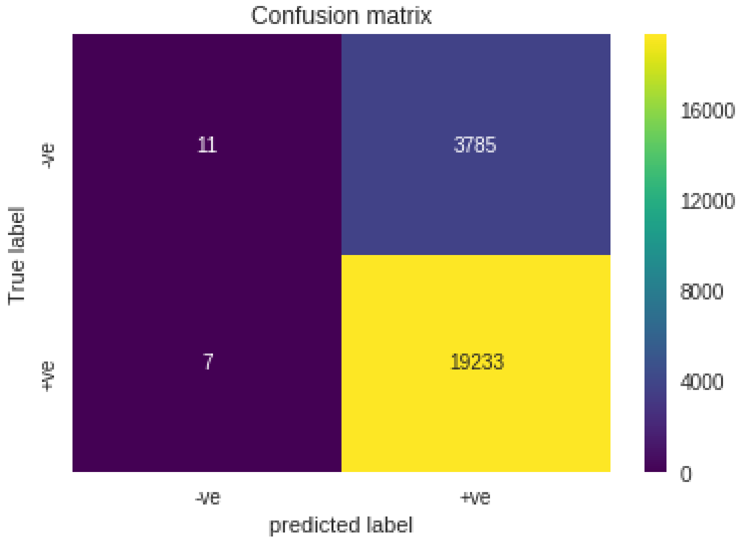
In [53]:
```
test(X_train_avgw2v,X_cv_avgw2v,X_test_avgw2v,y_train,y_cv,y_test,23,al
go='kd_tree')
```

AUC Score: 0.5559699681403739

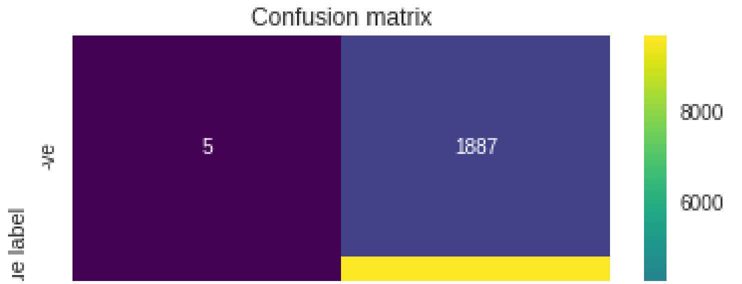


Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

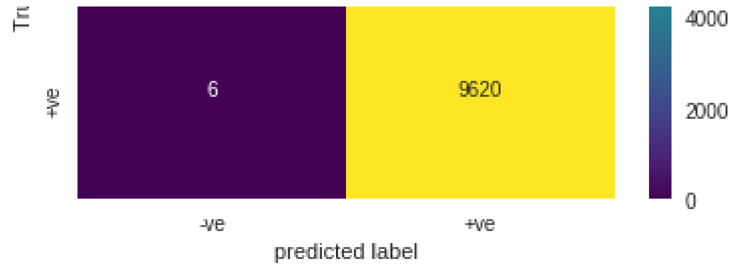
```
The optimal value of k is 23
Confusion matrix for train data
```




```
Confusion matrix for test data
```



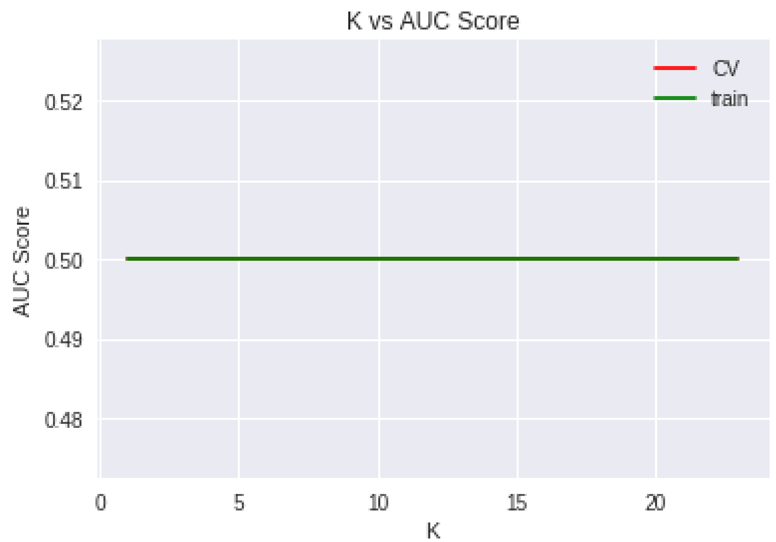
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



[5.2.4] Applying KNN kd-tree on TFIDF W2V, SET 4

In [54]:
```
KNN_model(X_train_tfidfw2v,X_cv_tfidfw2v,X_test_tfidfw2v,y_train,y_cv,y
_test,algo='kd_tree')
```

```
100%|██████████| 12/12 [23:26<00:00, 117.23s/it]
```

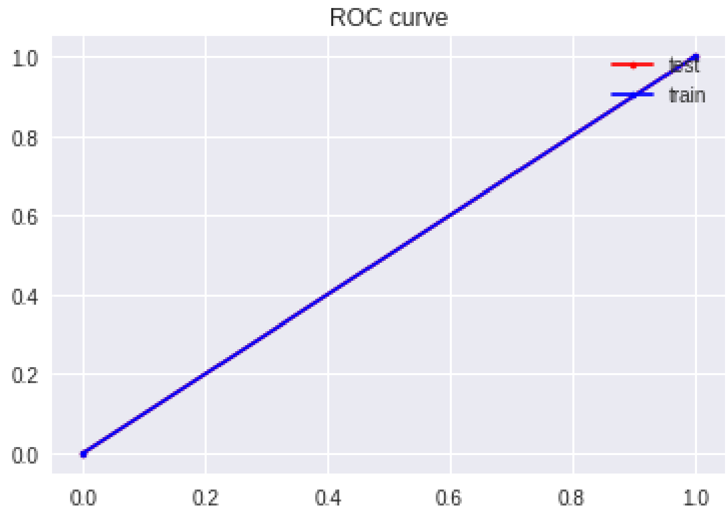



```
optimal k:  1
```

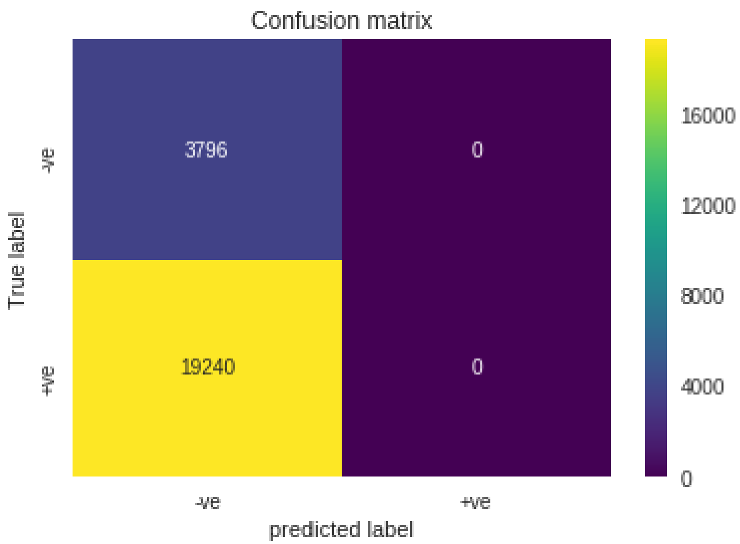
In [55]:
```
test(X_train_tfidfw2v,X_cv_tfidfw2v,X_test_tfidfw2v,y_train,y_cv,y_test
,1,algo='kd_tree')
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
AUC Score: 0.5
```

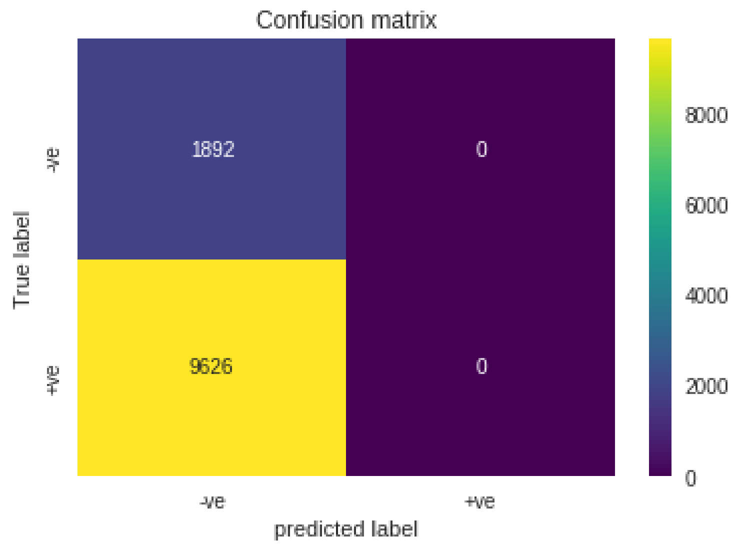



```
The optimal value of k is 1
Confusion matrix for train data
```




```
Confusion matrix for test data
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



[6] Conclusions

In [0]:
```
# Please compare all your models using Prettytable library
df = pd.DataFrame({"Model":["Brute with BOW","Brute with TFIDF","Brute
 with Avg W2V","Brute with TFIDF W2V","KD_Tree with BOW","KD_Tree with
 TFIDF","KD_Tree with Avg W2V","KD_Tree with TFIDF W2V"],
                   "Hyper parameter(K)":[3,7,23,1,7,5,23,1],
                   "AUC":[0.5398782872672628,
                          0.5090281386431832,
                          0.5557054504427534,
                          0.5,
                          0.6703097813840159,
                          0.5942960704996906,
                          0.5559699681403739,
                          0.5]}
                  ,columns=["Model","Hyper parameter(K)","AUC"])
```

In [57]:
```
df.sort_values(by="AUC",ascending=False)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Out[57]:

	Model	Hyper parameter(K)	AUC
4	KD_Tree with BOW	7	0.670310
5	KD_Tree with TFIDF	5	0.594296
6	KD_Tree with Avg W2V	23	0.555970
2	Brute with Avg W2V	23	0.555705
0	Brute with BOW	3	0.539878
1	Brute with TFIDF	7	0.509028
3	Brute with TFIDF W2V	1	0.500000
7	KD_Tree with TFIDF W2V	1	0.500000

From above table we can say that all model here performed nearly same as they all have similar AUC scores but according to AUC scores kd_tree performes batter than brute force.

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD