In [2]:

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


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

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

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

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

from tqdm import tqdm
import os
```

In [2]:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?clien
t_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.co
m&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2
F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.co
m%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.re
adonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&respon
se_type=code (https://accounts.google.com/o/oauth2/auth?client_id=9473189898
03-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=
urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googleap
is.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%
20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3
A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code)

Enter your authorization code:
··········
Mounted at /content/gdrive
```

In [4]:

```
#mounting the dataset from drive

#connecting to sqlite db
con = sqlite3.connect('/content/gdrive/My Drive/Colab Notebooks/Assignment 3/database.sqlit

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

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

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

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

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

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

Out[4]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
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

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 [6]:

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

```
(80668, 7)
```

Out[6]:

	UserId	ProductId	ProfileName	Time	Score	Text	CO
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 [7]:

```
# Removing duplicate reviews
final=filtered_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='fi
print(final.shape)
```

```
(364173, 10)
```

In [8]:

```
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[8]:

```
69.25890143662969
```

In [0]:

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

In [10]:

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

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

```
(364171, 10)
```

Out[10]:

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

In [0]:

```
final["cleanReview"] = final["Summary"].map(str) + ". " + final["Text"]
```

In [12]:

```
final['cleanReview'].head()
```

Out[12]:

```
0    Good Quality Dog Food. I have bought several o...
1    Not as Advertised. Product arrived labeled as ...
2    "Delight" says it all. This is a confection th...
3    Cough Medicine. If you are looking for the sec...
4    Great taffy. Great taffy at a great price.  Th...
Name: cleanReview, dtype: object
```

In [13]:

```
len(final['cleanReview'].str.split().str.len())
```

Out[13]:

```
364171
```

In [14]:

```
final['cleanReview'][0]
```

Out[14]:

```
'Good Quality Dog Food. I have bought several of the Vitality canned dog foo
d products and have found them all to be of good quality. The product looks
more like a stew than a processed meat and it smells better. My Labrador is
finicky and she appreciates this product better than  most.'
```

In [15]:

```
#remove urls from text python
from tqdm import tqdm
lst = []
removed_urls_list = []
for text in tqdm(final['cleanReview']):
  removed_urls_text = re.sub(r"http\S+", "", text)
  lst.append(removed_urls_text)
```

```
100%|██████████| 364171/364171 [00:00<00:00, 441731.62it/s]
```

In [16]:

```
#remove urls from text python
removed_urls_list = []
for text in tqdm(lst):
  removed_urls_text = re.sub(r"http\S+", "", text)
  removed_urls_list.append(removed_urls_text)
```

```
100%|██████████| 364171/364171 [00:00<00:00, 467025.67it/s]
```

In [17]:

```
from bs4 import BeautifulSoup
text_lst = []
for text in tqdm(removed_urls_list):
  soup = BeautifulSoup(text, 'lxml')
  text = soup.get_text()
  text_lst.append(text)
# print(text)
# print("="*50)
```

```
100%|██████████| 364171/364171 [01:35<00:00, 3817.94it/s]
```

In [18]:

```
print(len(final['Text']))
```

```
364171
```

In [0]:

```
# 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 [20]:

```
decat_lst = []
for decat_text in tqdm(text_lst):
  text = decontracted(decat_text)
  decat_lst.append(text)
```

```
100%|██████████| 364171/364171 [00:05<00:00, 65561.20it/s]
```

In [21]:

```
strip_list = []
for to_strip in tqdm(decat_lst):
  text = re.sub("\S*\d\S*", "", to_strip).strip()
  strip_list.append(text)
```

```
100%|██████████| 364171/364171 [00:15<00:00, 23402.57it/s]
```

In [22]:

```
spatial_list = []
for to_spatial in tqdm(strip_list):
  text = re.sub('[^A-Za-z0-9]+', ' ', to_spatial)
  spatial_list.append(text)
```

```
100%|██████████| 364171/364171 [00:09<00:00, 36822.48it/s]
```

In [0]:

```
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 [24]:

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

```
100%|██████████| 364171/364171 [02:19<00:00, 2618.24it/s]
```

In [25]:

```
print(len(preprocessed_reviews))
preprocessed_reviews[-1]
```

```
364171
```

Out[25]:

```
'great honey satisfied product advertised use cereal raw vinegar general swe
etner'
```

In [0]:

```
final['cleanReview'] = preprocessed_reviews
```

In [27]:

```
print(len(final))
final.tail(5)
```

```
364171
```

Out[27]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator
525809	568450	B001EO7N10	A28KG5XORO54AY	Lettie D. Carter	0
525810	568451	B003S1WTCU	A3I8AFVPEE8KI5	R. Sawyer	0
525811	568452	B004I613EE	A121AA1GQV751Z	pksd "pk_007"	2
525812	568453	B004I613EE	A3IBEVCTXKNOH	Kathy A. Welch "katwel"	1
525813	568454	B001LR2CU2	A3LGQPJCZVL9UC	srfell17	0

In [28]:

```
final['lengthOfReview'] = final['cleanReview'].str.split().str.len()
final['lengthOfReview'].head()
```

Out[28]:

```
0    27
1    21
2    43
3    20
4    15
Name: lengthOfReview, dtype: int64
```

In [29]:

```
final['cleanReview'][0]
```

Out[29]:

```
'good quality dog food bought several vitality canned dog food products foun
d good quality product looks like stew processed meat smells better labrador
finicky appreciates product better'
```

In [30]:

```
final['cleanReview'][525809]
```

Out[30]:

```
'not without great sesame chicken good not better resturants eaten husband l
oved find recipes use'
```

In [3]:

```
dir_path = os.getcwd()
# conn = sqlite3.connect(os.path.join(dir_path, '/content/gdrive/My Drive/Colab Notebooks/A
conn = sqlite3.connect(os.path.join(dir_path, 'final.sqlite'))
# final.to_sql('Reviews', conn, if_exists='replace', index=False)
```

In [4]:

```
review_3 = pd.read_sql_query(""" SELECT count(*) FROM Reviews""", conn)
print(review_3)
```

```
   count(*)
0    364171
```

In [5]:

```
filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews""", conn)
```

In [6]:

```
filtered_data.shape
```

Out[6]:

```
(364171, 12)
```

In [7]:

```
filtered_data["Time"] = pd.to_datetime(filtered_data["Time"], unit = "s")
filtered_data = filtered_data.sort_values(by = "Time")
```

In [8]:

```
filtered_data.head(5)
```

Out[8]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator
117924	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0
117901	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2
298792	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0
169281	230285	B00004RYGX	A344SMIA5JECGM	Vincent P. Ross	1
298791	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0

In [9]:

```
print(len(filtered_data))
filtered_data.info()
filtered_data = filtered_data.head(100000)
print(len(filtered_data))
```

```
364171
<class 'pandas.core.frame.DataFrame'>
Int64Index: 364171 entries, 117924 to 107253
Data columns (total 12 columns):
Id                        364171 non-null int64
ProductId                 364171 non-null object
UserId                    364171 non-null object
ProfileName               364171 non-null object
HelpfulnessNumerator      364171 non-null int64
HelpfulnessDenominator    364171 non-null int64
Score                     364171 non-null int64
Time                      364171 non-null datetime64[ns]
Summary                   364171 non-null object
Text                      364171 non-null object
cleanReview               364171 non-null object
lengthOfReview            364171 non-null int64
dtypes: datetime64[ns](1), int64(5), object(6)
memory usage: 36.1+ MB
100000
```

In [10]:

```
filtered_data['Score'].value_counts()
```

Out[10]:

```
1    87729
0    12271
Name: Score, dtype: int64
```

In [11]:

```
X = filtered_data["cleanReview"]
print(print("shape of X:", X.head(5)))
y = filtered_data["Score"]
print("shape of y:", y.head(5))
```

```
shape of X: 117924    every book educational witty little book makes...
117901    whole series great way spend time child rememb...
298792    entertainingl funny beetlejuice well written m...
169281    modern day fairy tale twist rumplestiskin capt...
298791    fantastic beetlejuice excellent funny movie ke...
Name: cleanReview, dtype: object
None
shape of y: 117924    1
117901    1
298792    1
169281    1
298791    1
Name: Score, dtype: int64
```

In [12]:

```
len(filtered_data['lengthOfReview'])
```

Out[12]:

```
100000
```

In [13]:

```
X_train = X[0:60000]
Y_train = y[0:60000]
X_val = X[60000:80000]
Y_val = y[60000:80000]
X_test = X[80000:100000]
Y_test = y[80000:100000]
```

In [14]:

```
print(len(X_train), len(X_test), len(X_val))
print(len(Y_train), len(Y_test), len(Y_val))
```

```
60000 20000 20000
60000 20000 20000
```

[4.1] BAG OF WORDS

In [14]:

```
from sklearn.feature_extraction.text import CountVectorizer

count_vect = CountVectorizer()
X_train_vect = count_vect.fit_transform(X_train)
X_test_vect = count_vect.transform(X_test)
X_val_vect = count_vect.transform(X_val)
# BoW_dict = {'X_train_vect':X_train_vect, 'X_test_vect': X_test_vect, 'X_val_vect': X_val_
print(X_train_vect.shape)
```

```
(60000, 47535)
```

In [15]:

```
X_train_vect.shape
```

Out[15]:

```
(60000, 47535)
```

In [0]:

```
from scipy.sparse import hstack
# len_review = final['lengthOfReview'].to_sparse()
concat_data = hstack((X_train_vect,np.array(final['lengthOfReview'][0:60000])[:,None]))
concat_data_val = hstack((X_val_vect,np.array(final['lengthOfReview'][60000:80000])[:,None]
concat_data_test = hstack((X_test_vect,np.array(final['lengthOfReview'][80000:100000])[:,No
```

In [47]:

```
print(concat_data.shape)
print(concat_data_val.shape)
print(concat_data_test.shape)
```

```
(60000, 47536)
(20000, 47536)
(20000, 47536)
```

In [48]:

```
BoW_dict = {'X_train_vect':concat_data, 'X_test_vect': concat_data_test, 'X_val_vect': conc
print(BoW_dict['X_train_vect'].shape)
```

```
(60000, 47536)
```

In [0]:

```
import pickle
with open('/content/gdrive/My Drive/Colab Notebooks/Assignment 3/BoW.pkl', 'wb') as handle:
    pickle.dump(BoW_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

[4.3] TF-IDF

In [52]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
train_tf_idf = tf_idf_vect.fit_transform(X_train)
cv_tf_idf = tf_idf_vect.transform(X_val)
test_tf_idf = tf_idf_vect.transform(X_test)

print("the shape of out text TFIDF vectorizer ",train_tf_idf.get_shape())
print("the type of count vectorizer ",type(train_tf_idf))
print("the number of unique words including both unigrams and bigrams ", train_tf_idf.get_s
```

```
the shape of out text TFIDF vectorizer  (60000, 35873)
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the number of unique words including both unigrams and bigrams  35873
```

In [0]:

```
tfidf_concat_data_train = hstack((train_tf_idf,np.array(final['lengthOfReview'][0:60000])[:
tfidf_concat_data_val = hstack((cv_tf_idf,np.array(final['lengthOfReview'][60000:80000])[:,
tfidf_concat_data_test = hstack((test_tf_idf,np.array(final['lengthOfReview'][80000:100000]
```

In [0]:

```
tf_idf_dict = {'train_tf_idf': tfidf_concat_data_train, 'cv_tf_idf': tfidf_concat_data_val,
```

In [0]:

```
import pickle
with open('/content/gdrive/My Drive/Colab Notebooks/Assignment 3/tf_idf.pkl', 'wb') as hanc
    pickle.dump(tf_idf_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

[4.4] Word2Vec

In [43]:

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

In [44]:

```
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_sen,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', b
        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_train_w2v = True, to tra
```

```
[('excellent', 0.8281802535057068), ('terrific', 0.8193535804748535), ('fant
astic', 0.8092791438102722), ('awesome', 0.7818319797515869), ('wonderful',
0.7789075970649719), ('good', 0.7625725865364075), ('perfect', 0.69207787513
73291), ('fabulous', 0.6512437462806702), ('amazing', 0.6420807242393494),
('love', 0.6288779973983765)]
==================================================
[('greatest', 0.7680816054344177), ('best', 0.7392822504043579), ('tasties
t', 0.6441187858581543), ('awful', 0.6294796466827393), ('tasted', 0.6065288
782119751), ('seen', 0.6033897399902344), ('experienced', 0.600575029850006
1), ('closest', 0.5983313322067261), ('terrible', 0.597944974899292), ('wors
e', 0.5959557294845581)]
```

In [45]:

```
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  15289
sample words  ['sally', 'munched', 'discussed', 'microwaveable', 'spice', 's
tab', 'contribution', 'energizer', 'ceiling', 'snap', 'wonders', 'mir', 'cv
b', 'pantothenate', 'umph', 'high', 'powder', 'physician', 'wilton', 'canva
s', 'austria', 'colander', 'whiter', 'finland', 'cakey', 'touches', 'soy',
'ben', 'traditionally', 'bouquet', 'pant', 'scenes', 'bhaji', 'role', 'apply
ing', 'surprise', 'followed', 'retain', 'parboiled', 'regulated', 'flights',
'mojo', 'ferals', 'nitrites', 'clerk', 'eskimo', 'baxter', 'persons', 'drin
k', 'questions']
```

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

[4.4.1.1] Avg W2v

In [46]:

```
print(X_train[117924])
print(len(X_val))
print(len(X_test))
```

```
every book educational witty little book makes son laugh loud recite car dri
ving along always sing refrain learned whales india drooping roses love new
words book introduces silliness classic book willing bet son still able reci
te memory college
20000
20000
```

In [47]:

```
# average Word2Vec
# compute average word2vec for each review.
def avg_w2vec(sentences_received):
    sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sent in sentences_received: # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to
        cnt_words =0; # num of words with a valid vector in the sentence/review
        for word in sent: # for each word in a review/sentence
            if word in w2v_words:
                vec = w2v_model.wv[word]
                sent_vec += vec
                cnt_words += 1
        if cnt_words != 0:
            sent_vec /= cnt_words
        sent_vectors.append(sent_vec)

    print(len(sent_vectors))
    print(len(sent_vectors[0]))
    return sent_vectors
```

In [48]:

```
print(len([sent.split() for sent in X_test]))
```

```
20000
```

In [49]:

```
avg_w2v_train = avg_w2vec([sent.split() for sent in X_train])
avg_w2v_cv = avg_w2vec([sent.split() for sent in X_val])
avg_w2v_test = avg_w2vec([sent.split() for sent in X_test])
```

```
60000
50
20000
50
20000
50
```

In [50]:

```
print(len(avg_w2v_test))
```

```
20000
```

In [51]:

```
Avg_w2v_dict = {'X_train_avgw2v':avg_w2v_train, 'Y_train_avgw2v': Y_train,
                     'X_val_avgw2v': avg_w2v_cv, 'Y_val_avgw2v': Y_val,
                    'X_test_avgw2v': avg_w2v_test, 'Y_test_avgw2v': Y_test}
```

In [52]:

```
import pickle
with open('avg_w2v.pkl', 'wb') as handle:
    pickle.dump(Avg_w2v_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

[4.4.1.2] TFIDF weighted W2v

In [69]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
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 [70]:

```
# 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 tfidf_w2v(sentences_received):
    tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this lis
    row=0;
    for sent in sentences_received: # 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 [71]:

```
tfidf_w2v_train = tfidf_w2v([sent.split() for sent in X_train])
tfidf_w2v_cv = tfidf_w2v([sent.split() for sent in X_val])
tfidf_w2v_test = tfidf_w2v([sent.split() for sent in X_test])
```

In [72]:

```
print(len(tfidf_w2v_train))
print(len(X_val))
print(len(X_test))
```

```
60000
20000
20000
```

In [73]:

```
tfidf_w2v_dict = {'X_train_tfidfw2v':tfidf_w2v_train, 'Y_train_tfidfw2v': Y_train,
                     'X_val_tfidfw2v': tfidf_w2v_cv, 'Y_val_tfidfw2v': Y_val,
                    'X_test_tfidfw2v': tfidf_w2v_test, 'Y_test_tfidfw2v': Y_test}
```

In [74]:

```
with open('tfidf_w2v.pkl', 'wb') as handle:
    pickle.dump(tfidf_w2v_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

K - NN on Bow

In [0]:

```
import pickle
with open(r"/content/gdrive/My Drive/Colab Notebooks/Assignment 3/BoW.pkl", "rb") as input_
    BoW_dict = pickle.load(input_file)
```

In [67]:

```
from sklearn.neighbors import KNeighborsClassifier
from tqdm import tqdm

bow_auc_train = []
bow_auc_cv = []
bow_auc_test = []

for k_value in tqdm(range(1, 20, 2)):
    knn = KNeighborsClassifier(n_neighbors = k_value, algorithm='brute')
    knn.fit(BoW_dict['X_train_vect'],Y_train)

    train_proba = knn.score(BoW_dict['X_train_vect'], Y_train)
    bow_auc_train.append(train_proba)
    cv_proba = knn.score(BoW_dict['X_val_vect'], Y_val)
    bow_auc_cv.append(cv_proba)
```

```
100%|██████████| 10/10 [48:59<00:00, 297.93s/it]
```

In [68]:

```
k_vals = range(1, 20, 2)
train_k_dict = dict(zip(k_vals, bow_auc_train))
val_k_dict = dict(zip(k_vals, bow_auc_cv))
print(train_k_dict)
print(val_k_dict)
```

```
{1: 1.0, 3: 0.9159166666666667, 5: 0.8991833333333333, 7: 0.893383333333333
3, 9: 0.89095, 11: 0.8895666666666666, 13: 0.8883666666666666, 15: 0.8881166
666666667, 17: 0.8877166666666667, 19: 0.8872833333333333}
{1: 0.808, 3: 0.84645, 5: 0.85635, 7: 0.8586, 9: 0.85955, 11: 0.85985, 13:
0.861, 15: 0.861, 17: 0.862, 19: 0.8628}
```

In [70]:

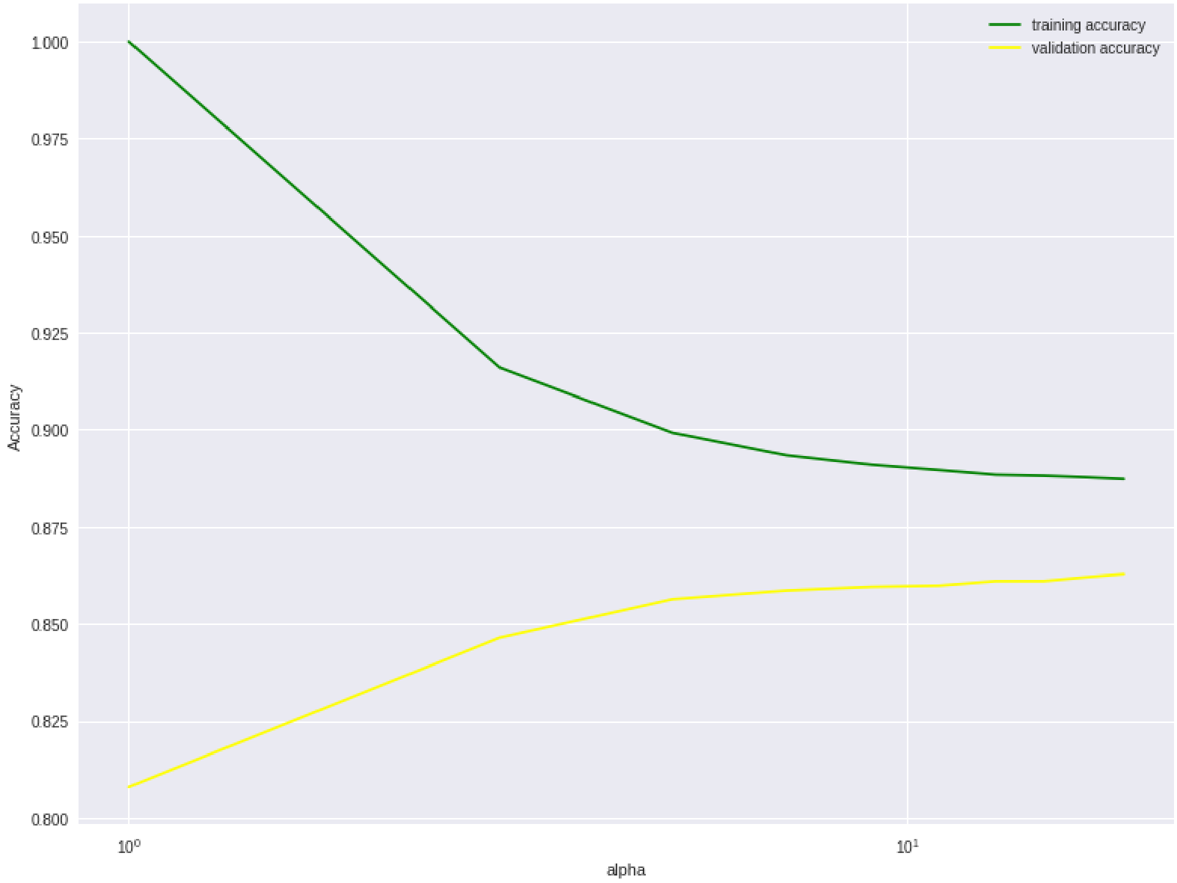
```
bow_best_k = max(val_k_dict, key=val_k_dict.get)
bow_best_k
```

Out[70]:

```
19
```

In [71]:

```
plt.figure(figsize=(13, 10))
neighbors_settings = range(1, 20, 2)
plt.plot(neighbors_settings, bow_auc_train, label="training accuracy", color='green')
plt.plot(neighbors_settings, bow_auc_cv, label="validation accuracy", color='yellow')
# plt.plot(neighbors_settings, auc_test, label="test accuracy", color='red')
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.legend()
plt.scatter()
plt.show()
```



In [72]:

```
bow_knn = KNeighborsClassifier(n_neighbors = bow_best_k, algorithm='brute')
bow_knn.fit(BoW_dict['X_train_vect'],Y_train)
test_predict_bow= bow_knn.predict_proba(BoW_dict['X_test_vect'])
train_predict_bow = bow_knn.predict_proba(BoW_dict['X_train_vect'])

bow_test_conf = bow_knn.predict(BoW_dict['X_test_vect'])
bow_train_conf = bow_knn.predict(BoW_dict['X_train_vect'])

print(type(test_predict_bow))
print(test_predict_bow[:, 1])
```

```
<class 'numpy.ndarray'>
[1.         0.89473684 0.89473684 ... 0.89473684 0.94736842 0.84210526]
```

In [73]:

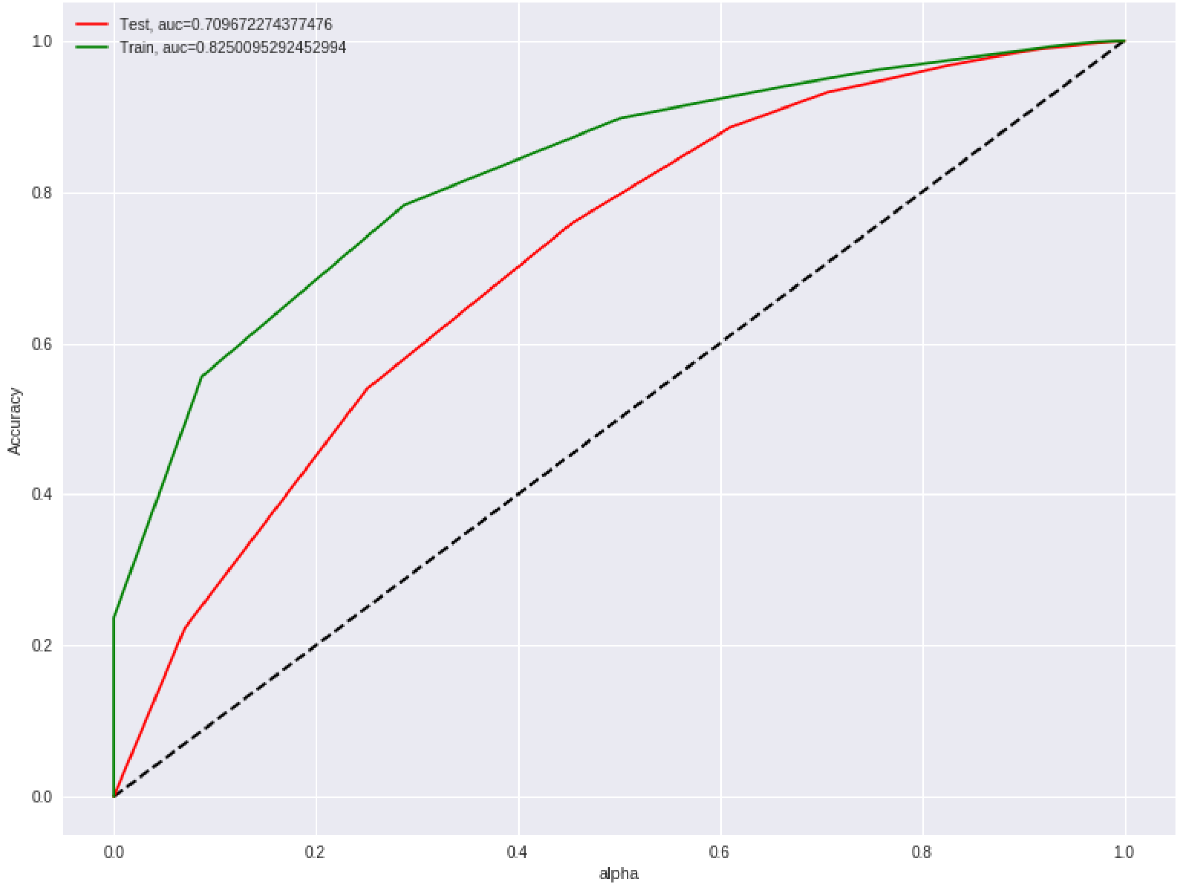
```
fpr_train, tpr_train, _ = roc_curve(Y_train, train_predict_bow[:, 1])
fpr_test, tpr_test, _ = roc_curve(Y_test, test_predict_bow[:, 1])
bow_test_auc = auc(fpr_test, tpr_test)
bow_train_auc = auc(fpr_train, tpr_train)
print(bow_test_auc)
print(bow_train_auc)
```

```
0.709672274377476
0.8250095292452994
```

In [74]:

```
import pylab
plt.figure(figsize=(13, 10))
plt.plot([0,1], [0,1], color='black', lw=2, linestyle='--')
plt.plot(fpr_test,tpr_test,label="Test, auc="+str(bow_test_auc), color = 'red')
plt.plot(fpr_train,tpr_train,label="Train, auc="+str(bow_train_auc), color = 'green')
plt.scatter(fpr_train, tpr_train, color = 'green')
plt.scatter(fpr_test, tpr_test, color = 'red')
plt.xlabel('alpha')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



In [75]:

```
from sklearn.metrics import classification_report, confusion_matrix
bow_train_conf_matrix = confusion_matrix(Y_train, bow_train_conf)
bow_test_conf_matrix = confusion_matrix(Y_test, bow_test_conf)
class_report = classification_report(Y_test, bow_test_conf)
print(bow_test_conf_matrix)
print(class_report)
```

```
[[  213  2463]
 [  177 17147]]
              precision    recall  f1-score   support

           0       0.55      0.08      0.14      2676
           1       0.87      0.99      0.93     17324

   micro avg       0.87      0.87      0.87     20000
   macro avg       0.71      0.53      0.53     20000
weighted avg       0.83      0.87      0.82     20000
```

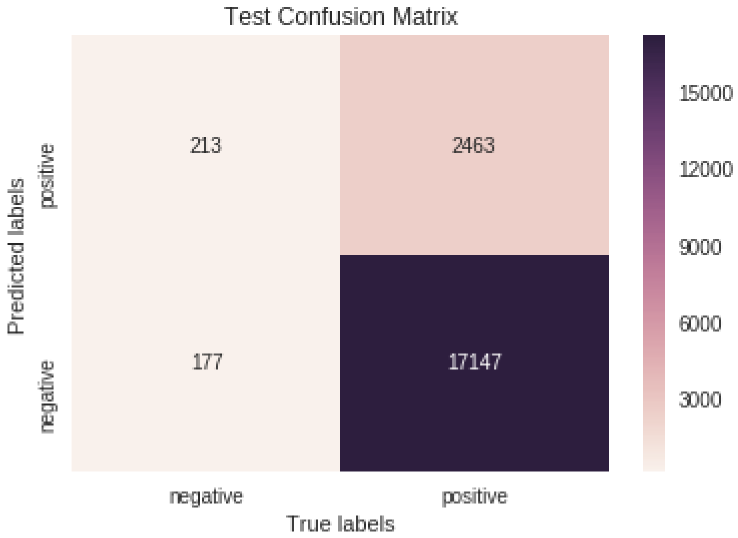
In [76]:

```
ax= plt.subplot()
sns.heatmap(bow_test_conf_matrix, annot=True, ax = ax, fmt='g')

ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Test Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[76]:

```
[Text(0, 0.5, 'negative'), Text(0, 1.5, 'positive')]
```



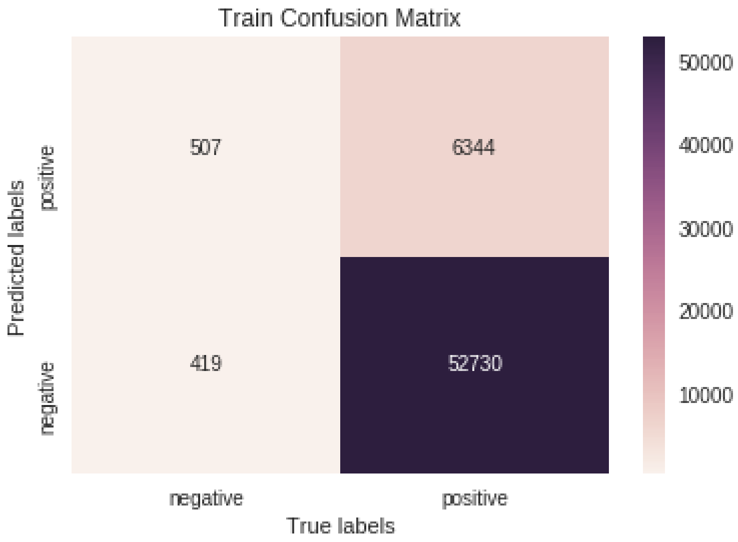
In [77]:

```
ax= plt.subplot()
sns.heatmap(bow_train_conf_matrix, annot=True, ax = ax, fmt='g')

ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Train Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[77]:

```
[Text(0, 0.5, 'negative'), Text(0, 1.5, 'positive')]
```



K-NN on tf-idf (Brute)

In [28]:

```
import pickle
with open(r"tf_idf.pkl", "rb") as input_file:
    tfidf_dict = pickle.load(input_file)
```

In [30]:

```
from sklearn.neighbors import KNeighborsClassifier

tfidf_auc_train = []
tfidf_auc_cv = []
tfidf_auc_test = []

for k_value in tqdm(range(1, 20, 2)):
    knn = KNeighborsClassifier(n_neighbors = k_value, algorithm='brute', n_jobs = -1)
    knn.fit(tfidf_dict['train_tf_idf'],Y_train)

    train_proba = knn.score(tfidf_dict['train_tf_idf'], Y_train)
    tfidf_auc_train.append(train_proba)
    cv_proba = knn.score(tfidf_dict['cv_tf_idf'], Y_val)
    tfidf_auc_cv.append(cv_proba)
```

```
  0%|          | 0/10 [00:00<?, ?it/s]
 10%|█         | 1/10 [04:52<43:52, 292.47s/it]
 20%|██        | 2/10 [09:56<39:27, 295.95s/it]
 30%|███       | 3/10 [15:28<35:47, 306.78s/it]
 40%|████      | 4/10 [20:59<31:23, 313.95s/it]
 50%|█████     | 5/10 [26:30<26:35, 319.13s/it]
 60%|██████    | 6/10 [32:00<21:29, 322.49s/it]
 70%|███████   | 7/10 [37:32<16:16, 325.39s/it]
 80%|████████  | 8/10 [43:03<10:53, 326.89s/it]
 90%|█████████ | 9/10 [48:34<05:28, 328.25s/it]
100%|██████████| 10/10 [54:07<00:00, 329.58s/it]
```

In [32]:

```
k_vals = range(1, 20, 2)
train_k_dict = dict(zip(k_vals, tfidf_auc_train))
val_k_dict = dict(zip(k_vals, tfidf_auc_cv))
print(train_k_dict)
print(val_k_dict)
```

```
{19: 0.8886666666666667, 1: 1.0, 3: 0.9231666666666667, 17: 0.88943333333333
33, 5: 0.9076333333333333, 7: 0.90035, 9: 0.8958833333333334, 11: 0.89343333
33333333, 13: 0.8918666666666667, 15: 0.8904166666666666}
{19: 0.86495, 1: 0.8259, 3: 0.8629, 17: 0.8656, 5: 0.87125, 7: 0.87015, 9:
0.8697, 11: 0.86845, 13: 0.8676, 15: 0.86635}
```

In [33]:

```
bow_best_k = max(val_k_dict, key=val_k_dict.get)
bow_best_k
```

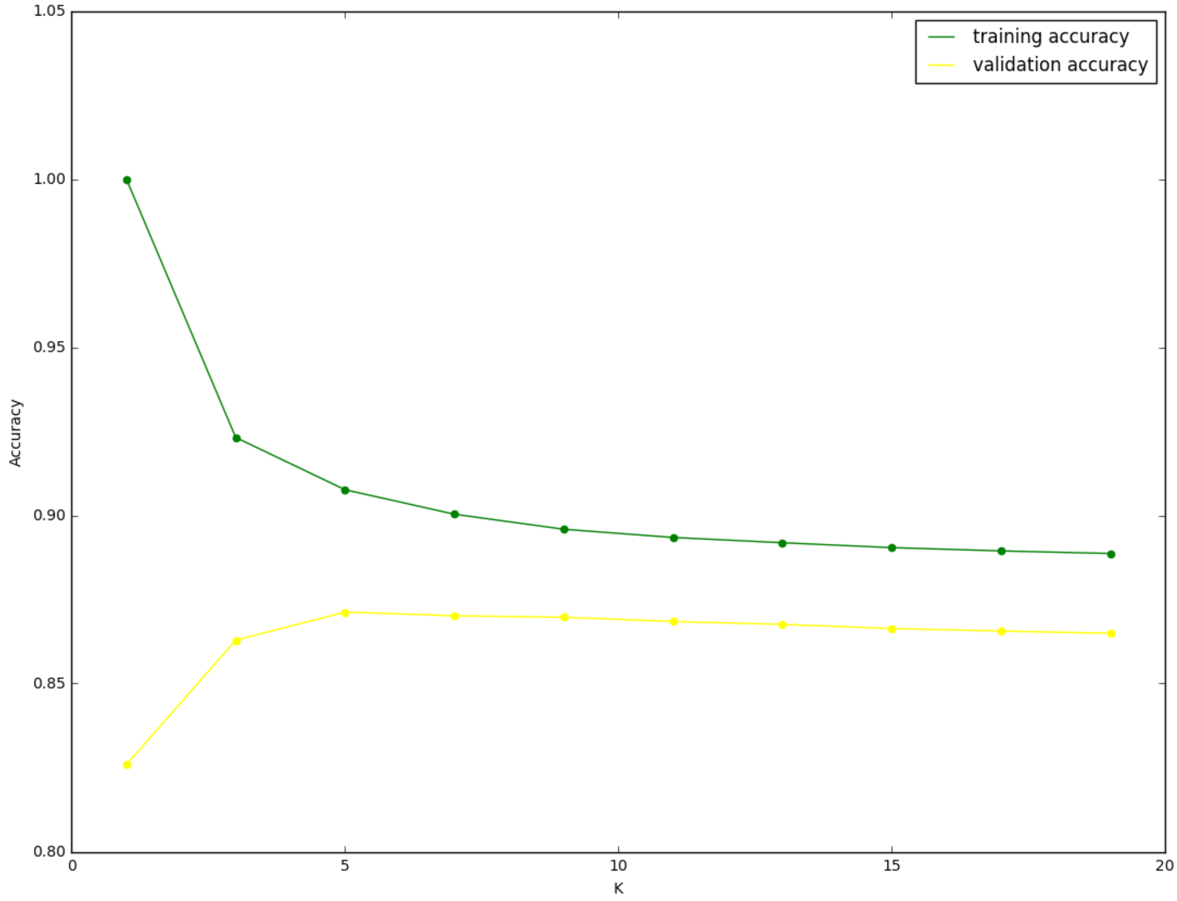
Out[33]:

```
5
```

In [119]:

```
plt.figure(figsize=(13, 10))
neighbors_settings = range(1, 20, 2)
plt.plot(neighbors_settings, tfidf_auc_train, label="training accuracy", color='green')
plt.plot(neighbors_settings, tfidf_auc_cv, label="validation accuracy", color='yellow')
plt.scatter(neighbors_settings, tfidf_auc_train, color='green')
plt.scatter(neighbors_settings, tfidf_auc_cv, color='yellow')
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



In [39]:

```
tfidf_knn = KNeighborsClassifier(n_neighbors = bow_best_k, algorithm='brute')
tfidf_knn.fit(tfidf_dict['train_tf_idf'],Y_train)
test_predict_tfidf= tfidf_knn.predict_proba(tfidf_dict['test_tf_idf'])
train_predict_tfidf = tfidf_knn.predict_proba(tfidf_dict['train_tf_idf'])

tfidf_test_conf = tfidf_knn.predict(tfidf_dict['test_tf_idf'])
tfidf_train_conf = tfidf_knn.predict(tfidf_dict['train_tf_idf'])

print(type(test_predict_tfidf))
print(test_predict_tfidf[:, 1])
```

```
<class 'numpy.ndarray'>
[1.  0.8 0.6 ... 1.  1.  0.8]
```

In [41]:

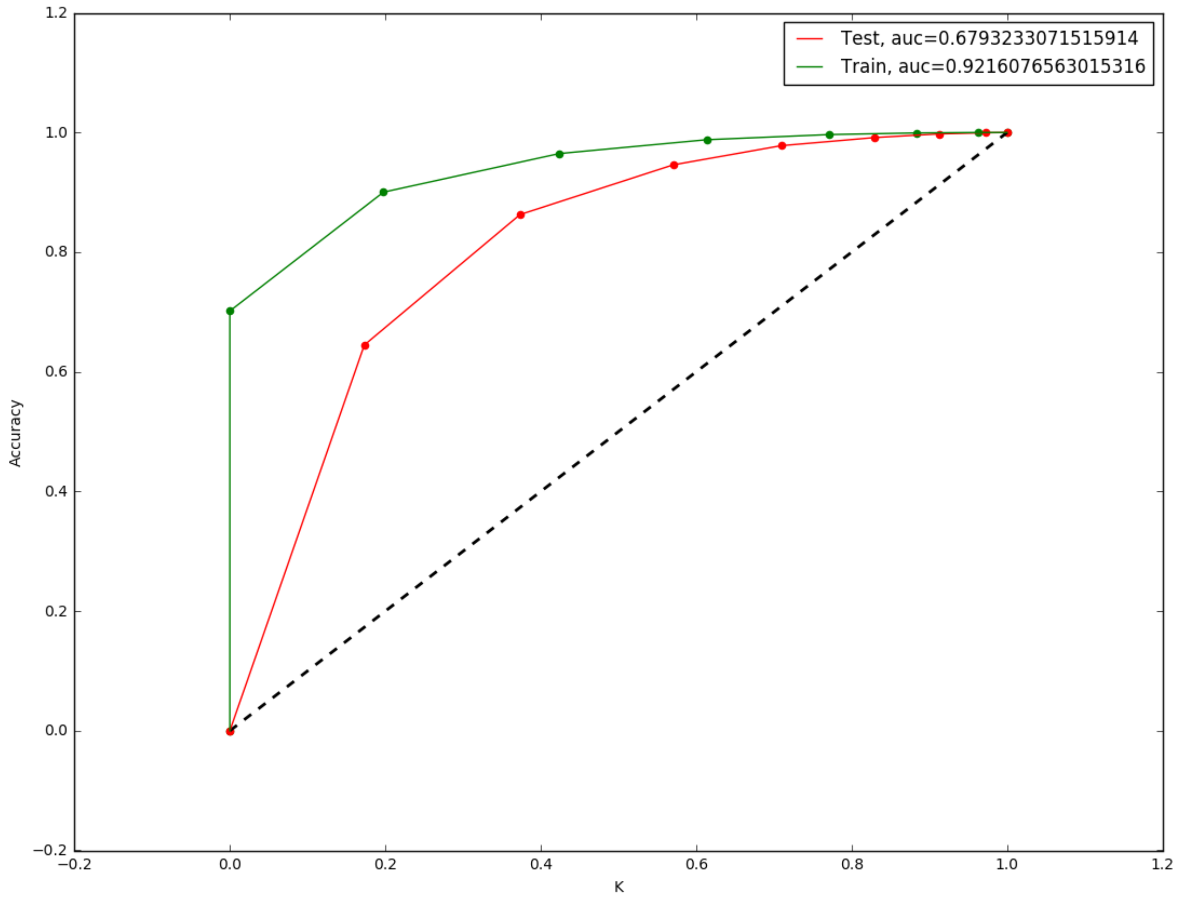
```
fpr_train, tpr_train, _ = roc_curve(Y_train, train_predict_tfidf[:, 1])
fpr_test, tpr_test, _ = roc_curve(Y_test, test_predict_tfidf[:, 1])
tfidf_test_auc = auc(fpr_test, tpr_test)
tfidf_train_auc = auc(fpr_train, tpr_train)
print(tfidf_test_auc)
print(tfidf_train_auc)
```

```
0.6793233071515914
0.9216076563015316
```

In [116]:

```
import pylab
plt.figure(figsize=(13, 10))
plt.plot([0,1], [0,1], color='black', lw=2, linestyle='--')
plt.plot(fpr_test, tpr_test, label="Test, auc="+str(tfidf_test_auc), color = 'red')
plt.plot(fpr_train, tpr_train, label="Train, auc="+str(tfidf_train_auc), color = 'green')
plt.scatter(fpr_train, tpr_train, color = 'green')
plt.scatter(fpr_test, tpr_test, color = 'red')
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



In [45]:

```
from sklearn.metrics import classification_report, confusion_matrix
tfidf_train_conf_matrix = confusion_matrix(Y_train, tfidf_train_conf)
tfidf_test_conf_matrix = confusion_matrix(Y_test, tfidf_test_conf)
class_report = classification_report(Y_test, tfidf_test_conf)
print(tfidf_test_conf_matrix)
print(class_report)
```

```
[[  283  2393]
 [  185 17139]]
              precision    recall  f1-score   support

           0       0.60      0.11      0.18      2676
           1       0.88      0.99      0.93     17324

   micro avg       0.87      0.87      0.87     20000
   macro avg       0.74      0.55      0.56     20000
weighted avg       0.84      0.87      0.83     20000
```

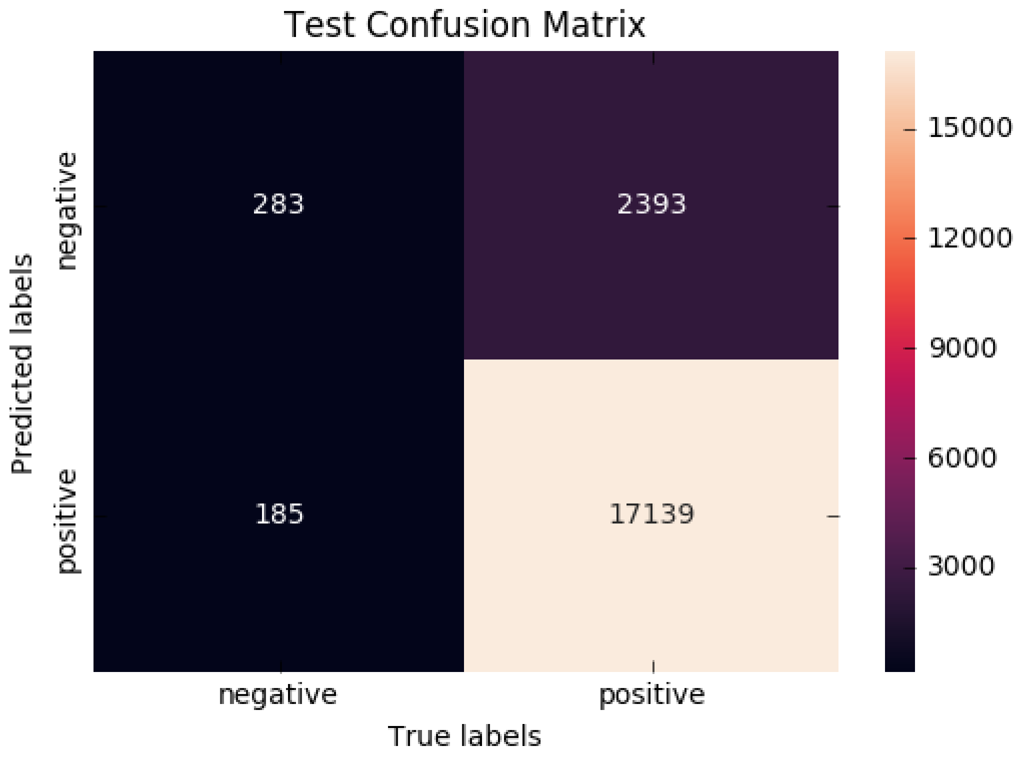
In [46]:

```
ax= plt.subplot()
sns.heatmap(tfidf_test_conf_matrix, annot=True, ax = ax, fmt='g')

ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Test Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[46]:

```
[<matplotlib.text.Text at 0x7f70b142e3c8>,
 <matplotlib.text.Text at 0x7f70b14ffb38>]
```



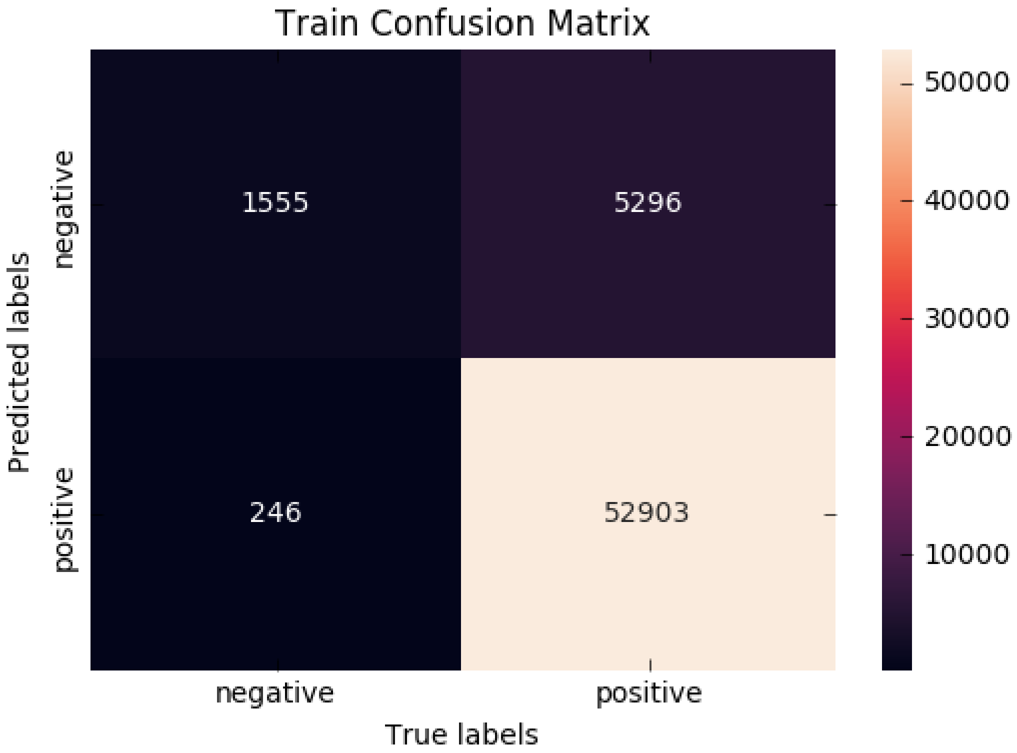
In [47]:

```
ax= plt.subplot()
sns.heatmap(tfidf_train_conf_matrix, annot=True, ax = ax, fmt='g')

ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Train Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[47]:

```
[<matplotlib.text.Text at 0x7f70b27bb160>,
 <matplotlib.text.Text at 0x7f70b2243f28>]
```



K-NN on Avg-tfidf (Brute)

In [77]:

```
import pickle
with open(r"avg_w2v.pkl", "rb") as input_file:
    avgtfidf_dict = pickle.load(input_file)
```

In [78]:

```
from sklearn.neighbors import KNeighborsClassifier

avg_auc_train = []
avg_auc_cv = []
avg_auc_test = []

for k_value in tqdm(range(1, 20, 2)):
    knn = KNeighborsClassifier(n_neighbors = k_value, algorithm='brute', n_jobs = -1)
    knn.fit(avgtfidf_dict['X_train_avgw2v'],Y_train)

    train_proba = knn.score(avgtfidf_dict['X_train_avgw2v'], Y_train)
    avg_auc_train.append(train_proba)
    cv_proba = knn.score(avgtfidf_dict['X_val_avgw2v'], Y_val)
    avg_auc_cv.append(cv_proba)
```

```
  0%|          | 0/10 [00:00<?, ?it/s]

 10%|█         | 1/10 [04:31<40:47, 271.92s/it]

 20%|██        | 2/10 [09:16<36:45, 275.66s/it]

 30%|███       | 3/10 [14:36<33:42, 288.87s/it]

 40%|████      | 4/10 [19:56<29:49, 298.31s/it]

 50%|█████     | 5/10 [25:13<25:20, 304.04s/it]

 60%|██████    | 6/10 [30:35<20:37, 309.49s/it]

 70%|███████   | 7/10 [35:55<15:37, 312.47s/it]

 80%|████████  | 8/10 [41:17<10:30, 315.38s/it]

 90%|█████████ | 9/10 [46:37<05:16, 316.66s/it]

100%|██████████| 10/10 [51:59<00:00, 318.34s/it]
```

In [79]:

```
k_vals = range(1, 20, 2)
train_k_dict = dict(zip(k_vals, avg_auc_train))
val_k_dict = dict(zip(k_vals, avg_auc_cv))
print(train_k_dict)
print(val_k_dict)
```

```
{19: 0.91235, 1: 0.9999666666666667, 3: 0.9427333333333333, 17: 0.9129166666
666667, 5: 0.9299166666666666, 7: 0.9243166666666667, 9: 0.9206166666666666,
11: 0.9186, 13: 0.91645, 15: 0.9145}
{19: 0.8913, 1: 0.8649, 3: 0.88485, 17: 0.89285, 5: 0.89155, 7: 0.8937, 9:
0.8946, 11: 0.8938, 13: 0.894, 15: 0.8937}
```

In [80]:

```
avg_best_k = max(val_k_dict, key=val_k_dict.get)
avg_best_k
```

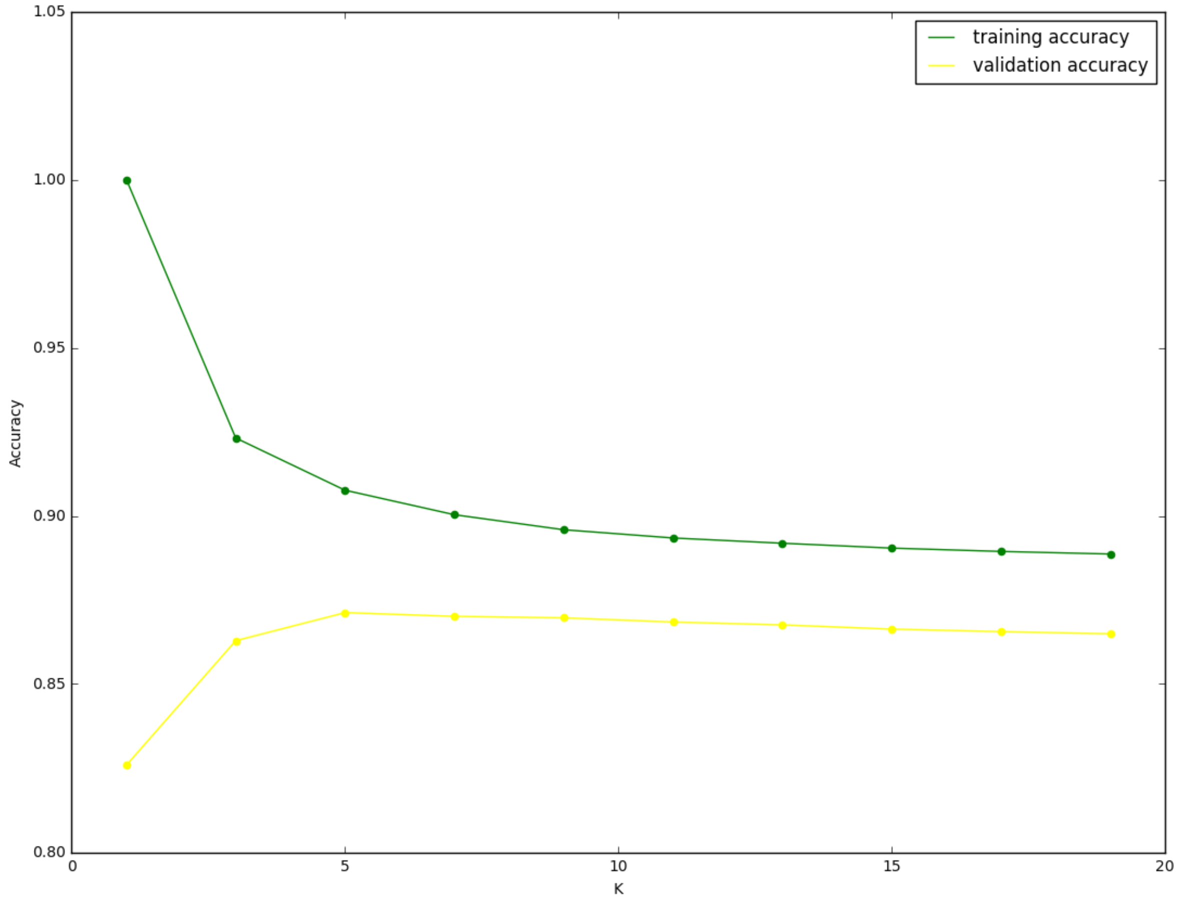
Out[80]:

```
9
```

In [118]:

```
plt.figure(figsize=(13, 10))
neighbors_settings = range(1, 20, 2)
plt.plot(neighbors_settings, avg_auc_train, label="training accuracy", color='green')
plt.plot(neighbors_settings, avg_auc_cv, label="validation accuracy", color='yellow')
plt.scatter(neighbors_settings, avg_auc_train, color='green')
plt.scatter(neighbors_settings, avg_auc_cv, color='yellow')
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



In [83]:

```
avg_knn = KNeighborsClassifier(n_neighbors = bow_best_k, algorithm='brute', n_jobs=-1)
avg_knn.fit(avgtfidf_dict['X_train_avgw2v'],Y_train)
test_predict_avg = avg_knn.predict_proba(avgtfidf_dict['X_test_avgw2v'])
train_predict_avg = avg_knn.predict_proba(avgtfidf_dict['X_train_avgw2v'])

avg_test_conf = avg_knn.predict(avgtfidf_dict['X_test_avgw2v'])
avg_train_conf = avg_knn.predict(avgtfidf_dict['X_train_avgw2v'])

print(type(train_predict_avg))
print(train_predict_avg[:, 1])
```

```
<class 'numpy.ndarray'>
[0.8 1.  1.  ... 1.  1.  1. ]
```

In [85]:

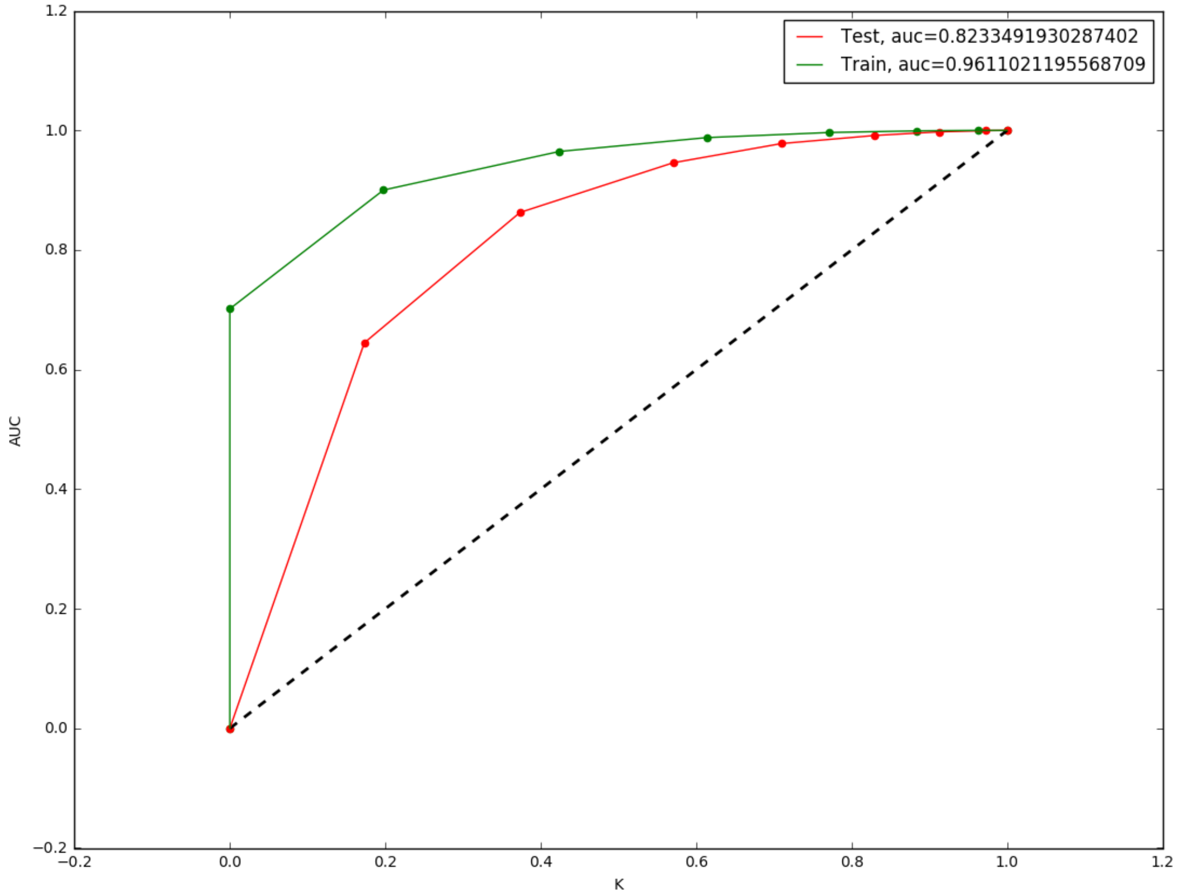
```
fpr_train, tpr_train, _ = roc_curve(Y_train, train_predict_avg[:, 1])
fpr_test, tpr_test, _ = roc_curve(Y_test, test_predict_avg[:, 1])
avg_test_auc = auc(fpr_test, tpr_test)
avg_train_auc = auc(fpr_train, tpr_train)
print(avg_test_auc)
print(avg_train_auc)
```

```
0.8233491930287402
0.9611021195568709
```

In [115]:

```
plt.figure(figsize=(13, 10))
plt.plot([0,1], [0,1], color='black', lw=2, linestyle='--')
plt.plot(fpr_test, tpr_test, label="Test, auc="+str(avg_test_auc), color = 'red')
plt.plot(fpr_train, tpr_train, label="Train, auc="+str(avg_train_auc), color = 'green')
plt.scatter(fpr_train, tpr_train, color = 'green')
plt.scatter(fpr_test, tpr_test, color = 'red')
plt.xlabel('K')
plt.ylabel('AUC')
plt.legend()

plt.show()
```



In [87]:

```
from sklearn.metrics import classification_report, confusion_matrix
avg_train_conf_matrix = confusion_matrix(Y_train, avg_train_conf)
avg_test_conf_matrix = confusion_matrix(Y_test, avg_test_conf)
class_report = classification_report(Y_test, avg_test_conf)
print(avg_test_conf_matrix)
print(class_report)
```

```
[[  893  1783]
 [  422 16902]]
              precision    recall  f1-score   support

           0       0.68      0.33      0.45      2676
           1       0.90      0.98      0.94     17324

   micro avg       0.89      0.89      0.89     20000
   macro avg       0.79      0.65      0.69     20000
weighted avg       0.87      0.89      0.87     20000
```

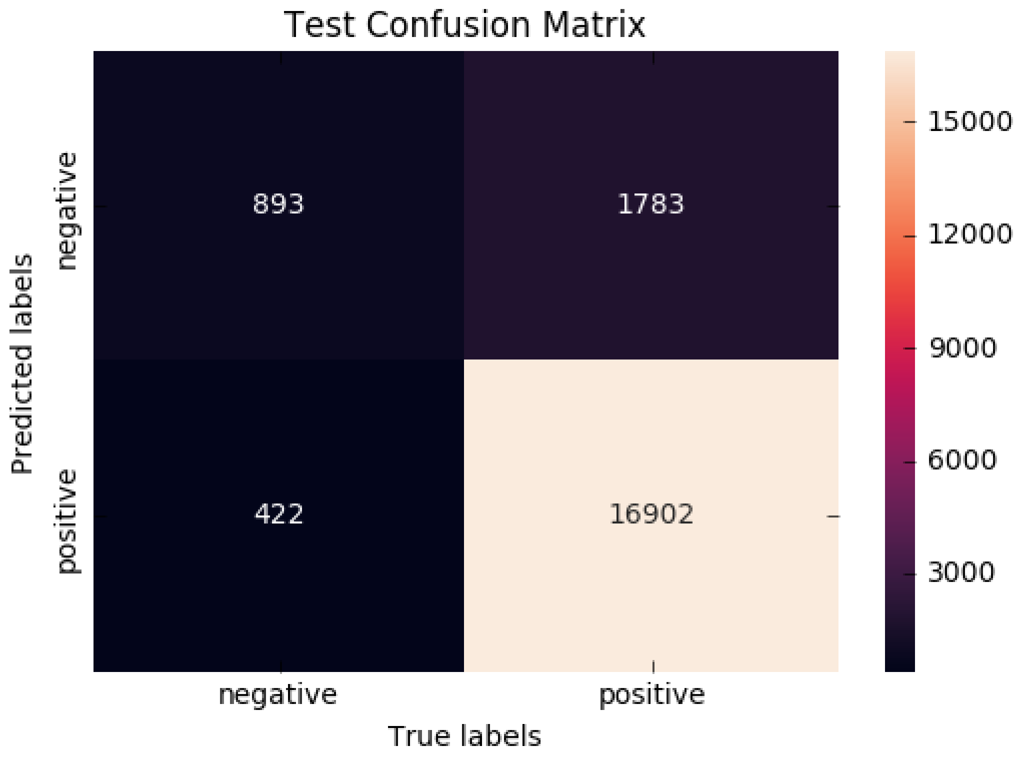
In [88]:

```
ax= plt.subplot()
sns.heatmap(avg_test_conf_matrix, annot=True, ax = ax, fmt='g')

ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Test Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[88]:

```
[<matplotlib.text.Text at 0x7f70aea5ac18>,
 <matplotlib.text.Text at 0x7f70ae71b6d8>]
```



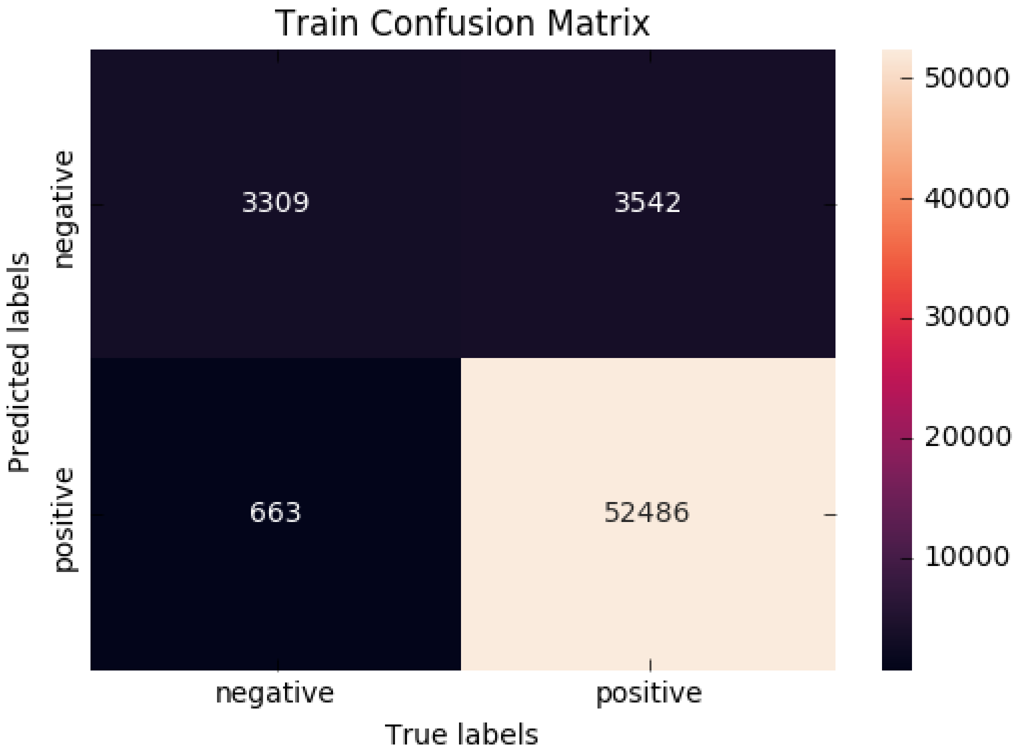
In [89]:

```
ax= plt.subplot()
sns.heatmap(avg_train_conf_matrix, annot=True, ax = ax, fmt='g')

ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Train Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[89]:

```
[<matplotlib.text.Text at 0x7f70aff7f160>,
 <matplotlib.text.Text at 0x7f70b23370b8>]
```



K-NN on TF-IDF weighted w2v (Brute)

In [ ]:

```
import pickle
with open(r"tfidf_w2v.pkl", "rb") as input_file:
    tfidfw2v_dict = pickle.load(input_file)
```

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier

avg_auc_train = []
avg_auc_cv = []
avg_auc_test = []

for k_value in tqdm(range(1, 20, 2)):
    knn = KNeighborsClassifier(n_neighbors = k_value, algorithm='brute', n_jobs = -1)
    knn.fit(tfidfw2v_dict['X_train_tfidfw2v'],Y_train)

    train_proba = knn.score(tfidfw2v_dict['X_train_tfidfw2v'], Y_train)
    avg_auc_train.append(train_proba)
    cv_proba = knn.score(tfidfw2v_dict['X_val_tfidfw2v'], Y_val)
    avg_auc_cv.append(cv_proba)
```

```
  0%|          | 0/10 [00:00<?, ?it/s]

 10%|█         | 1/10 [04:33<40:57, 273.11s/it]

 20%|██        | 2/10 [09:20<37:00, 277.51s/it]

 30%|███       | 3/10 [14:40<33:51, 290.14s/it]

 40%|████      | 4/10 [20:00<29:55, 299.18s/it]

 50%|█████     | 5/10 [25:19<25:25, 305.07s/it]

 60%|██████    | 6/10 [30:40<20:39, 309.79s/it]

 70%|███████   | 7/10 [35:58<15:36, 312.23s/it]

 80%|████████  | 8/10 [41:20<10:30, 315.13s/it]

 90%|█████████ | 9/10 [46:41<05:16, 316.85s/it]

100%|██████████| 10/10 [52:01<00:00, 318.02s/it]
```

In [100]:

```
k_vals = range(1, 20, 2)
train_k_dict = dict(zip(k_vals, avg_auc_train))
val_k_dict = dict(zip(k_vals, avg_auc_cv))
print(train_k_dict)
print(val_k_dict)
```

```
{19: 0.9085666666666666, 1: 0.9999666666666667, 3: 0.9360166666666667, 17:
0.9094833333333333, 5: 0.9248166666666666, 7: 0.9192666666666667, 9: 0.91621
66666666667, 11: 0.9139333333333334, 13: 0.9121666666666667, 15: 0.910666666
6666666}
{19: 0.88555, 1: 0.85065, 3: 0.88025, 17: 0.8857, 5: 0.88615, 7: 0.8868, 9:
0.8862, 11: 0.8866, 13: 0.886, 15: 0.88565}
```

In [103]:

```
tfidfw2v_best_k = max(val_k_dict, key=val_k_dict.get)
tfidfw2v_best_k
```

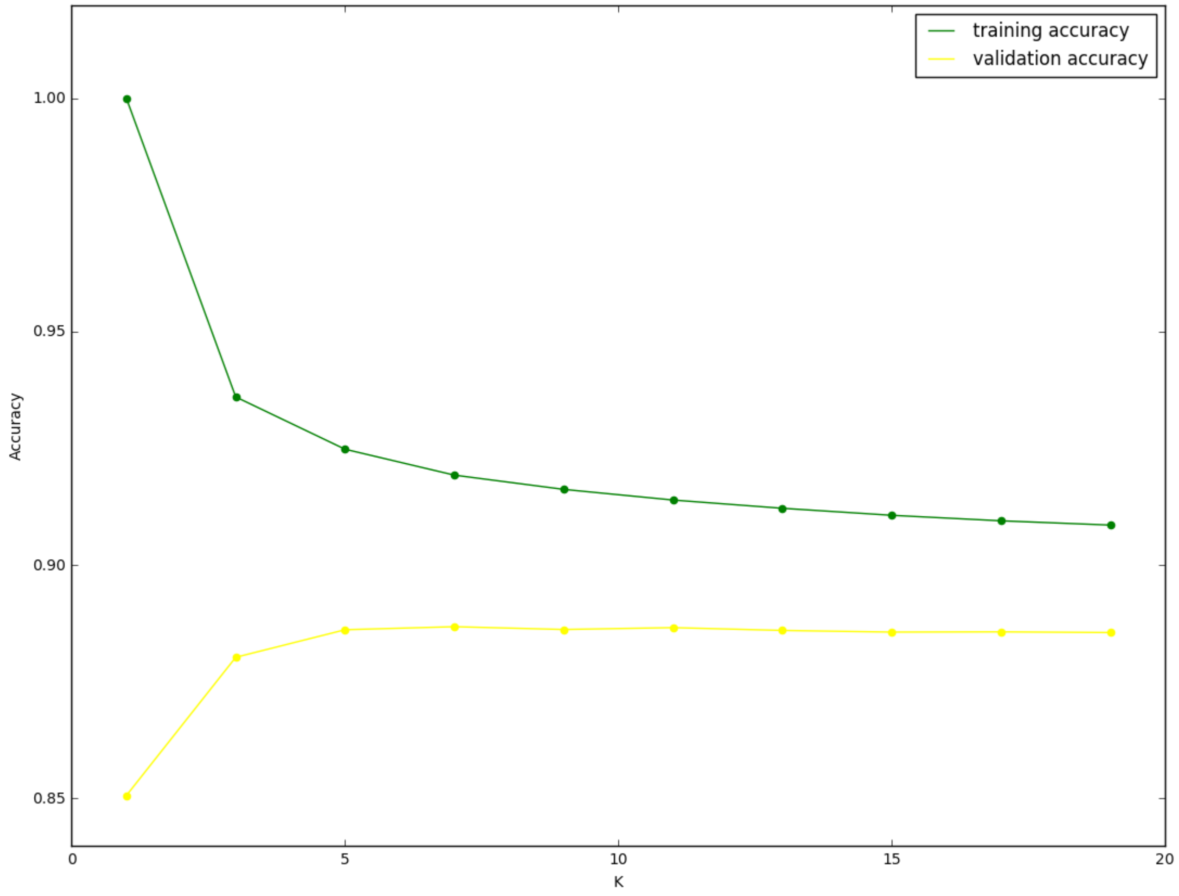
Out[103]:

```
7
```

In [117]:

```
plt.figure(figsize=(13, 10))
neighbors_settings = range(1, 20, 2)
plt.plot(neighbors_settings, avg_auc_train, label="training accuracy", color='green')
plt.plot(neighbors_settings, avg_auc_cv, label="validation accuracy", color='yellow')
plt.scatter(neighbors_settings, avg_auc_train, color='green')
plt.scatter(neighbors_settings, avg_auc_cv, color='yellow')
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



In [104]:

```
tfidfw2v_knn = KNeighborsClassifier(n_neighbors = tfidfw2v_best_k, algorithm='brute', n_job
tfidfw2v_knn.fit(tfidfw2v_dict['X_train_tfidfw2v'],Y_train)
test_predict_tfidfw2v = tfidfw2v_knn.predict_proba(tfidfw2v_dict['X_test_tfidfw2v'])
train_predict_tfidfw2v = tfidfw2v_knn.predict_proba(tfidfw2v_dict['X_train_tfidfw2v'])
tfidfw2v_test_conf = tfidfw2v_knn.predict(tfidfw2v_dict['X_test_tfidfw2v'])
tfidfw2v_train_conf = tfidfw2v_knn.predict(tfidfw2v_dict['X_train_tfidfw2v'])

print(type(train_predict_tfidfw2v))
print(train_predict_tfidfw2v[:, 1])
```

```
<class 'numpy.ndarray'>
[1. 1. 1. ... 1. 1. 1.]
```

In [106]:

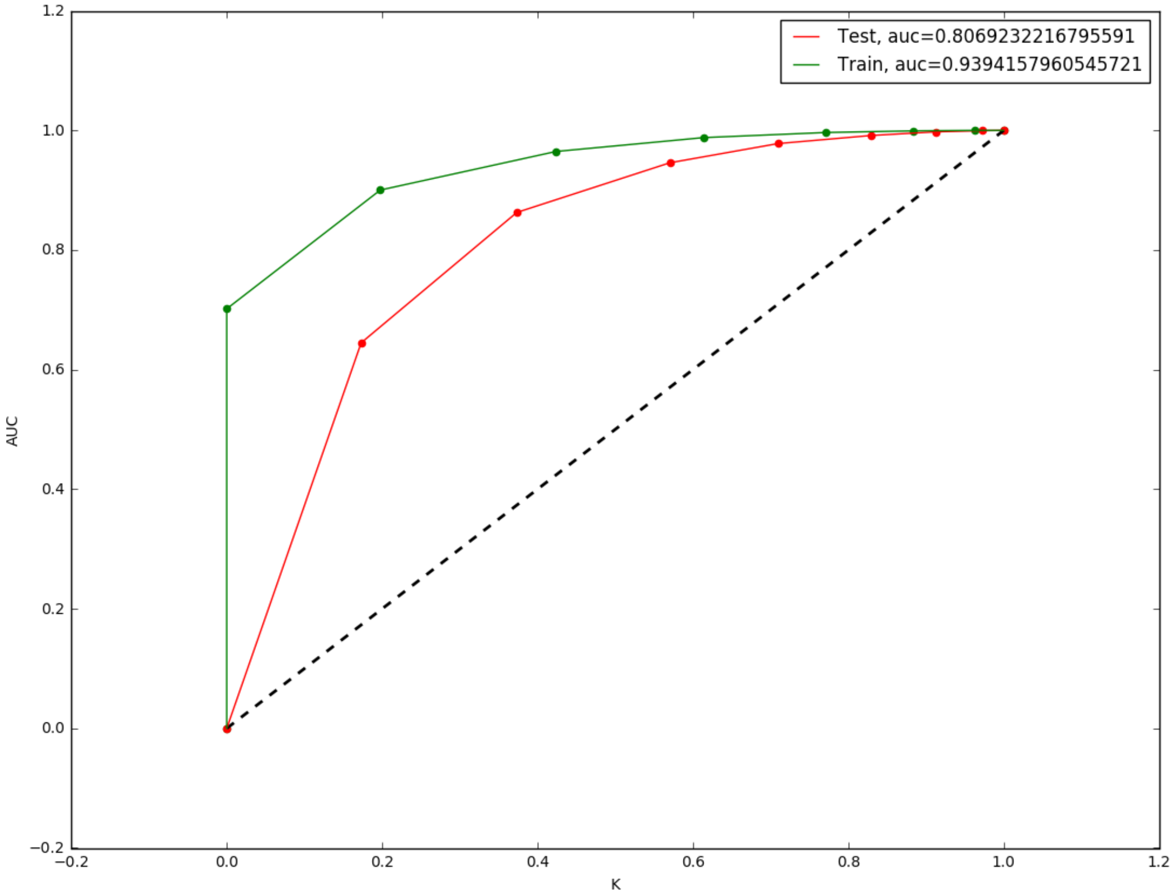
```
fpr_train, tpr_train, _ = roc_curve(Y_train, train_predict_tfidfw2v[:, 1])
fpr_test, tpr_test, _ = roc_curve(Y_test, test_predict_tfidfw2v[:, 1])
tfidfw2v_test_auc = auc(fpr_test, tpr_test)
tfidfw2v_train_auc = auc(fpr_train, tpr_train)
print(tfidfw2v_test_auc)
print(tfidfw2v_train_auc)
```

```
0.8069232216795591
0.9394157960545721
```

In [114]:

```
plt.figure(figsize=(13, 10))
plt.plot([0,1], [0,1], color='black', lw=2, linestyle='--')
plt.plot(fpr_test, tpr_test, label="Test, auc="+str(tfidfw2v_test_auc), color = 'red')
plt.plot(fpr_train, tpr_train, label="Train, auc="+str(tfidfw2v_train_auc), color = 'green'
plt.scatter(fpr_train, tpr_train, color = 'green')
plt.scatter(fpr_test, tpr_test, color = 'red')
plt.xlabel('K')
plt.ylabel('AUC')
plt.legend()

plt.show()
```



In [108]:

```
from sklearn.metrics import classification_report, confusion_matrix
tfidfw2v_train_conf_matrix = confusion_matrix(Y_train, tfidfw2v_train_conf)
tfidfw2v_test_conf_matrix = confusion_matrix(Y_test, tfidfw2v_test_conf)
class_report = classification_report(Y_test, tfidfw2v_test_conf)
print(tfidfw2v_test_conf_matrix)
print(class_report)
```

```
[[  777  1899]
 [  381 16943]]
              precision    recall  f1-score   support

           0       0.67      0.29      0.41      2676
           1       0.90      0.98      0.94     17324

   micro avg       0.89      0.89      0.89     20000
   macro avg       0.79      0.63      0.67     20000
weighted avg       0.87      0.89      0.87     20000
```

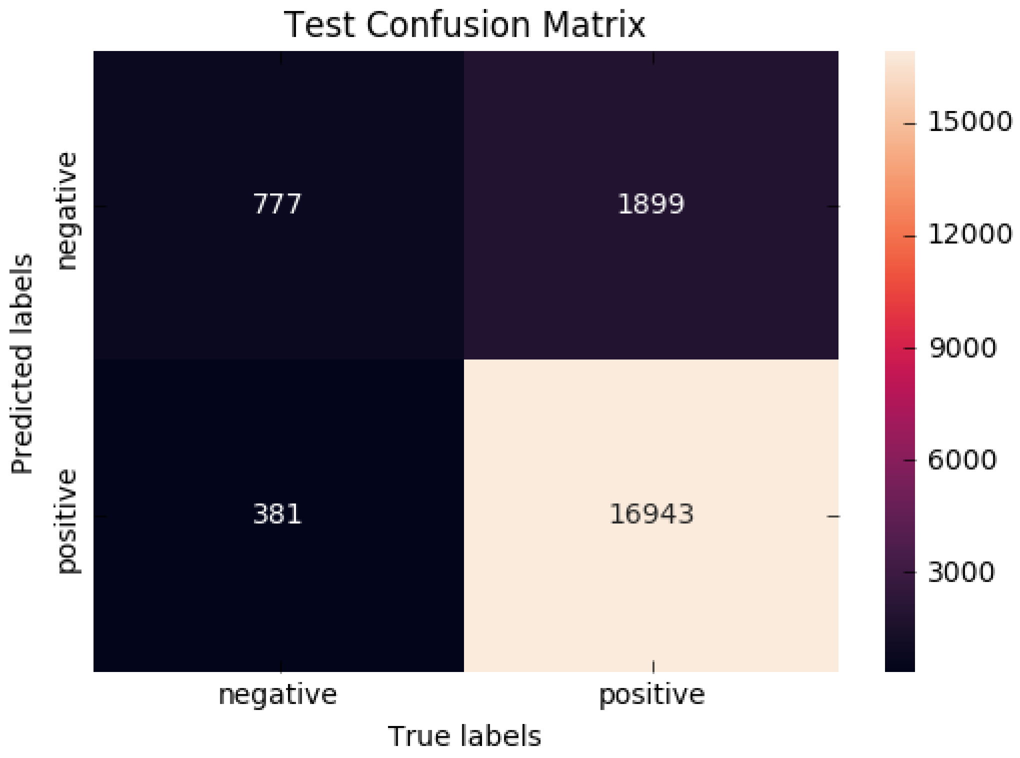
In [109]:

```
ax= plt.subplot()
sns.heatmap(tfidfw2v_test_conf_matrix, annot=True, ax = ax, fmt='g')

ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Test Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[109]:

```
[<matplotlib.text.Text at 0x7f70b025f908>,
 <matplotlib.text.Text at 0x7f708d66a2e8>]
```



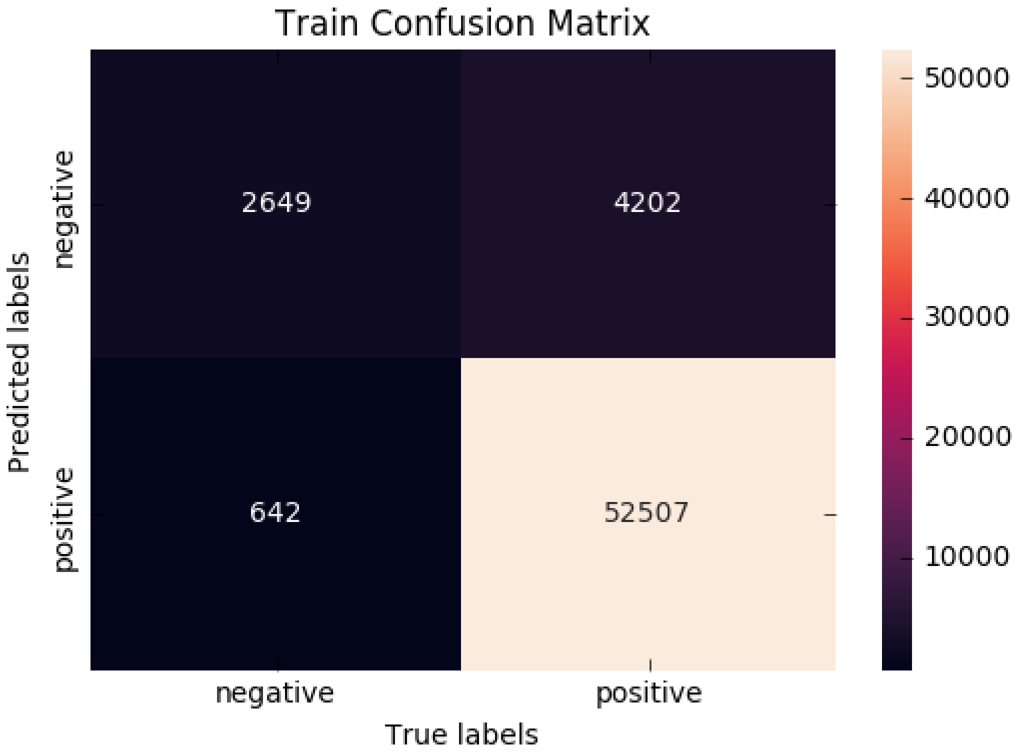
In [110]:

```
ax= plt.subplot()
sns.heatmap(tfidfw2v_train_conf_matrix, annot=True, ax = ax, fmt='g')

ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Train Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[110]:

```
[<matplotlib.text.Text at 0x7f70aef2e358>,
 <matplotlib.text.Text at 0x7f70b57af2b0>]
```



K-NN on BoW (k-d tree)

In [16]:

```
import pickle
with open(r"BoW.pkl", "rb") as input_file:
    BoW_dict = pickle.load(input_file)
```

In [18]:

```
from sklearn.neighbors import KNeighborsClassifier
from tqdm import tqdm

bow_auc_train = []
bow_auc_cv = []
bow_auc_test = []

for k_value in tqdm(range(1, 20, 2)):
    knn = KNeighborsClassifier(n_neighbors = k_value, algorithm='kd_tree', n_jobs = -1)
    knn.fit(BoW_dict['X_train_vect'],Y_train)

    train_proba = knn.score(BoW_dict['X_train_vect'], Y_train)
    bow_auc_train.append(train_proba)
    cv_proba = knn.score(BoW_dict['X_val_vect'], Y_val)
    bow_auc_cv.append(cv_proba)
```

```
100%|██████████| 10/10 [55:41<00:00, 336.26s/it]
```

In [19]:

```
k_vals = range(1, 20, 2)
train_k_dict = dict(zip(k_vals, bow_auc_train))
val_k_dict = dict(zip(k_vals, bow_auc_cv))
print(train_k_dict)
print(val_k_dict)
```

```
{19: 0.8872833333333333, 1: 1.0, 3: 0.9159166666666667, 17: 0.88771666666666
67, 5: 0.8991833333333333, 7: 0.8933833333333333, 9: 0.89095, 11: 0.88956666
66666666, 13: 0.8883666666666666, 15: 0.8881166666666667}
{19: 0.8628, 1: 0.808, 3: 0.84645, 17: 0.862, 5: 0.85635, 7: 0.8586, 9: 0.85
955, 11: 0.85985, 13: 0.861, 15: 0.861}
```

In [20]:

```
bow_best_k = max(val_k_dict, key=val_k_dict.get)
bow_best_k
```

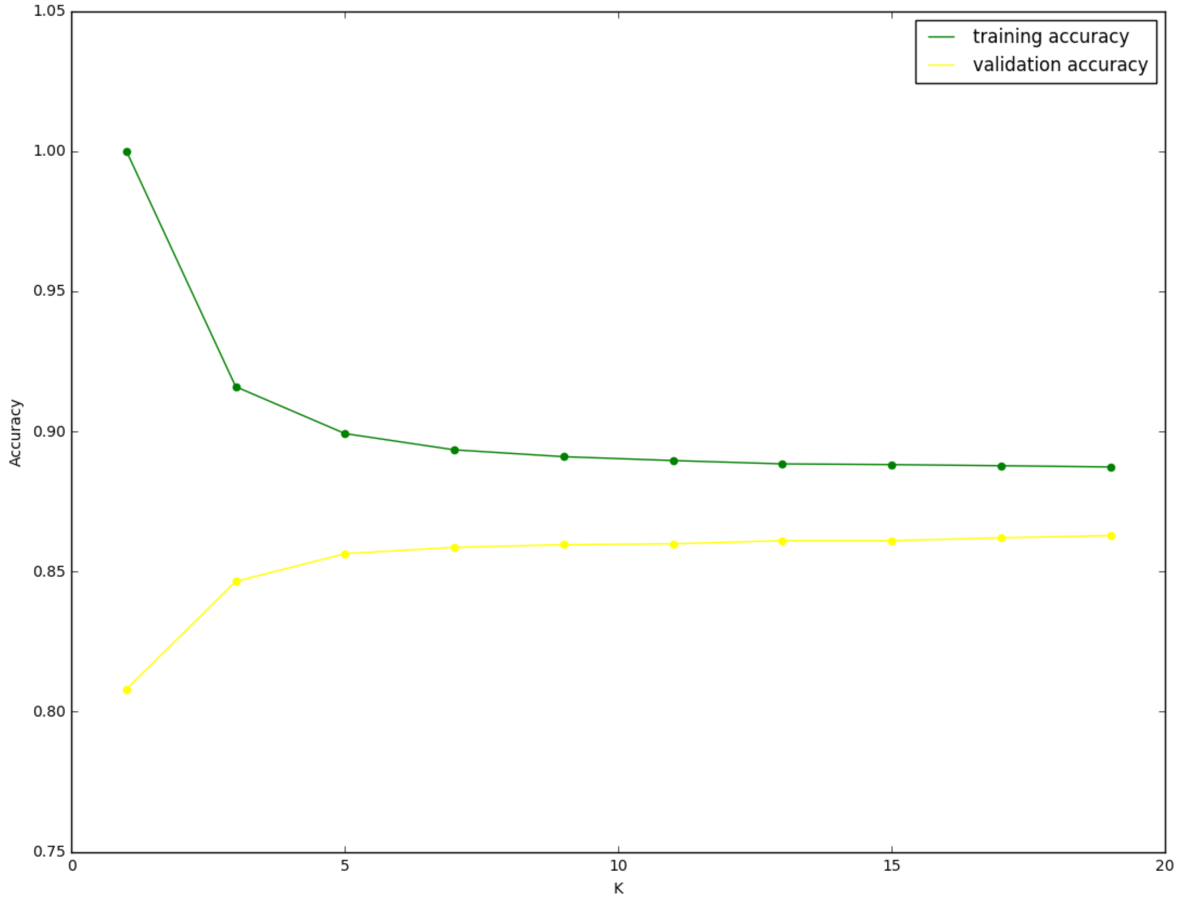
Out[20]:

```
19
```

In [21]:

```
plt.figure(figsize=(13, 10))
neighbors_settings = range(1, 20, 2)
plt.plot(neighbors_settings, bow_auc_train, label="training accuracy", color='green')
plt.plot(neighbors_settings, bow_auc_cv, label="validation accuracy", color='yellow')
plt.scatter(neighbors_settings, bow_auc_train, color='green')
plt.scatter(neighbors_settings, bow_auc_cv, color='yellow')
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



In [23]:

```
bow_knn = KNeighborsClassifier(n_neighbors = bow_best_k, algorithm='kd_tree', n_jobs=-1)
bow_knn.fit(BoW_dict['X_train_vect'],Y_train)
test_predict_bow = bow_knn.predict_proba(BoW_dict['X_test_vect'])
train_predict_bow = bow_knn.predict_proba(BoW_dict['X_train_vect'])
bow_test_conf = bow_knn.predict(BoW_dict['X_test_vect'])
bow_train_conf = bow_knn.predict(BoW_dict['X_train_vect'])

print(type(train_predict_bow))
print(train_predict_bow[:, 1])
```

```
<class 'numpy.ndarray'>
[0.84210526 0.94736842 0.94736842 ... 0.89473684 0.94736842 0.89473684]
```

In [24]:

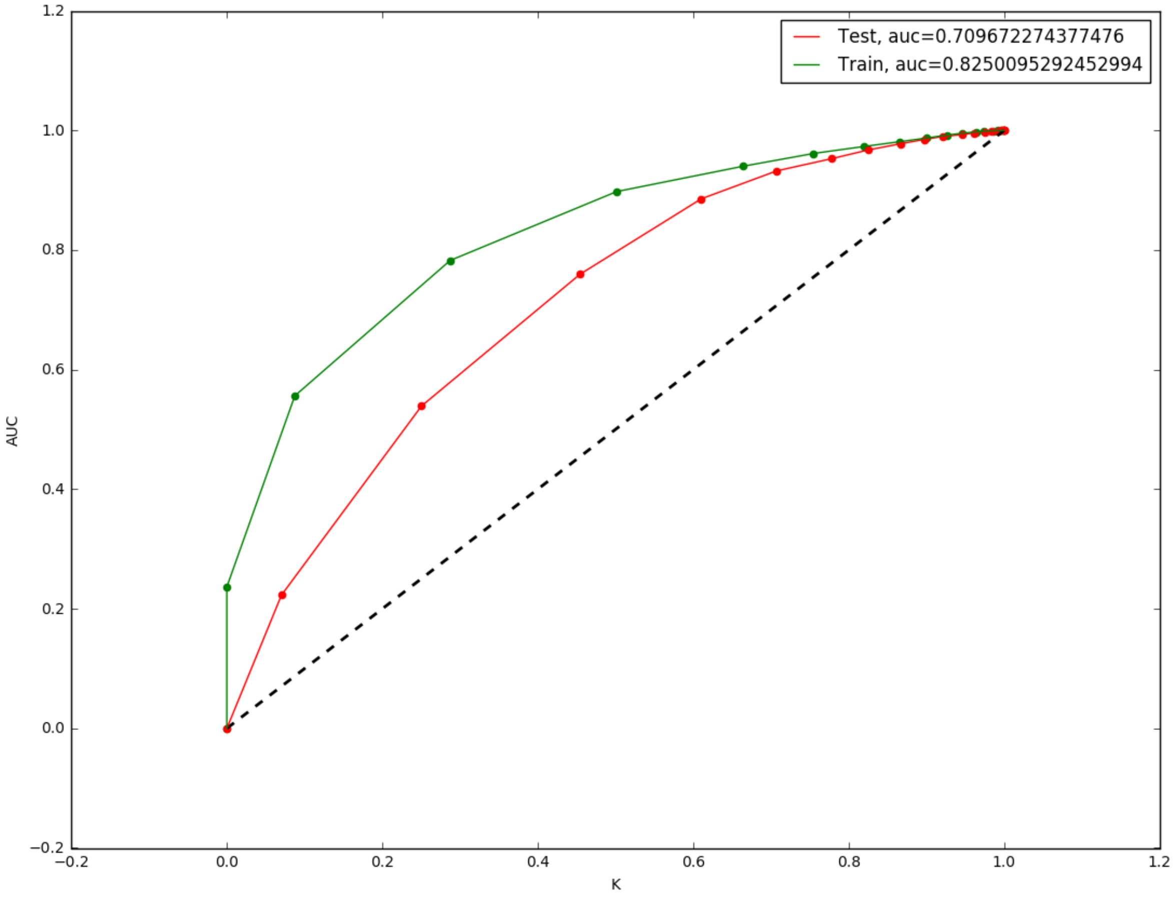
```
fpr_train, tpr_train, _ = roc_curve(Y_train, train_predict_bow[:, 1])
fpr_test, tpr_test, _ = roc_curve(Y_test, test_predict_bow[:, 1])
bow_test_auc = auc(fpr_test, tpr_test)
bow_train_auc = auc(fpr_train, tpr_train)
print(bow_test_auc)
print(bow_train_auc)
```

```
0.709672274377476
0.8250095292452994
```

In [26]:

```
plt.figure(figsize=(13, 10))
plt.plot([0,1], [0,1], color='black', lw=2, linestyle='--')
plt.plot(fpr_test, tpr_test, label="Test, auc="+str(bow_test_auc), color = 'red')
plt.plot(fpr_train, tpr_train, label="Train, auc="+str(bow_train_auc), color = 'green')
plt.scatter(fpr_train, tpr_train, color = 'green')
plt.scatter(fpr_test, tpr_test, color = 'red')
plt.xlabel('K')
plt.ylabel('AUC')
plt.legend()

plt.show()
```



In [28]:

```
from sklearn.metrics import classification_report, confusion_matrix
bow_train_conf_matrix = confusion_matrix(Y_train, bow_train_conf)
bow_test_conf_matrix = confusion_matrix(Y_test, bow_test_conf)
class_report = classification_report(Y_test, bow_test_conf)
print(bow_test_conf_matrix)
print(class_report)
```

```
[[  213  2463]
 [  177 17147]]
              precision    recall  f1-score   support

           0       0.55      0.08      0.14      2676
           1       0.87      0.99      0.93     17324

   micro avg       0.87      0.87      0.87     20000
   macro avg       0.71      0.53      0.53     20000
weighted avg       0.83      0.87      0.82     20000
```

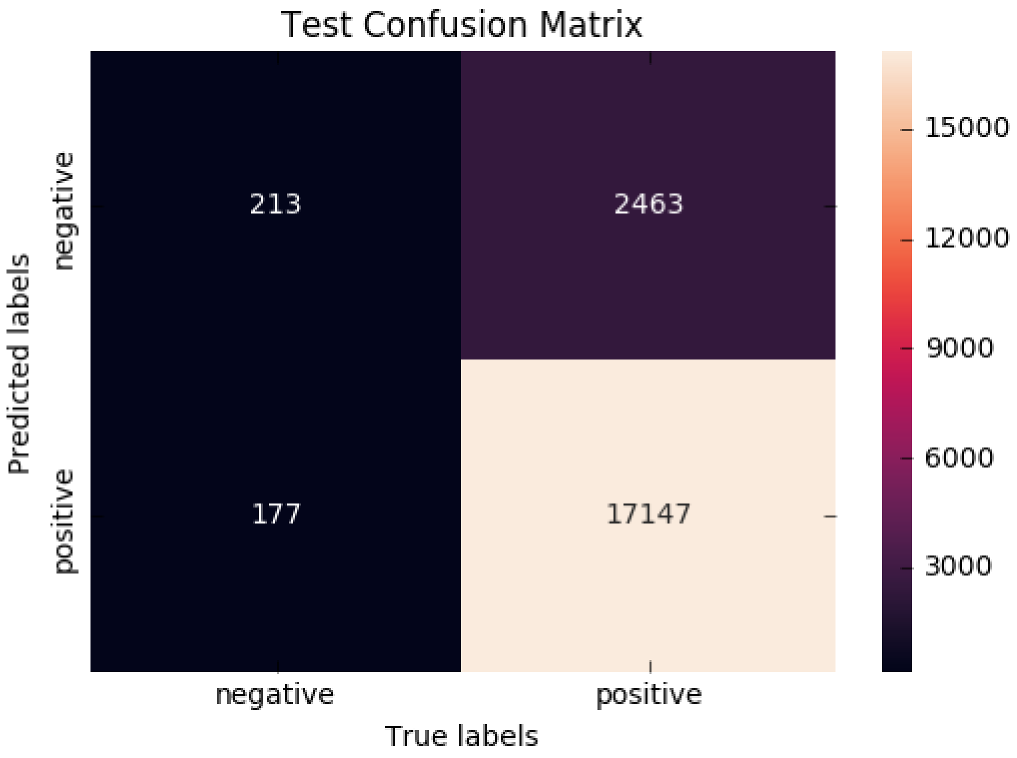
In [29]:

```
ax= plt.subplot()
sns.heatmap(bow_test_conf_matrix, annot=True, ax = ax, fmt='g')

ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Test Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[29]:

```
[<matplotlib.text.Text at 0x7f03c1d49588>,
 <matplotlib.text.Text at 0x7f03bc8bb978>]
```



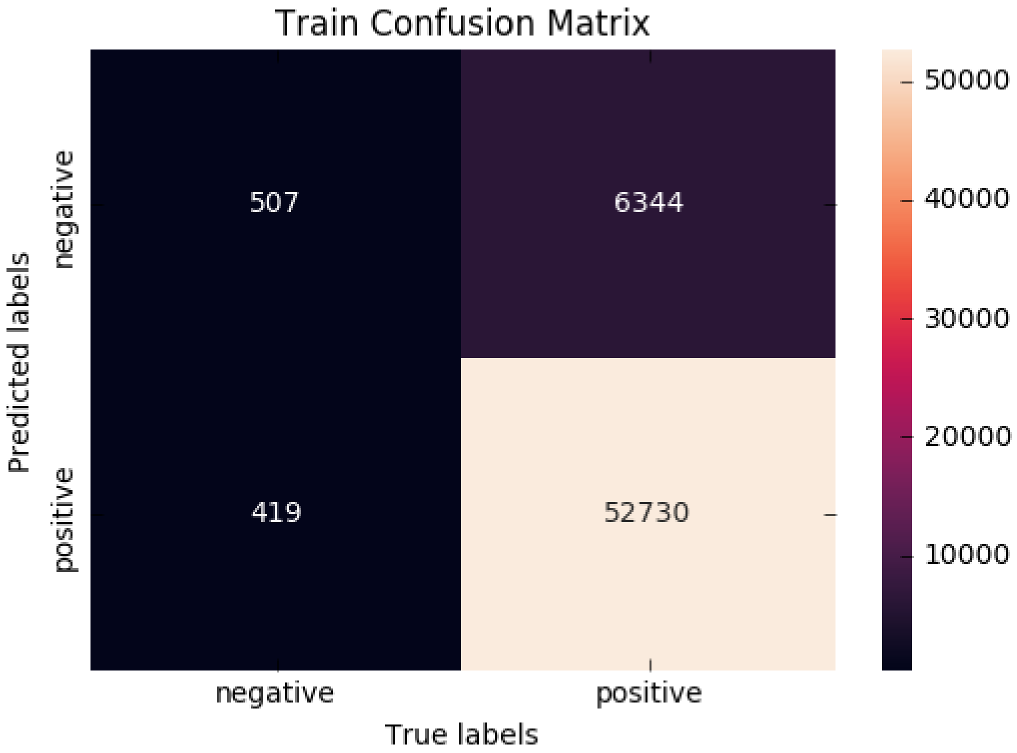
In [31]:

```
ax= plt.subplot()
sns.heatmap(bow_train_conf_matrix, annot=True, ax = ax, fmt='g')

ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Train Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[31]:

```
[<matplotlib.text.Text at 0x7f03c4891e10>,
 <matplotlib.text.Text at 0x7f03c48890b8>]
```



K-NN on tfi-idf (k-d tree)

In [17]:

```
import pickle
with open(r"tf_idf.pkl", "rb") as input_file:
    tfidf_dict = pickle.load(input_file)
```

In [20]:

```
from sklearn.neighbors import KNeighborsClassifier
from tqdm import tqdm

tfidf_auc_train = []
tfidf_auc_cv = []
tfidf_auc_test = []

for k_value in tqdm(range(1, 20, 2)):
    knn = KNeighborsClassifier(n_neighbors = k_value, algorithm='kd_tree', n_jobs = -1)
    knn.fit(tfidf_dict['train_tf_idf'],Y_train)

    train_proba = knn.score(tfidf_dict['train_tf_idf'], Y_train)
    tfidf_auc_train.append(train_proba)
    cv_proba = knn.score(tfidf_dict['cv_tf_idf'], Y_val)
    tfidf_auc_cv.append(cv_proba)
```

```
  0%|          | 0/10 [00:00<?, ?it/s]

 10%|█         | 1/10 [04:39<41:52, 279.15s/it]

 20%|██        | 2/10 [09:35<37:54, 284.31s/it]

 30%|███       | 3/10 [14:58<34:30, 295.80s/it]

 40%|████      | 4/10 [20:21<30:23, 303.95s/it]

 50%|█████     | 5/10 [25:43<25:46, 309.34s/it]

 60%|██████    | 6/10 [31:05<20:52, 313.24s/it]

 70%|███████   | 7/10 [36:28<15:48, 316.30s/it]

 80%|████████  | 8/10 [41:51<10:36, 318.14s/it]

 90%|█████████ | 9/10 [47:14<05:19, 319.77s/it]

100%|██████████| 10/10 [52:38<00:00, 320.87s/it]
```

In [21]:

```
k_vals = range(1, 20, 2)
train_k_dict = dict(zip(k_vals, tfidf_auc_train))
val_k_dict = dict(zip(k_vals, tfidf_auc_cv))
print(train_k_dict)
print(val_k_dict)
```

```
{19: 0.8886666666666667, 1: 1.0, 3: 0.9231666666666667, 17: 0.88943333333333
33, 5: 0.9076333333333333, 7: 0.90035, 9: 0.8958833333333334, 11: 0.89343333
33333333, 13: 0.8918666666666667, 15: 0.8904166666666666}
{19: 0.86495, 1: 0.8259, 3: 0.8629, 17: 0.8656, 5: 0.87125, 7: 0.87015, 9:
0.8697, 11: 0.86845, 13: 0.8676, 15: 0.86635}
```

In [26]:

```
tfidf_best_k = max(val_k_dict, key=val_k_dict.get)
tfidf_best_k
```

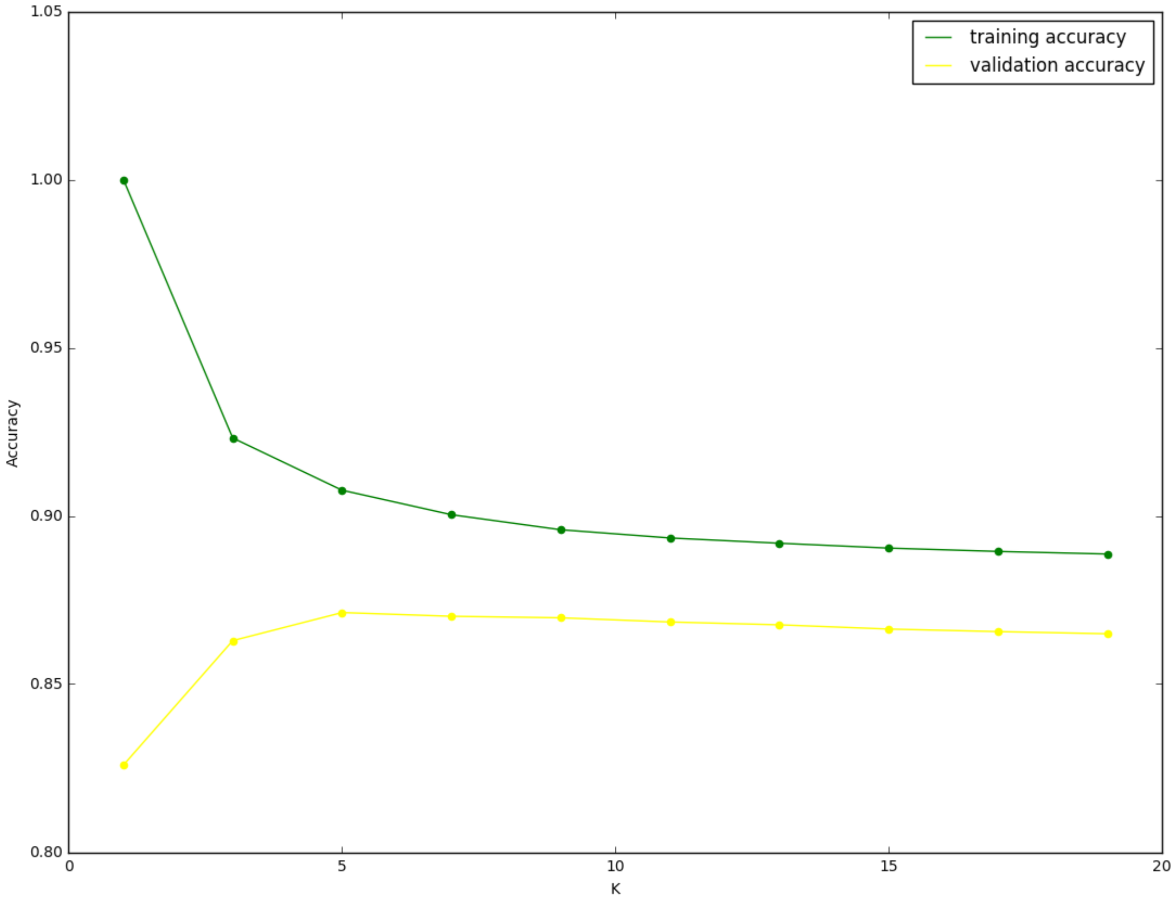
Out[26]:

```
5
```

In [25]:

```
plt.figure(figsize=(13, 10))
neighbors_settings = range(1, 20, 2)
plt.plot(neighbors_settings, tfidf_auc_train, label="training accuracy", color='green')
plt.plot(neighbors_settings, tfidf_auc_cv, label="validation accuracy", color='yellow')
plt.scatter(neighbors_settings, tfidf_auc_train, color='green')
plt.scatter(neighbors_settings, tfidf_auc_cv, color='yellow')
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



In [27]:

```
tfidf_knn = KNeighborsClassifier(n_neighbors = tfidf_best_k, algorithm='kd_tree', n_jobs=-1
tfidf_knn.fit(tfidf_dict['train_tf_idf'],Y_train)
test_predict_tfidf = tfidf_knn.predict_proba(tfidf_dict['test_tf_idf'])
train_predict_tfidf = tfidf_knn.predict_proba(tfidf_dict['train_tf_idf'])
tfidf_test_conf = tfidf_knn.predict(tfidf_dict['test_tf_idf'])
tfidf_train_conf = tfidf_knn.predict(tfidf_dict['train_tf_idf'])

print(type(train_predict_tfidf))
print(train_predict_tfidf[:, 1])
```

```
<class 'numpy.ndarray'>
[1. 1. 1. ... 1. 1. 1.]
```

In [28]:

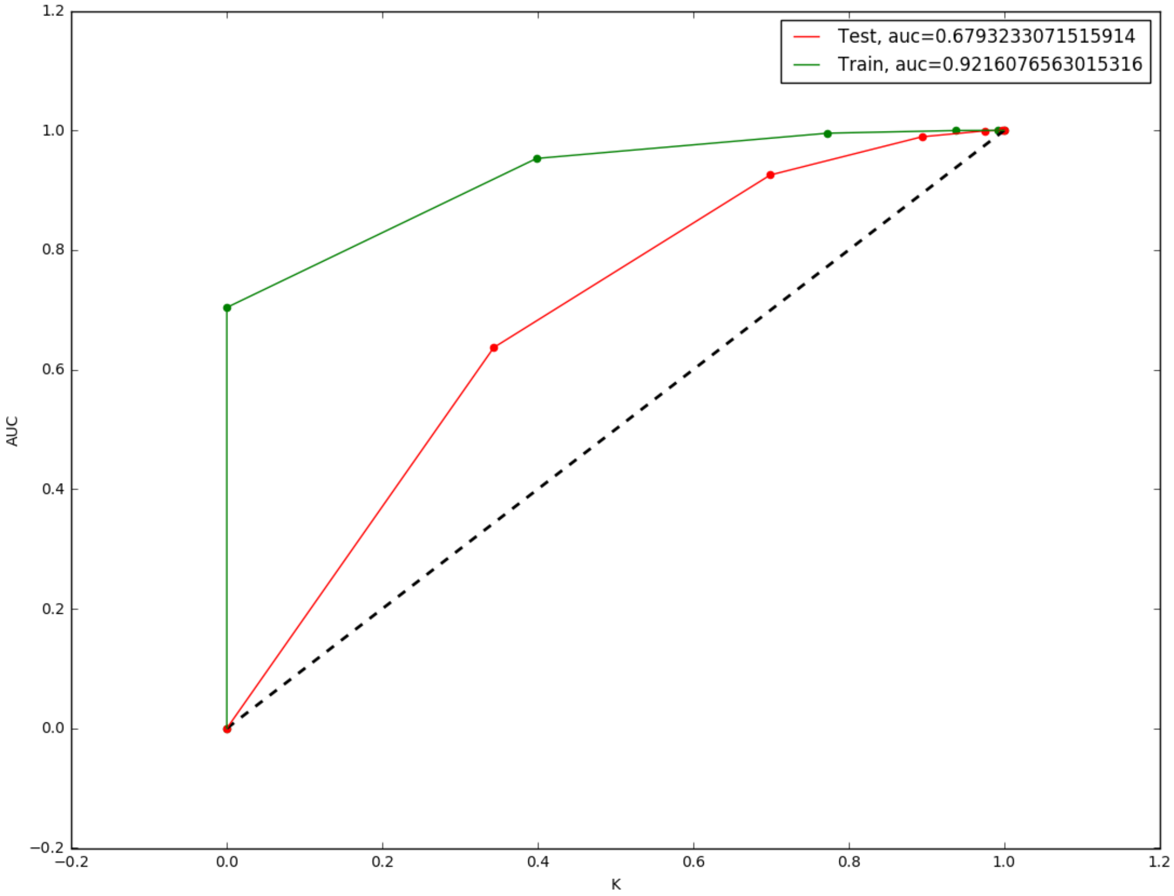
```
fpr_train, tpr_train, _ = roc_curve(Y_train, train_predict_tfidf[:, 1])
fpr_test, tpr_test, _ = roc_curve(Y_test, test_predict_tfidf[:, 1])
tfidf_test_auc = auc(fpr_test, tpr_test)
tfidf_train_auc = auc(fpr_train, tpr_train)
print(tfidf_test_auc)
print(tfidf_train_auc)
```

```
0.6793233071515914
0.9216076563015316
```

In [29]:

```
plt.figure(figsize=(13, 10))
plt.plot([0,1], [0,1], color='black', lw=2, linestyle='--')
plt.plot(fpr_test, tpr_test, label="Test, auc="+str(tfidf_test_auc), color = 'red')
plt.plot(fpr_train, tpr_train, label="Train, auc="+str(tfidf_train_auc), color = 'green')
plt.scatter(fpr_train, tpr_train, color = 'green')
plt.scatter(fpr_test, tpr_test, color = 'red')
plt.xlabel('K')
plt.ylabel('AUC')
plt.legend()

plt.show()
```



In [30]:

```
from sklearn.metrics import classification_report, confusion_matrix
tfidf_train_conf_matrix = confusion_matrix(Y_train, tfidf_train_conf)
tfidf_test_conf_matrix = confusion_matrix(Y_test, tfidf_test_conf)
class_report = classification_report(Y_test, tfidf_test_conf)
print(tfidf_test_conf_matrix)
print(class_report)
```

```
[[  283  2393]
 [  185 17139]]
              precision    recall  f1-score   support

           0       0.60      0.11      0.18      2676
           1       0.88      0.99      0.93     17324

   micro avg       0.87      0.87      0.87     20000
   macro avg       0.74      0.55      0.56     20000
weighted avg       0.84      0.87      0.83     20000
```

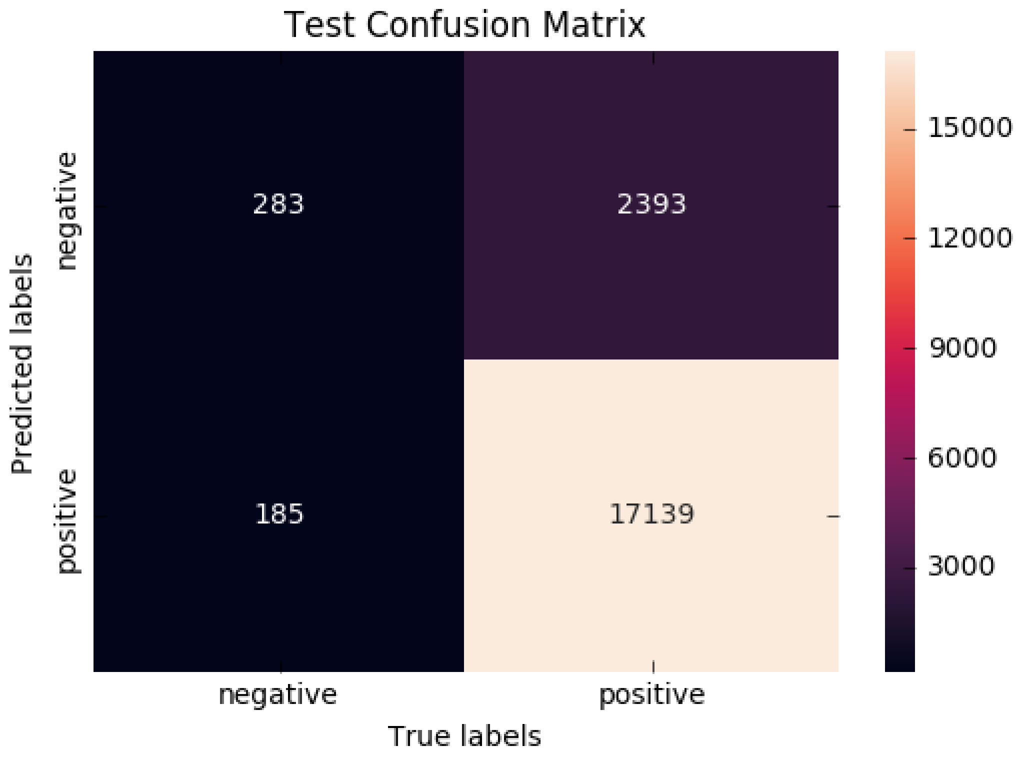
In [31]:

```
ax= plt.subplot()
sns.heatmap(tfidf_test_conf_matrix, annot=True, ax = ax, fmt='g')

ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Test Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[31]:

```
[<matplotlib.text.Text at 0x7f180d5b0908>,
 <matplotlib.text.Text at 0x7f180d4cfc50>]
```



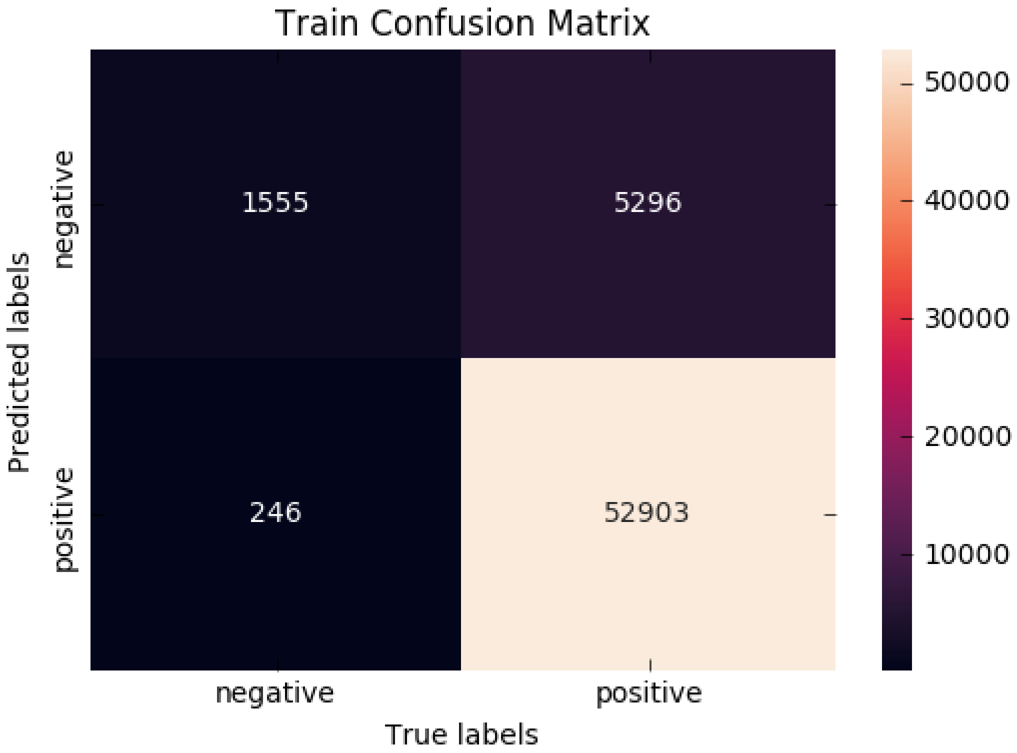
In [32]:

```
ax= plt.subplot()
sns.heatmap(tfidf_train_conf_matrix, annot=True, ax = ax, fmt='g')

ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Train Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[32]:

```
[<matplotlib.text.Text at 0x7f180928af98>,
 <matplotlib.text.Text at 0x7f18092a5438>]
```



K-NN on Avg-w2v(k-d tree)

In [55]:

```
import pickle
with open(r"avg_w2v.pkl", "rb") as input_file:
    avg_dict = pickle.load(input_file)
```

In [56]:

```
from sklearn.neighbors import KNeighborsClassifier

avg_auc_train = []
avg_auc_cv = []
avg_auc_test = []

for k_value in tqdm(range(1, 20, 2)):
    knn = KNeighborsClassifier(n_neighbors = k_value, algorithm='kd_tree', n_jobs = -1)
    knn.fit(avg_dict['X_train_avgw2v'],Y_train)

    train_proba = knn.score(avg_dict['X_train_avgw2v'], Y_train)
    avg_auc_train.append(train_proba)
    cv_proba = knn.score(avg_dict['X_val_avgw2v'], Y_val)
    avg_auc_cv.append(cv_proba)
```

```
  0%|          | 0/10 [00:00<?, ?it/s]

 10%|█         | 1/10 [00:52<07:49, 52.12s/it]

 20%|██        | 2/10 [04:19<13:10, 98.81s/it]

 30%|███       | 3/10 [07:55<15:37, 133.96s/it]

 40%|████      | 4/10 [11:31<15:51, 158.54s/it]

 50%|█████     | 5/10 [15:08<14:39, 175.86s/it]

 60%|██████    | 6/10 [18:47<12:36, 189.08s/it]

 70%|███████   | 7/10 [22:24<09:52, 197.38s/it]

 80%|████████  | 8/10 [26:05<06:49, 204.55s/it]

 90%|█████████ | 9/10 [29:47<03:29, 209.60s/it]

100%|██████████| 10/10 [33:28<00:00, 213.17s/it]
```

In [57]:

```
k_vals = range(1, 20, 2)
train_k_dict = dict(zip(k_vals, avg_auc_train))
val_k_dict = dict(zip(k_vals, avg_auc_cv))
print(train_k_dict)
print(val_k_dict)
```

```
{19: 0.91245, 1: 0.9999666666666667, 3: 0.9424833333333333, 17: 0.9139, 5:
0.9298666666666666, 7: 0.9231833333333334, 9: 0.9201333333333334, 11: 0.9184
333333333333, 13: 0.9164666666666667, 15: 0.9150333333333334}
{19: 0.8914, 1: 0.8647, 3: 0.8878, 17: 0.8922, 5: 0.8927, 7: 0.89305, 9: 0.8
9315, 11: 0.8931, 13: 0.8927, 15: 0.8921}
```

In [58]:

```
avg_best_k = max(val_k_dict, key=val_k_dict.get)
avg_best_k
```

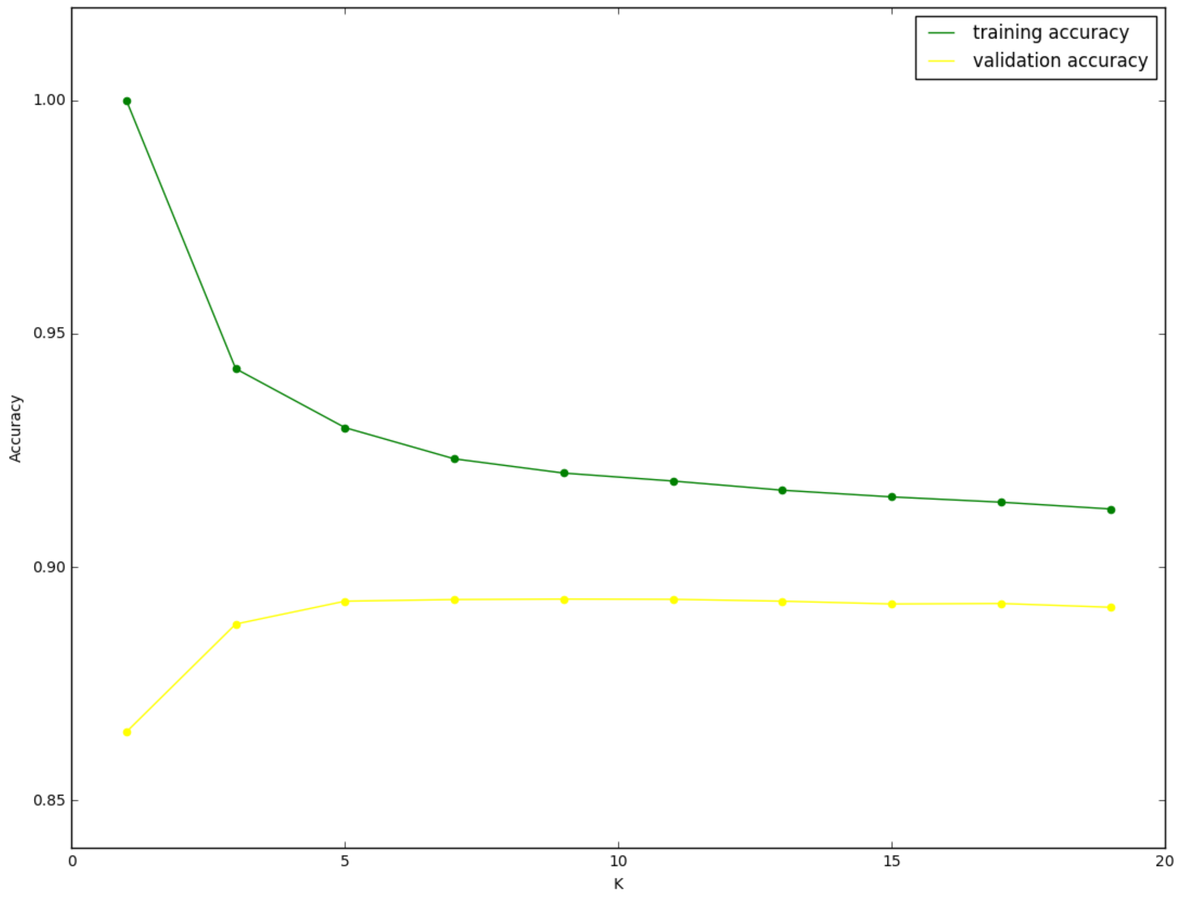
Out[58]:

```
9
```

In [60]:

```
plt.figure(figsize=(13, 10))
neighbors_settings = range(1, 20, 2)
plt.plot(neighbors_settings, avg_auc_train, label="training accuracy", color='green')
plt.plot(neighbors_settings, avg_auc_cv, label="validation accuracy", color='yellow')
plt.scatter(neighbors_settings, avg_auc_train, color='green')
plt.scatter(neighbors_settings, avg_auc_cv, color='yellow')
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



In [61]:

```
avg_knn = KNeighborsClassifier(n_neighbors = bow_best_k, algorithm='kd_tree', n_jobs=-1)
avg_knn.fit(avg_dict['X_train_avgw2v'],Y_train)
test_predict_avg = avg_knn.predict_proba(avg_dict['X_test_avgw2v'])
train_predict_avg = avg_knn.predict_proba(avg_dict['X_train_avgw2v'])

avg_test_conf = avg_knn.predict(avg_dict['X_test_avgw2v'])
avg_train_conf = avg_knn.predict(avg_dict['X_train_avgw2v'])

print(type(train_predict_avg))
print(train_predict_avg[:, 1])
```

```
<class 'numpy.ndarray'>
[0.8 1.  1.  ... 1.  1.  1. ]
```

In [63]:

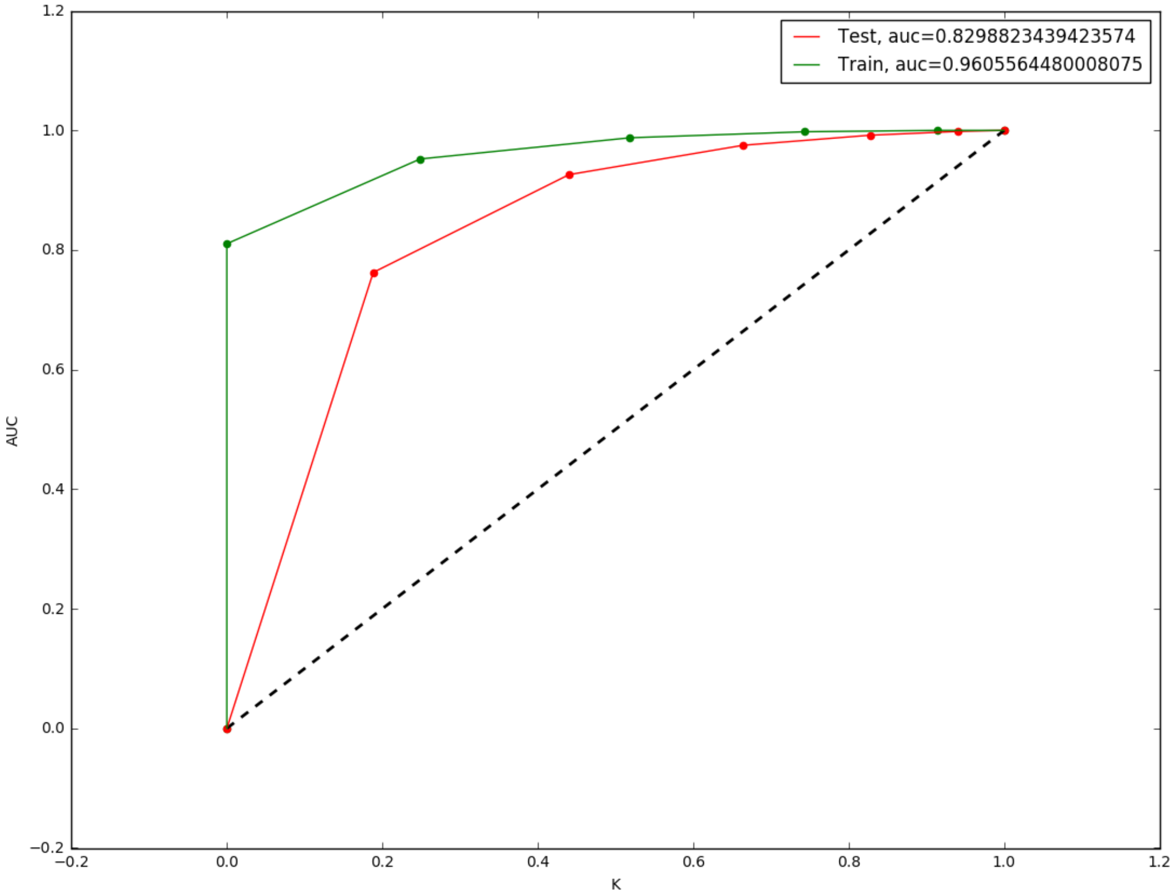
```
fpr_train, tpr_train, _ = roc_curve(Y_train, train_predict_avg[:, 1])
fpr_test, tpr_test, _ = roc_curve(Y_test, test_predict_avg[:, 1])
avg_test_auc = auc(fpr_test, tpr_test)
avg_train_auc = auc(fpr_train, tpr_train)
print(avg_test_auc)
print(avg_train_auc)
```

```
0.8298823439423574
0.9605564480008075
```

In [64]:

```
plt.figure(figsize=(13, 10))
plt.plot([0,1], [0,1], color='black', lw=2, linestyle='--')
plt.plot(fpr_test, tpr_test, label="Test, auc="+str(avg_test_auc), color = 'red')
plt.plot(fpr_train, tpr_train, label="Train, auc="+str(avg_train_auc), color = 'green')
plt.scatter(fpr_train, tpr_train, color = 'green')
plt.scatter(fpr_test, tpr_test, color = 'red')
plt.xlabel('K')
plt.ylabel('AUC')
plt.legend()

plt.show()
```



In [65]:

```
from sklearn.metrics import classification_report, confusion_matrix
avg_train_conf_matrix = confusion_matrix(Y_train, avg_train_conf)
avg_test_conf_matrix = confusion_matrix(Y_test, avg_test_conf)
class_report = classification_report(Y_test, avg_test_conf)
print(avg_test_conf_matrix)
print(class_report)
```

```
[[  899  1777]
 [  431 16893]]
              precision    recall  f1-score   support

           0       0.68      0.34      0.45      2676
           1       0.90      0.98      0.94     17324

   micro avg       0.89      0.89      0.89     20000
   macro avg       0.79      0.66      0.69     20000
weighted avg       0.87      0.89      0.87     20000
```

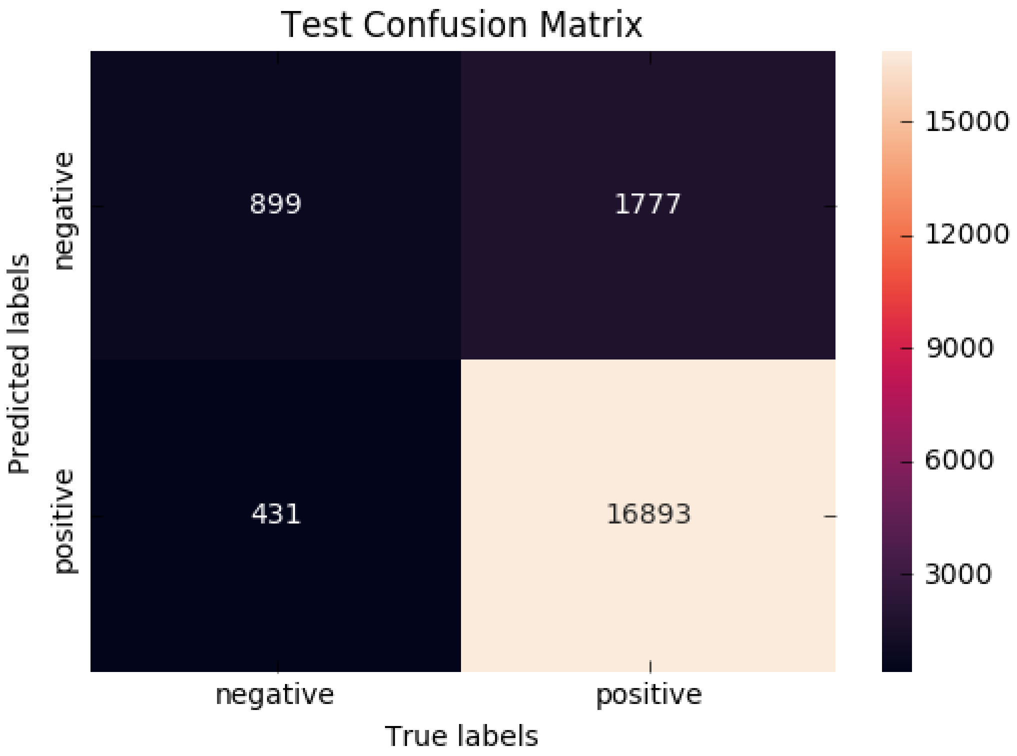
In [66]:

```
ax= plt.subplot()
sns.heatmap(avg_test_conf_matrix, annot=True, ax = ax, fmt='g')

ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Test Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[66]:

```
[<matplotlib.text.Text at 0x7f17b2019e80>,
 <matplotlib.text.Text at 0x7f17b074def0>]
```



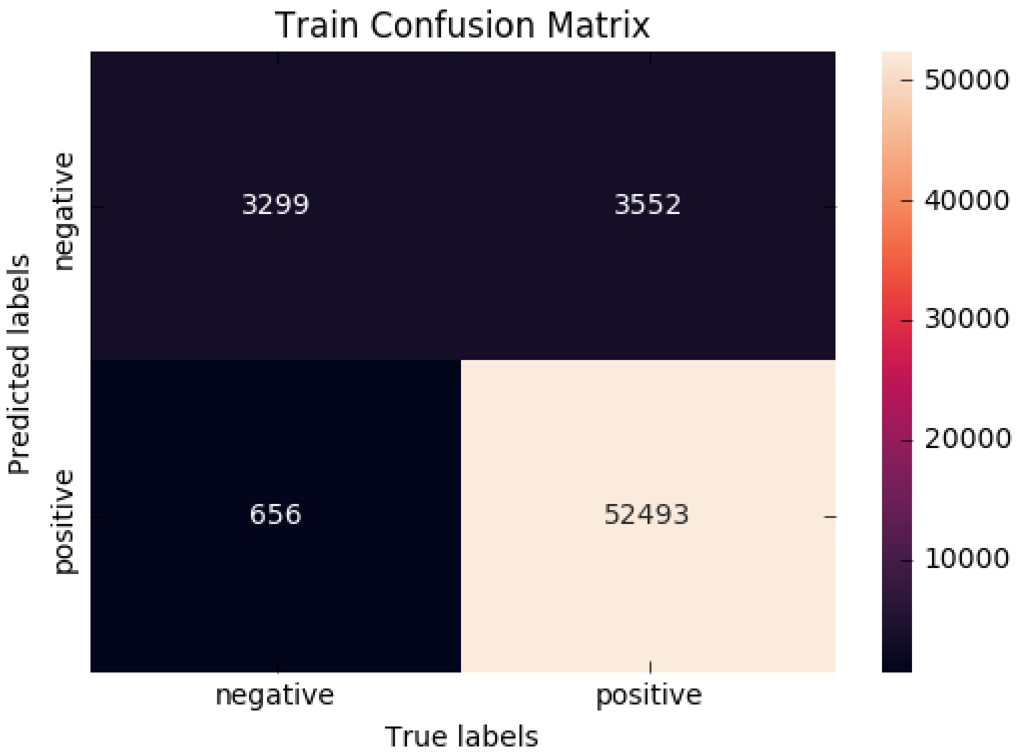
In [67]:

```
ax= plt.subplot()
sns.heatmap(avg_train_conf_matrix, annot=True, ax = ax, fmt='g')

ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Train Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[67]:

```
[<matplotlib.text.Text at 0x7f17b0f3df60>,
 <matplotlib.text.Text at 0x7f17b0f312b0>]
```



KNN on TFIDF weighted W2V (k-d tree)

In [75]:

```
import pickle
with open(r"tfidf_w2v.pkl", "rb") as input_file:
    tfidfw2v_dict = pickle.load(input_file)
```

In [76]:

```
from sklearn.neighbors import KNeighborsClassifier

tfidfw2v_auc_train = []
tfidfw2v_auc_cv = []
tfidfw2v_auc_test = []

for k_value in tqdm(range(1, 20, 2)):
    knn = KNeighborsClassifier(n_neighbors = k_value, algorithm='kd_tree', n_jobs = -1)
    knn.fit(tfidfw2v_dict['X_train_tfidfw2v'],Y_train)

    train_proba = knn.score(tfidfw2v_dict['X_train_tfidfw2v'], Y_train)
    tfidfw2v_auc_train.append(train_proba)
    cv_proba = knn.score(tfidfw2v_dict['X_val_tfidfw2v'], Y_val)
    tfidfw2v_auc_cv.append(cv_proba)
```

```
  0%|          | 0/10 [00:00<?, ?it/s]

 10%|█         | 1/10 [00:41<06:14, 41.57s/it]

 20%|██        | 2/10 [03:30<10:38, 79.78s/it]

 30%|███       | 3/10 [06:30<12:48, 109.82s/it]

 40%|████      | 4/10 [09:36<13:16, 132.67s/it]

 50%|█████     | 5/10 [12:41<12:22, 148.50s/it]

 60%|██████    | 6/10 [15:49<10:40, 160.18s/it]

 70%|███████   | 7/10 [18:58<08:27, 169.03s/it]

 80%|████████  | 8/10 [22:11<05:51, 176.00s/it]

 90%|█████████ | 9/10 [25:25<03:01, 181.35s/it]

100%|██████████| 10/10 [28:39<00:00, 185.41s/it]
```

In [77]:

```
k_vals = range(1, 20, 2)
train_k_dict = dict(zip(k_vals, tfidfw2v_auc_train))
val_k_dict = dict(zip(k_vals, tfidfw2v_auc_cv))
print(train_k_dict)
print(val_k_dict)
```

```
{19: 0.9080166666666667, 1: 0.9999666666666667, 3: 0.93635, 17: 0.9089, 5:
0.9240666666666667, 7: 0.9182833333333333, 9: 0.9152166666666667, 11: 0.9127
333333333333, 13: 0.9108166666666667, 15: 0.9095666666666666}
{19: 0.8854, 1: 0.85215, 3: 0.875, 17: 0.8864, 5: 0.8845, 7: 0.886, 9: 0.885
1, 11: 0.8858, 13: 0.8864, 15: 0.8873}
```

In [78]:

```
tfidfw2v_best_k = max(val_k_dict, key=val_k_dict.get)
tfidfw2v_best_k
```

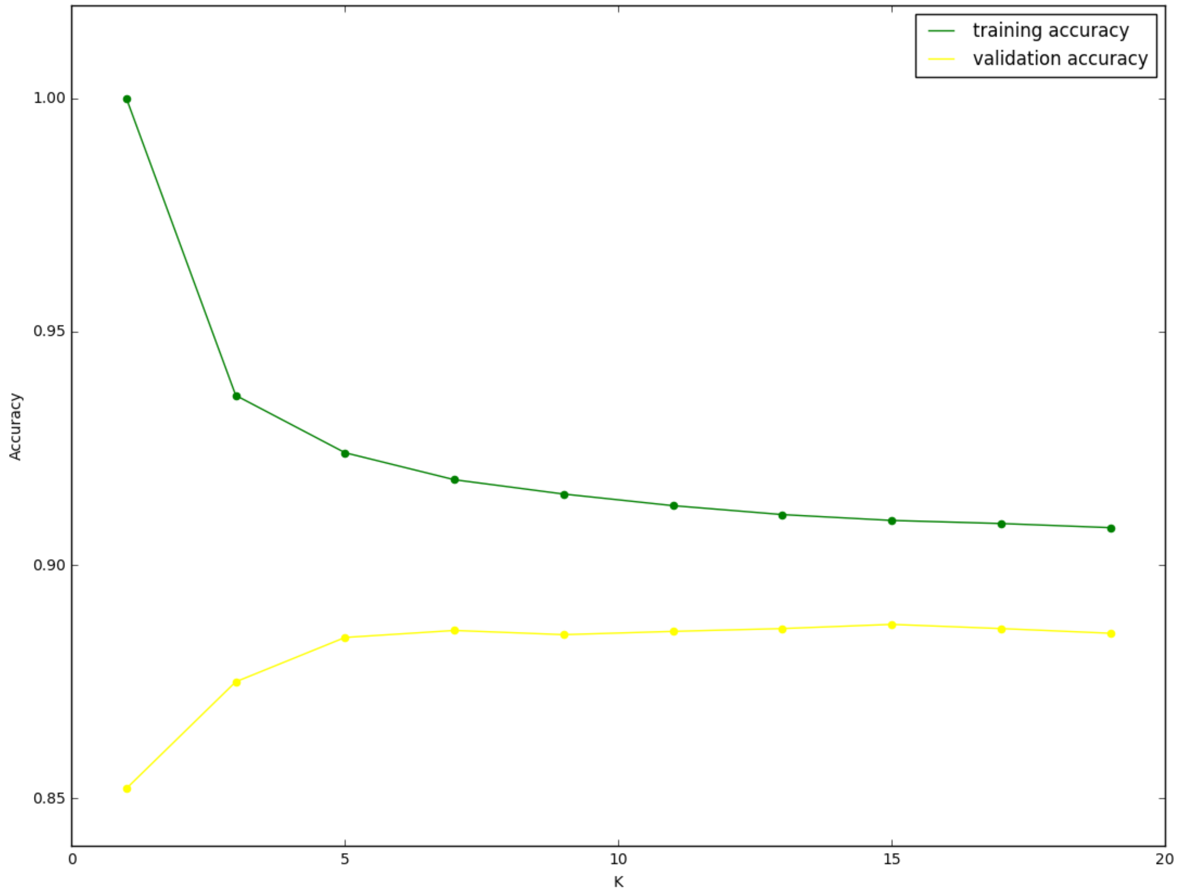
Out[78]:

```
15
```

In [79]:

```
plt.figure(figsize=(13, 10))
neighbors_settings = range(1, 20, 2)
plt.plot(neighbors_settings, tfidfw2v_auc_train, label="training accuracy", color='green')
plt.plot(neighbors_settings, tfidfw2v_auc_cv, label="validation accuracy", color='yellow')
plt.scatter(neighbors_settings, tfidfw2v_auc_train, color='green')
plt.scatter(neighbors_settings, tfidfw2v_auc_cv, color='yellow')
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



In [80]:

```
tfidfw2v_knn = KNeighborsClassifier(n_neighbors = tfidfw2v_best_k, algorithm='kd_tree', n_j
tfidfw2v_knn.fit(tfidfw2v_dict['X_train_tfidfw2v'],Y_train)
test_predict_tfidfw2v = avg_knn.predict_proba(tfidfw2v_dict['X_test_tfidfw2v'])
train_predict_tfidfw2v = avg_knn.predict_proba(tfidfw2v_dict['X_train_tfidfw2v'])

tfidfw2v_test_conf = tfidfw2v_knn.predict(tfidfw2v_dict['X_test_tfidfw2v'])
tfidfw2v_train_conf = tfidfw2v_knn.predict(tfidfw2v_dict['X_train_tfidfw2v'])

print(type(train_predict_tfidfw2v))
print(train_predict_tfidfw2v[:, 1])
```

```
<class 'numpy.ndarray'>
[0.8 1.  1.  ... 1.  1.  1. ]
```

In [81]:

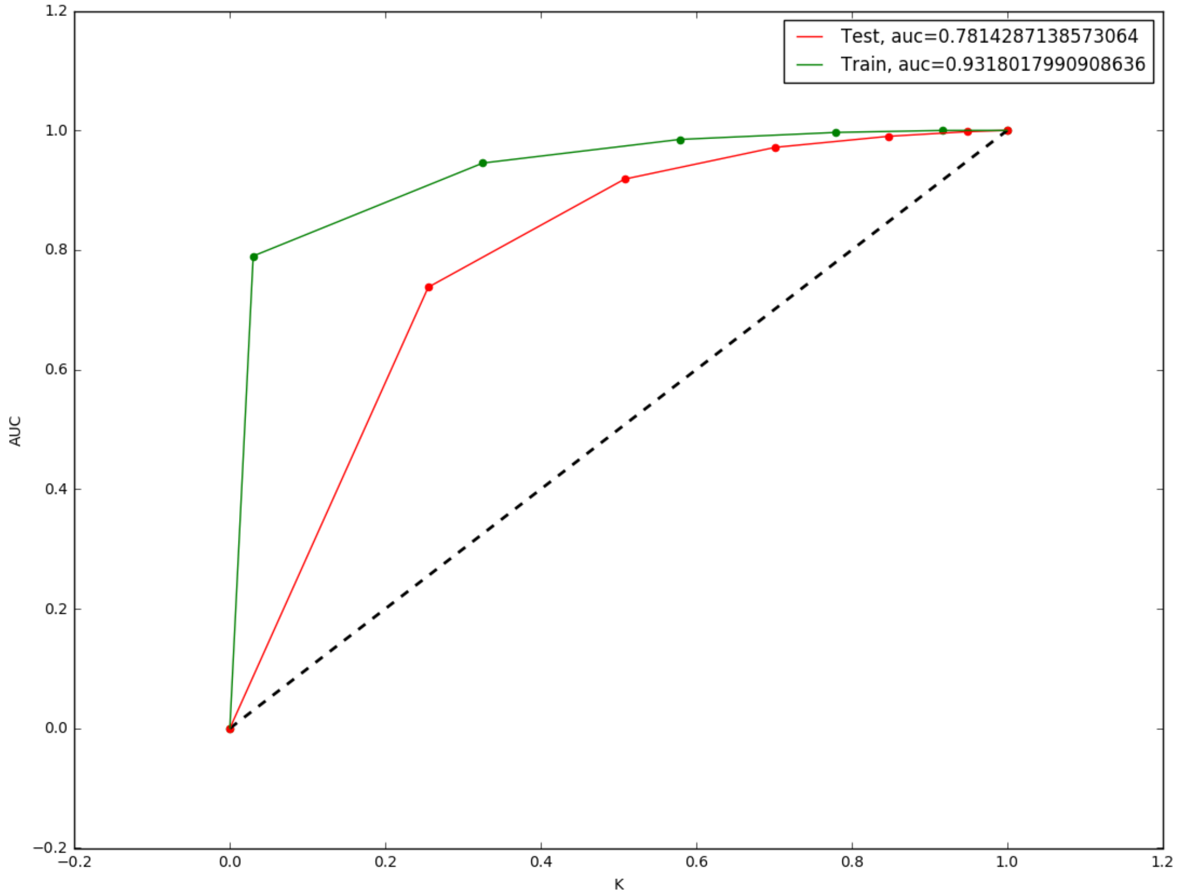
```
fpr_train, tpr_train, _ = roc_curve(Y_train, train_predict_tfidfw2v[:, 1])
fpr_test, tpr_test, _ = roc_curve(Y_test, test_predict_tfidfw2v[:, 1])
tfidfw2v_test_auc = auc(fpr_test, tpr_test)
tfidfw2v_train_auc = auc(fpr_train, tpr_train)
print(tfidfw2v_test_auc)
print(tfidfw2v_train_auc)
```

```
0.7814287138573064
0.9318017990908636
```

In [82]:

```
plt.figure(figsize=(13, 10))
plt.plot([0,1], [0,1], color='black', lw=2, linestyle='--')
plt.plot(fpr_test, tpr_test, label="Test, auc="+str(tfidfw2v_test_auc), color = 'red')
plt.plot(fpr_train, tpr_train, label="Train, auc="+str(tfidfw2v_train_auc), color = 'green'
plt.scatter(fpr_train, tpr_train, color = 'green')
plt.scatter(fpr_test, tpr_test, color = 'red')
plt.xlabel('K')
plt.ylabel('AUC')
plt.legend()

plt.show()
```



In [83]:

```
from sklearn.metrics import classification_report, confusion_matrix
tfidfw2v_train_conf_matrix = confusion_matrix(Y_train, tfidfw2v_train_conf)
tfidfw2v_test_conf_matrix = confusion_matrix(Y_test, tfidfw2v_test_conf)
class_report = classification_report(Y_test, tfidfw2v_test_conf)
print(tfidfw2v_test_conf_matrix)
print(class_report)
```

```
[[  630  2046]
 [  224 17100]]
              precision    recall  f1-score   support

           0       0.74      0.24      0.36      2676
           1       0.89      0.99      0.94     17324

   micro avg       0.89      0.89      0.89     20000
   macro avg       0.82      0.61      0.65     20000
weighted avg       0.87      0.89      0.86     20000
```

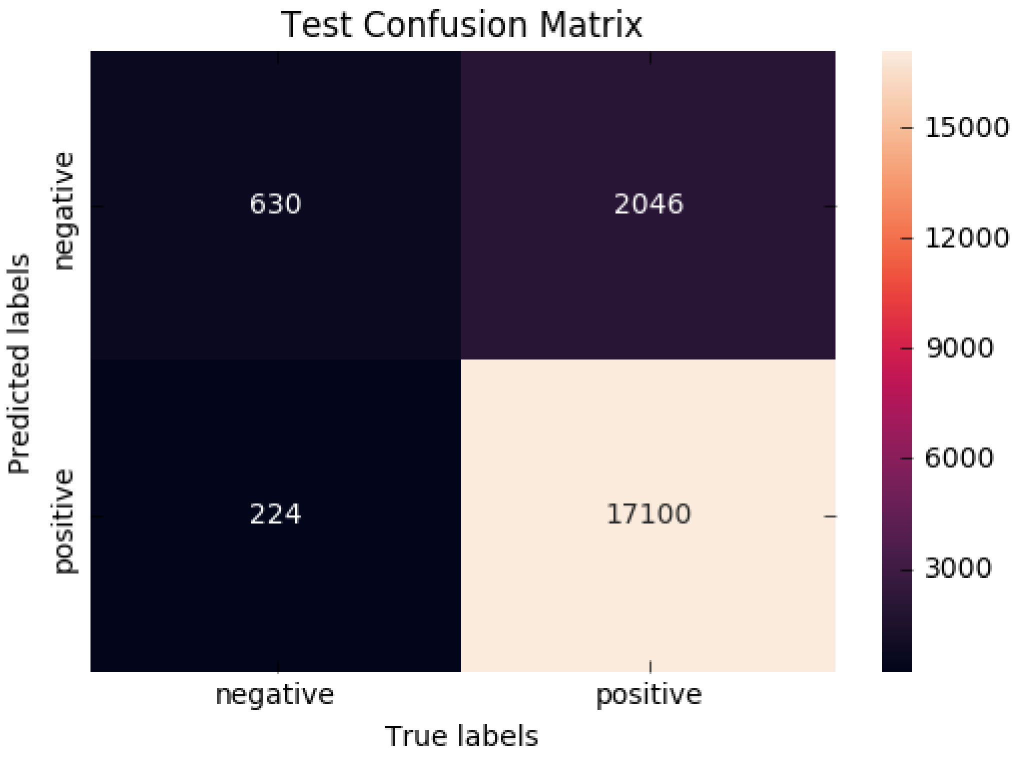
In [84]:

```
ax= plt.subplot()
sns.heatmap(tfidfw2v_test_conf_matrix, annot=True, ax = ax, fmt='g')

ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Test Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[84]:

```
[<matplotlib.text.Text at 0x7f180ec8a160>,
 <matplotlib.text.Text at 0x7f17ac4fb9e8>]
```



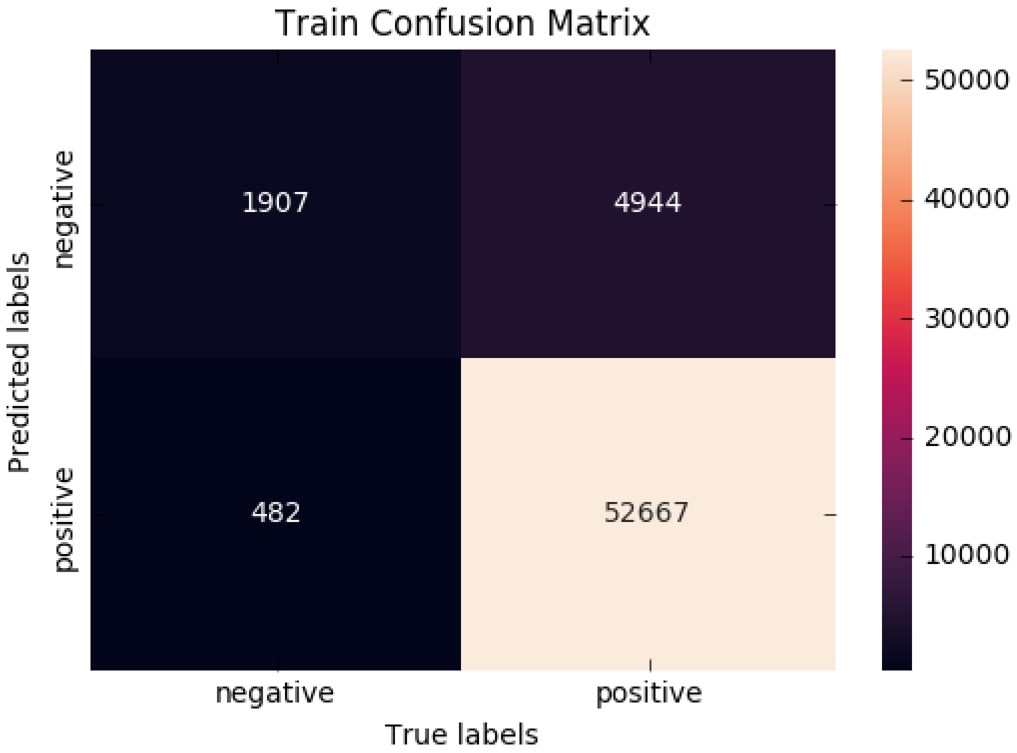
In [86]:

```
ax= plt.subplot()
sns.heatmap(tfidfw2v_train_conf_matrix, annot=True, ax = ax, fmt='g')

ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Train Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[86]:

```
[<matplotlib.text.Text at 0x7f17d3a6c358>,
 <matplotlib.text.Text at 0x7f1807602240>]
```



In [ ]:

```
```