In [1]:

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


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

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

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

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

from tqdm import tqdm
import os
```

In [3]:

```
#mounting the dataset from drive
# from google.colab import drive
# drive.mount('/content/gdrive')

#connecting to sqlite db
con = sqlite3.connect('database.sqlite')

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

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

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

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

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

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

Out[3]:

	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 [4]:

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

In [5]:

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

```
(80668, 7)
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	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 [6]:

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

```
(364173, 10)
```

In [7]:

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

Out[7]:

```
69.25890143662969
```

In [8]:

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

In [9]:

```
#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[9]:

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

In [10]:

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

Out[10]:

```
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 [11]:

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

Out[11]:

```
0    52
1    34
2    98
3    43
4    29
Name: lengthOfReview, dtype: int64
```

In [10]:

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

100%|██████████| 364171/364171 [00:00<00:00, 447313.57it/s]

In [11]:

```
#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, 452270.97it/s]

In [12]:

```
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 [01:49<00:00, 3330.00it/s]

In [13]:

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

364171

In [14]:

```
# 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 [15]:

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

100%|██████████| 364171/364171 [00:05<00:00, 65510.16it/s]

In [16]:

```
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 [00:22<00:00, 16465.51it/s]

In [17]:

```
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 [00:12<00:00, 29401.19it/s]

In [18]:

```
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 [19]:

```
# 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 [02:44<00:00, 2216.92it/s]

In [20]:

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

364171

Out[20]:

'satisfied product advertised use cereal raw vinegar general sweetner'

In [21]:

```
final['Preprocessed_text'] = preprocessed_reviews
```

In [22]:

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

364171

Out[22]:

	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 [2]:

```
dir_path = os.getcwd()
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))
X_len = filtered_data['lengthOfReview']
```

```
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 [74]:

```
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)
feature_names = count_vect.get_feature_names()
# BoW_dict = {'X_train_vect':X_train_vect, 'X_test_vect': X_test_vect, 'X_val_vect': X_val_
print(X_train_vect.shape)
# print(feature_names)
```

```
(60000, 47535)
```

In [25]:

```
X_train_vect.shape
```

Out[25]:

```
(60000, 47535)
```

In [26]:

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

Out[26]:

```
364171
```

In [27]:

```
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 [28]:

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

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

In [29]:

```
print(len(feature_names))
```

```
47535
```

In [30]:

```
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 [ ]:

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

[4.3] TF-IDF

In [101]:

```
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 [32]:

```
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 [33]:

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

In [ ]:

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

[4.4] Word2Vec

In [34]:

```
# 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 [35]:

```
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
```

```
[('terrific', 0.8565828204154968), ('excellent', 0.8381140828132629), ('fant
astic', 0.8366681337356567), ('awesome', 0.7857832908630371), ('wonderful',
0.7829444408416748), ('good', 0.742619514465332), ('perfect', 0.717479586601
2573), ('nice', 0.6593438386917114), ('fabulous', 0.6570981740951538), ('inc
redible', 0.6524804830551147)]
==================================================
[('greatest', 0.7822151780128479), ('best', 0.7523022294044495), ('tasties
t', 0.6484744548797607), ('coolest', 0.6170215606689453), ('terrible', 0.612
8978729248047), ('awful', 0.6031897664070129), ('nicest', 0.598495066165924
1), ('nastiest', 0.5957451462745667), ('closest', 0.5847468376159668), ('sof
test', 0.5774857401847839)]
```

In [36]:

```
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  ['flat', 'mater', 'elements', 'crock', 'tripe', 'reversed', 'l
actaid', 'capsule', 'easiest', 'clarify', 'pees', 'swore', 'similiar', 'powd
ery', 'cement', 'deb', 'burned', 'seasonally', 'stove', 'reinforcement', 'co
nfusion', 'sky', 'mama', 'evil', 'contrast', 'start', 'booklet', 'moves', 'c
hestnuts', 'virtuous', 'monitors', 'twain', 'liquified', 'recommendations',
'quinoa', 'micro', 'corned', 'celebrated', 'pitcher', 'clip', 'movie', 'hfc
s', 'single', 'leftover', 'inhaled', 'impulse', 'leak', 'gag', 'farming', 'b
razilian']
```

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

[4.4.1.1] Avg W2v

In [37]:

```
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 [38]:

```
# 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 tqdm(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 [39]:

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

```
20000
```

In [22]:

```
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])
```

In [ ]:

```
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 [ ]:

```
import pickle
with open('/content/gdrive/My Drive/Colab Notebooks/Assignment 3/avg_w2v.pkl', 'wb') as har
    pickle.dump(Avg_w2v_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

[4.4.1.2] TFIDF weighted W2v

In [79]:

```
# 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 [ ]:

```
# 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 tqdm(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 [73]:

```
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 [74]:

```
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 [75]:

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

In [17]:

```
from sklearn import tree
from tqdm import tqdm
```

Decision Trees on BoW

In [75]:

```
import pickle
# with open(r"/content/gdrive/My Drive/Colab Notebooks/Assignment 4/BoW.pkl", "rb") as inpu
with open(r"BoW.pkl", "rb") as input_file:
    BoW_dict = pickle.load(input_file)
```

In [32]:

```
bow_train_score_list = []
bow_val_score_list = []
fpr = dict()
tpr = dict()
fpr_val = dict()
tpr_val = dict()
roc_auc_train = dict()
roc_auc_val = dict()
roc_auc_test = dict()
for depth_val in tqdm([1, 5, 10, 50, 100, 500, 1000]):
    dt_clf = tree.DecisionTreeClassifier(max_depth=depth_val)
    dt_clf.fit(BoW_dict['X_train_vect'],Y_train)
    train_proba = dt_clf.predict_proba(BoW_dict['X_train_vect'])
    val_proba = dt_clf.predict_proba(BoW_dict['X_val_vect'])

    fpr[depth_val], tpr[depth_val], _ = roc_curve(Y_train, train_proba[:,1])
    roc_auc_train[depth_val] = auc(fpr[depth_val], tpr[depth_val])
    bow_train_score_list.append(auc(fpr[depth_val], tpr[depth_val]))
    fpr_val[depth_val], tpr_val[depth_val], _ = roc_curve(Y_val, val_proba[:,1])
    roc_auc_val[depth_val] = auc(fpr_val[depth_val], tpr_val[depth_val])
    bow_val_score_list.append(auc(fpr_val[depth_val], tpr_val[depth_val]))
print(roc_auc_train)
print(roc_auc_val)
```

```
  0%|
| 0/7 [00:00<?, ?it/s]

 14%|
| 1/7 [00:00<00:01,  3.77it/s]

 29%|
| 2/7 [00:00<00:02,  2.08it/s]

 43%|
| 3/7 [00:03<00:04,  1.24s/it]

 57%|
| 4/7 [0
0:24<00:18,  6.02s/it]

 71%|
| 5/7 [0
0:54<00:21, 10.87s/it]

 86%|
| 6/7 [0
1:34<00:15, 15.73s/it]

100%|
| 7/7 [0
2:08<00:00, 18.38s/it]


{1: 0.63072296875601919, 50: 0.96736236677570198, 100: 0.99078409867958128,
5: 0.70550021367869997, 1000: 1.0, 500: 1.0, 10: 0.81238447970823235}
{1: 0.63665589422735125, 50: 0.7067480373581978, 100: 0.68836748885924326,
```

```
5: 0.70952653177802028, 1000: 0.71947991892962238, 500: 0.7200477697536396,
10: 0.78715341627908864}
```

In [33]:

```
bow_best_depth = max(roc_auc_val, key=roc_auc_val.get)
bow_best_depth
```

Out[33]:

```
10
```

In [34]:

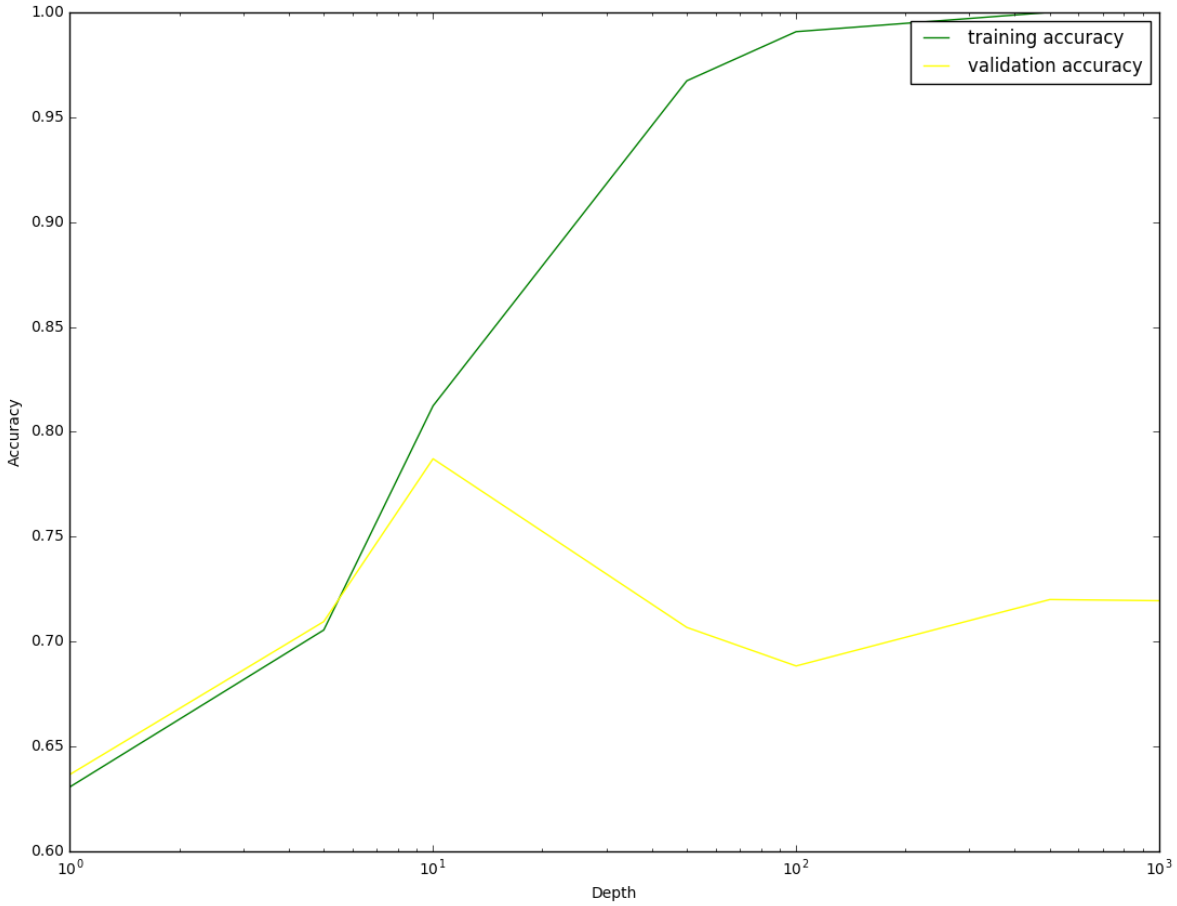
```
print(bow_train_score_list)
print(bow_val_score_list)
```

```
[0.63072296875601919, 0.70550021367869997, 0.81238447970823235, 0.9673623667
7570198, 0.99078409867958128, 1.0, 1.0]
[0.63665589422735125, 0.70952653177802028, 0.78715341627908864, 0.7067480373
581978, 0.68836748885924326, 0.7200477697536396, 0.71947991892962238]
```

In [35]:

```
import pylab
plt.figure(figsize=(13, 10))
neighbors_settings = [1, 5, 10, 50, 100, 500, 1000]
plt.plot(neighbors_settings, bow_train_score_list, label="training accuracy", color='green'
plt.plot(neighbors_settings, bow_val_score_list, label="validation accuracy", color='yellow
plt.xlabel('Depth')
plt.ylabel('Accuracy')
plt.legend()
plt.xscale('log')

plt.show()
```



Important Features

In [48]:

```
print("Top 20 Important Features")
d = sorted(list(zip(count_vect.get_feature_names(), dt_clf.feature_importances_ )), key=lam
d
```

```
Top 20 Important Features
```

Out[48]:

```
[('not', 0.1474960333555736),
 ('great', 0.11318978181638117),
 ('best', 0.086486011064321452),
 ('disappointed', 0.072570256092524849),
 ('worst', 0.064499980093152345),
 ('awful', 0.048825978626978093),
 ('terrible', 0.046148668494813437),
 ('delicious', 0.044232847892431884),
 ('horrible', 0.043618937022260683),
 ('disappointing', 0.032090872187176059),
 ('money', 0.028141034289785895),
 ('yuck', 0.027563205456815391),
 ('good', 0.025164483750128877),
 ('love', 0.025123253160794067),
 ('excellent', 0.014857872896807881),
 ('perfect', 0.012222012425145021),
 ('wonderful', 0.0057978353873340696),
 ('waste', 0.0056985143161600305),
 ('nice', 0.005675880570838056),
 ('refund', 0.0049193572184558264)]
```

In [36]:

```
dt_clf = tree.DecisionTreeClassifier(max_depth=bow_best_depth)
dt_clf.fit(BoW_dict['X_train_vect'],Y_train)
bow_test_proba = dt_clf.predict_proba(BoW_dict['X_test_vect'])
bow_train_proba = dt_clf.predict_proba(BoW_dict['X_train_vect'])
bow_test_proba
```

Out[36]:

```
array([[ 0.01698429,  0.98301571],
       [ 0.10073385,  0.89926615],
       [ 0.0677556 ,  0.9322444 ],
       ...,
       [ 0.10073385,  0.89926615],
       [ 0.01698429,  0.98301571],
       [ 0.        ,  1.        ]])
```

In [37]:

```
bow_fpr_train, bow_tpr_train, _ = roc_curve(Y_train, bow_train_proba[:, 1])
bow_fpr_test, bow_tpr_test, _ = roc_curve(Y_test, bow_test_proba[:, 1])
bow_test_auc = auc(bow_fpr_test, bow_tpr_test)
bow_train_auc = auc(bow_fpr_train, bow_tpr_train)
print(bow_test_auc)
print(bow_train_auc)
```

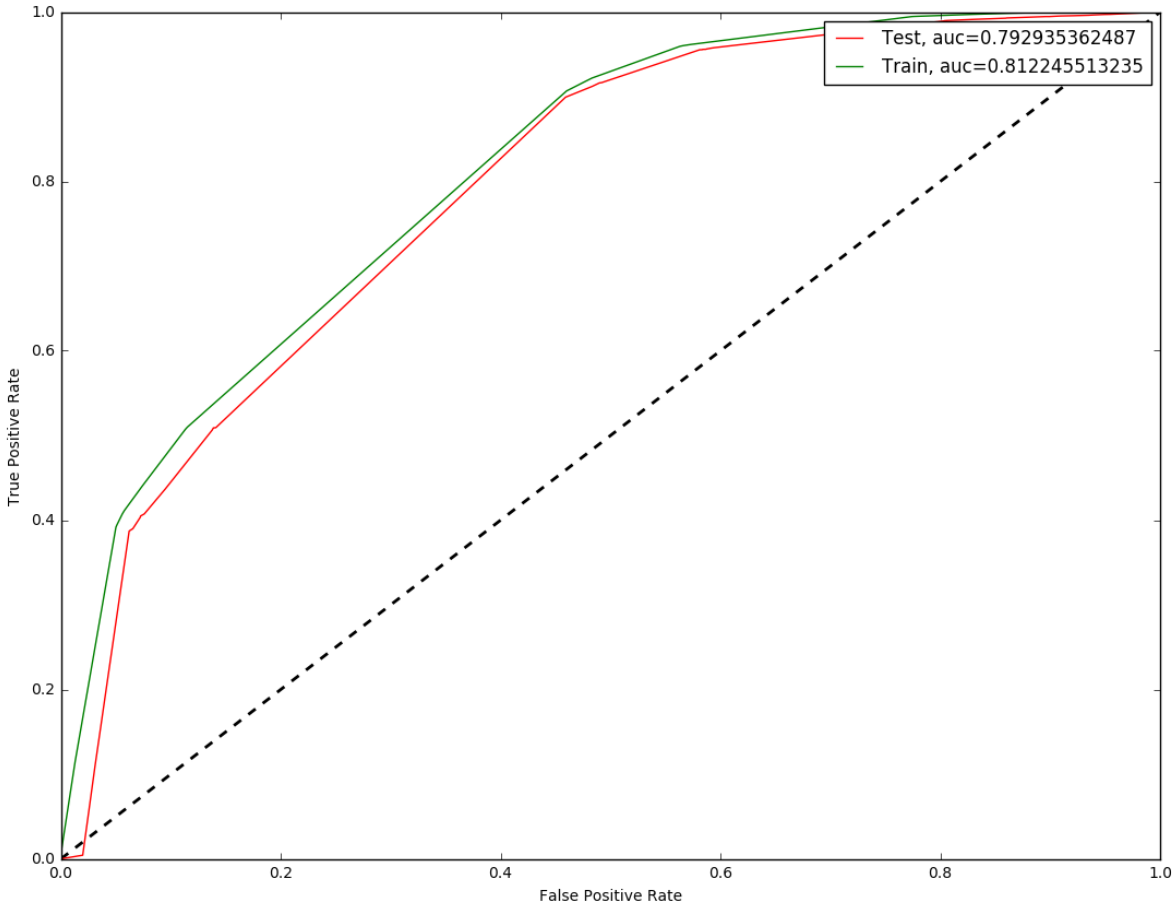
```
0.792935362487
0.812245513235
```

In [38]:

```
import pylab
plt.figure(figsize=(13, 10))
plt.plot([0,1], [0,1], color='black', lw=2, linestyle='--')
plt.plot(bow_fpr_test, bow_tpr_test, label="Test, auc="+str(bow_test_auc), color = 'red')
plt.plot(bow_fpr_train, bow_tpr_train, label="Train, auc="+str(bow_train_auc), color = 'gre

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()

plt.show()
```



In [40]:

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

In [41]:

```
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)
```

```
[[  529  2147]
 [  185 17139]]
             precision    recall  f1-score   support

          0       0.74      0.20      0.31      2676
          1       0.89      0.99      0.94     17324

avg / total       0.87      0.88      0.85     20000
```

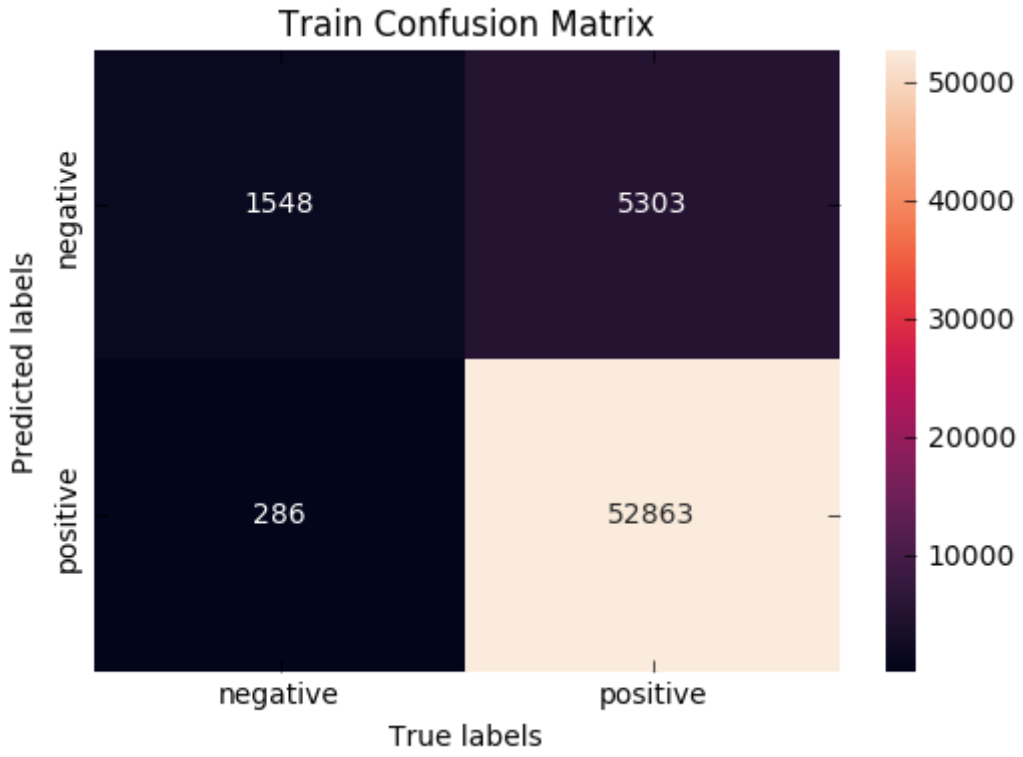
In [42]:

```
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[42]:

```
[<matplotlib.text.Text at 0x11033748>, <matplotlib.text.Text at 0x10e095c0>]
```



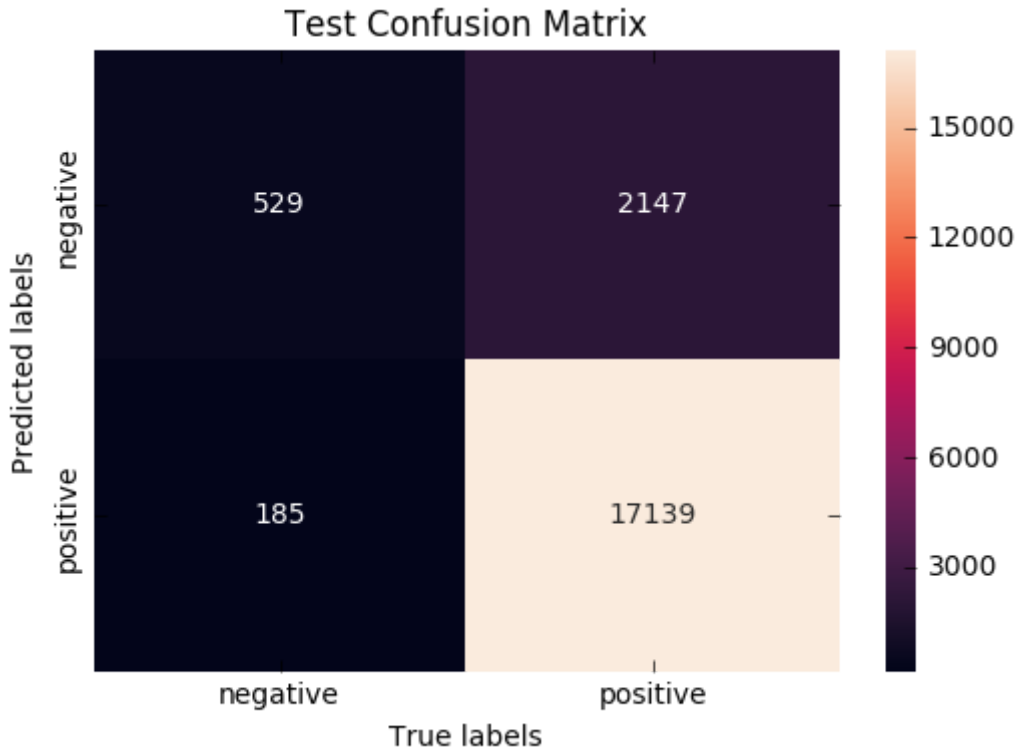
In [43]:

```
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[43]:

```
[<matplotlib.text.Text at 0x24dee2e8>, <matplotlib.text.Text at 0xf7cf080>]
```



Exporting Decision Tree

In [76]:

```
tclassifier = tree.DecisionTreeClassifier(max_depth = 3)
tclassifier.fit(BoW_dict['X_train_vect'],Y_train)
```

Out[76]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3,
            max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best')
```

In [ ]:

```
from graphviz import Source
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'
graph = Source(tree.export_graphviz(tclassifier, out_file='tree.dot', feature_names=count_v
png_bytes = graph.pipe(format='png')
with open('bow.png','wb') as f:
    f.write(png_bytes)
```

Decision Tree on Tf-IDF

In [96]:

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

In [97]:

```
tfidf_train_score_list = []
tfidf_val_score_list = []
fpr = dict()
tpr = dict()
fpr_val = dict()
tpr_val = dict()
roc_auc_train = dict()
roc_auc_val = dict()
roc_auc_test = dict()
for depth_val in tqdm([1, 5, 10, 50, 100, 500, 1000]):
    dt_clf = tree.DecisionTreeClassifier(max_depth=depth_val)
    dt_clf.fit(tfidf_dict['train_tf_idf'],Y_train)
    train_proba = dt_clf.predict_proba(tfidf_dict['train_tf_idf'])
    val_proba = dt_clf.predict_proba(tfidf_dict['cv_tf_idf'])

    fpr[depth_val], tpr[depth_val], _ = roc_curve(Y_train, train_proba[:,1])
    roc_auc_train[depth_val] = auc(fpr[depth_val], tpr[depth_val])
    tfidf_train_score_list.append(auc(fpr[depth_val], tpr[depth_val]))
    fpr_val[depth_val], tpr_val[depth_val], _ = roc_curve(Y_val, val_proba[:,1])
    roc_auc_val[depth_val] = auc(fpr_val[depth_val], tpr_val[depth_val])
    tfidf_val_score_list.append(auc(fpr_val[depth_val], tpr_val[depth_val]))
print(roc_auc_train)
print(roc_auc_val)
```

```
  0%|
| 0/7 [00:00<?, ?it/s]

 14%|
| 1/7 [00:01<00:07,  1.22s/it]

 29%|
| 2/7 [00:03<00:07,  1.59s/it]

 43%|
| 3/7 [00:07<00:10,  2.67s/it]

 57%|
                                                                | 4/7 [0
0:36<00:27,  9.13s/it]

 71%|
                                                                | 5/7 [0
1:20<00:32, 16.14s/it]

 86%|
                                                                | 6/7 [0
2:22<00:23, 23.69s/it]

100%|
                                                               | 7/7 [0
3:19<00:00, 28.48s/it]


{1: 0.63150924254747764, 50: 0.96730711221652399, 100: 0.99138311335700413,
5: 0.70266623110784354, 1000: 1.0, 500: 1.0, 10: 0.81711210532547496}
{1: 0.63287523433772475, 50: 0.73488383345092467, 100: 0.73349733172625298,
```

```
5: 0.69818905048111035, 1000: 0.73713871103776307, 500: 0.73792163895162677,
10: 0.78437271491151594}
```

In [98]:

```
tfidf_best_depth = max(roc_auc_val, key=roc_auc_val.get)
tfidf_best_depth
```

Out[98]:

```
10
```

In [99]:

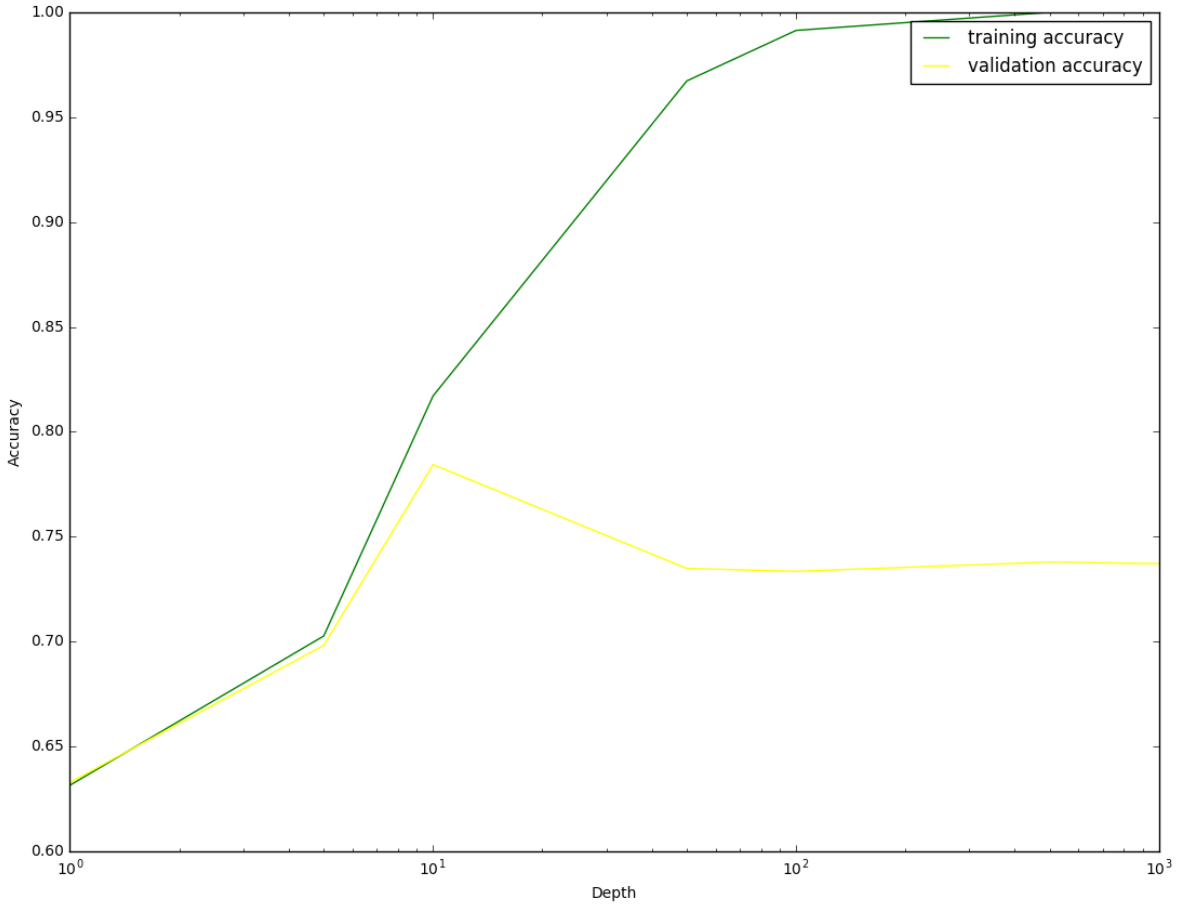
```
print(tfidf_train_score_list)
print(tfidf_val_score_list)
```

```
[0.63150924254747764, 0.70266623110784354, 0.81711210532547496, 0.9673071122
1652399, 0.99138311335700413, 1.0, 1.0]
[0.63287523433772475, 0.69818905048111035, 0.78437271491151594, 0.7348838334
5092467, 0.73349733172625298, 0.73792163895162677, 0.73713871103776307]
```

In [100]:

```
import pylab
plt.figure(figsize=(13, 10))
neighbors_settings = [1, 5, 10, 50, 100, 500, 1000]
plt.plot(neighbors_settings, tfidf_train_score_list, label="training accuracy", color='gree
plt.plot(neighbors_settings, tfidf_val_score_list, label="validation accuracy", color='yell
plt.xlabel('Depth')
plt.ylabel('Accuracy')
plt.legend()
plt.xscale('log')

plt.show()
```



In [102]:

```
print("Top 20 Important Features")
d = sorted(list(zip(tf_idf_vect.get_feature_names(), dt_clf.feature_importances_ )), key=la
d
```

```
Top 20 Important Features
```

Out[102]:

```
[('not', 0.071259929794744589),
 ('great', 0.031464090901213305),
 ('awful', 0.019613522489146842),
 ('disappointed', 0.018209254683905097),
 ('best', 0.018123288885595841),
 ('horrible', 0.01645972779088253),
 ('worst', 0.015308000953517524),
 ('delicious', 0.014940451774096536),
 ('good', 0.014341912645783468),
 ('terrible', 0.014144244723051059),
 ('not disappointed', 0.011463617931585882),
 ('love', 0.010046078489882072),
 ('disappointing', 0.0099461167020618163),
 ('not worth', 0.0086120262892855886),
 ('not buy', 0.0083185710380251753),
 ('not good', 0.0075581878131418239),
 ('bad', 0.0071914693550694641),
 ('waste money', 0.0071377255273794909),
 ('yuck', 0.0070940329107006964),
 ('poor', 0.0063804950408451145)]
```

In [104]:

```
dt_clf = tree.DecisionTreeClassifier(max_depth=bow_best_depth)
dt_clf.fit(tfidf_dict['train_tf_idf'],Y_train)
tfidf_test_proba = dt_clf.predict_proba(tfidf_dict['test_tf_idf'])
tfidf_train_proba = dt_clf.predict_proba(tfidf_dict['train_tf_idf'])
tfidf_test_proba
```

Out[104]:

```
array([[ 0.01912164,  0.98087836],
       [ 0.10272429,  0.89727571],
       [ 0.05752212,  0.94247788],
       ...,
       [ 0.10272429,  0.89727571],
       [ 0.01912164,  0.98087836],
       [ 1.        ,  0.        ]])
```

In [105]:

```
tfidf_fpr_train, tfidf_tpr_train, _ = roc_curve(Y_train, tfidf_train_proba[:, 1])
tfidf_fpr_test, tfidf_tpr_test, _ = roc_curve(Y_test, tfidf_test_proba[:, 1])
tfidf_test_auc = auc(tfidf_fpr_test, tfidf_tpr_test)
tfidf_train_auc = auc(tfidf_fpr_train, tfidf_tpr_train)
print(tfidf_test_auc)
print(tfidf_train_auc)
```

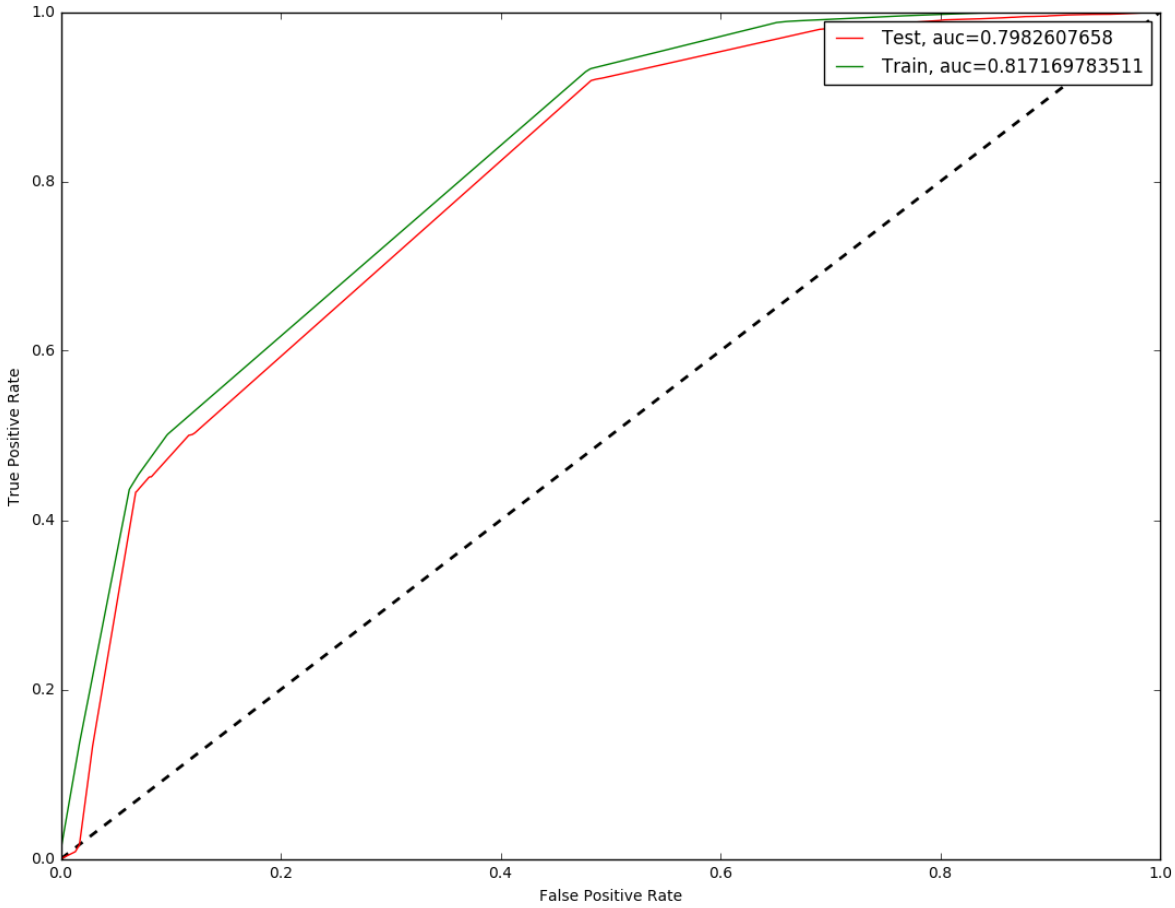
```
0.7982607658
0.817169783511
```

In [106]:

```
import pylab
plt.figure(figsize=(13, 10))
plt.plot([0,1], [0,1], color='black', lw=2, linestyle='--')
plt.plot(tfidf_fpr_test, tfidf_tpr_test, label="Test, auc="+str(tfidf_test_auc), color = 'r
plt.plot(tfidf_fpr_train, tfidf_tpr_train, label="Train, auc="+str(tfidf_train_auc), color

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()

plt.show()
```



In [111]:

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

In [112]:

```
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)
```

```
[[  806  1870]
 [  336 16988]]
             precision    recall  f1-score   support

          0       0.71      0.30      0.42      2676
          1       0.90      0.98      0.94     17324

avg / total       0.87      0.89      0.87     20000
```

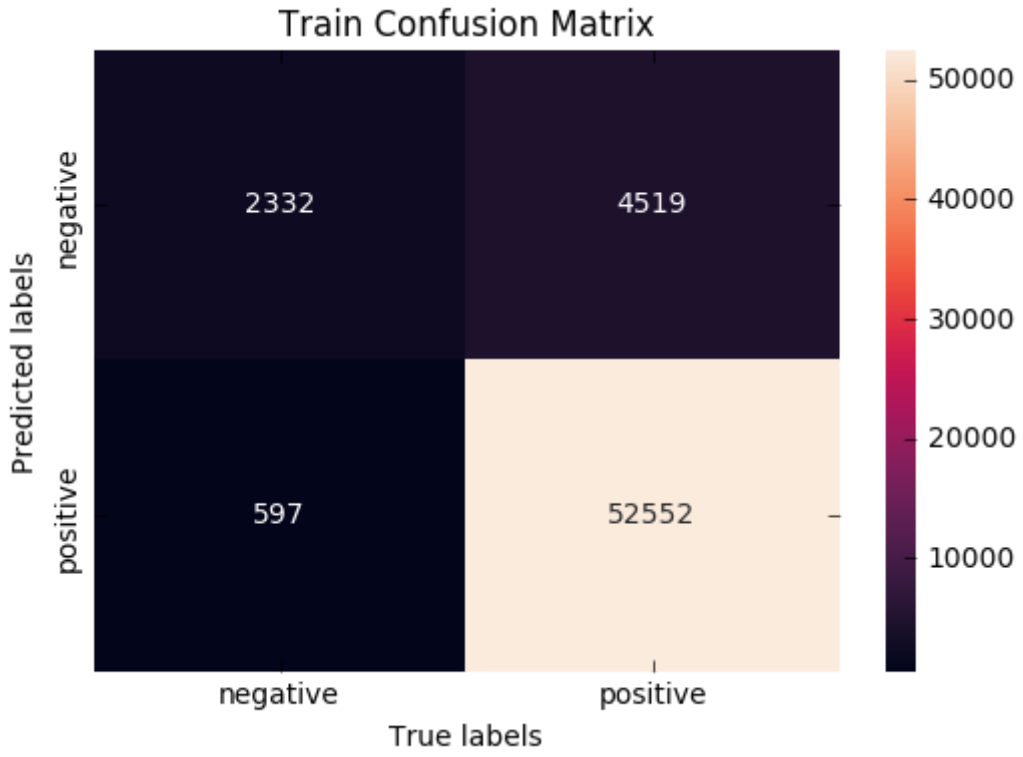
In [113]:

```
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[113]:

```
[<matplotlib.text.Text at 0x216916a0>, <matplotlib.text.Text at 0x21691278>]
```



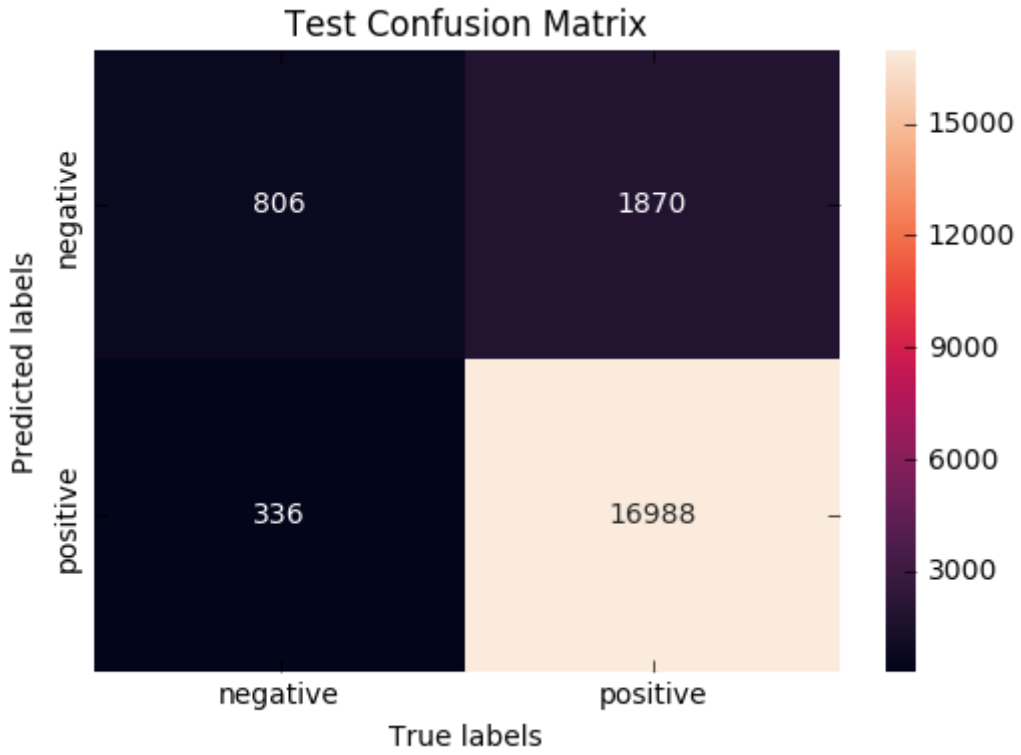
In [114]:

```
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[114]:

```
[<matplotlib.text.Text at 0x13b0c6a0>, <matplotlib.text.Text at 0x276cf208>]
```



Exporting Decision Tree

In [115]:

```
tclassifier = tree.DecisionTreeClassifier(max_depth = 3)
tclassifier.fit(tfidf_dict['train_tf_idf'],Y_train)
```

Out[115]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3,
            max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best')
```

In [ ]:

```
from graphviz import Source
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'
graph = Source(tree.export_graphviz(tclassifier, out_file='tftree.dot', feature_names=count
png_bytes = graph.pipe(format='png')
with open('tfidf.png','wb') as f:
    f.write(png_bytes)
```

Decision Tree on Avg_w2v

In [118]:

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

In [128]:

```
avg_lgr_train_score_list = []
avg_lgr_val_score_list = []
fpr = dict()
tpr = dict()
fpr_val = dict()
tpr_val = dict()
roc_auc_train = dict()
roc_auc_val = dict()
roc_auc_test = dict()
for depth_val in tqdm([1, 5, 10, 50, 100, 500, 1000]):
    tfidfclf = tree.DecisionTreeClassifier(max_depth=depth_val)
    tfidfclf.fit(avg_tfidf_dict['X_train_avgw2v'],Y_train)
    train_proba = tfidfclf.predict_proba(avg_tfidf_dict['X_train_avgw2v'])
    val_proba = tfidfclf.predict_proba(avg_tfidf_dict['X_val_avgw2v'])

    fpr[depth_val], tpr[depth_val], _ = roc_curve(Y_train, train_proba[:, 1])
    roc_auc_train[depth_val] = auc(fpr[depth_val], tpr[depth_val])
    avg_lgr_train_score_list.append(auc(fpr[depth_val], tpr[depth_val]))
    fpr_val[depth_val], tpr_val[depth_val], _ = roc_curve(Y_val, val_proba[:, 1])
    roc_auc_val[depth_val] = auc(fpr_val[depth_val], tpr_val[depth_val])
    avg_lgr_val_score_list.append(auc(fpr_val[depth_val], tpr_val[depth_val]))
print(roc_auc_train)
print(roc_auc_val)
```

```
  0%|
| 0/7 [00:00<?, ?it/s]

 14%|
| 1/7 [00:00<00:03,  1.75it/s]

 29%|
| 2/7 [00:02<00:06,  1.30s/it]

 43%|
| 3/7 [00:06<00:08,  2.08s/it]

 57%|
                                                                | 4/7 [0
0:11<00:08,  2.94s/it]

 71%|
                                                                | 5/7 [0
0:17<00:06,  3.44s/it]

 86%|
                                                                | 6/7 [0
0:22<00:03,  3.78s/it]

100%|
                                                               | 7/7 [0
0:28<00:00,  4.08s/it]


{1: 0.66766684893343109, 50: 0.99999999176104382, 100: 0.99999999176104382,
5: 0.82967684158430965, 1000: 0.99999999176104382, 500: 0.99999999176104382,
10: 0.91595788826755598}
{1: 0.67157686142209716, 50: 0.66628367147574319, 100: 0.66586440420098103,
```

```
5: 0.820100580218179, 1000: 0.6641456459645253, 500: 0.66444379510198681, 1
0: 0.80462356187259332}
```

In [129]:

```
avg_best_depth = max(roc_auc_val, key=roc_auc_val.get)
avg_best_depth
```

Out[129]:

```
5
```

In [130]:

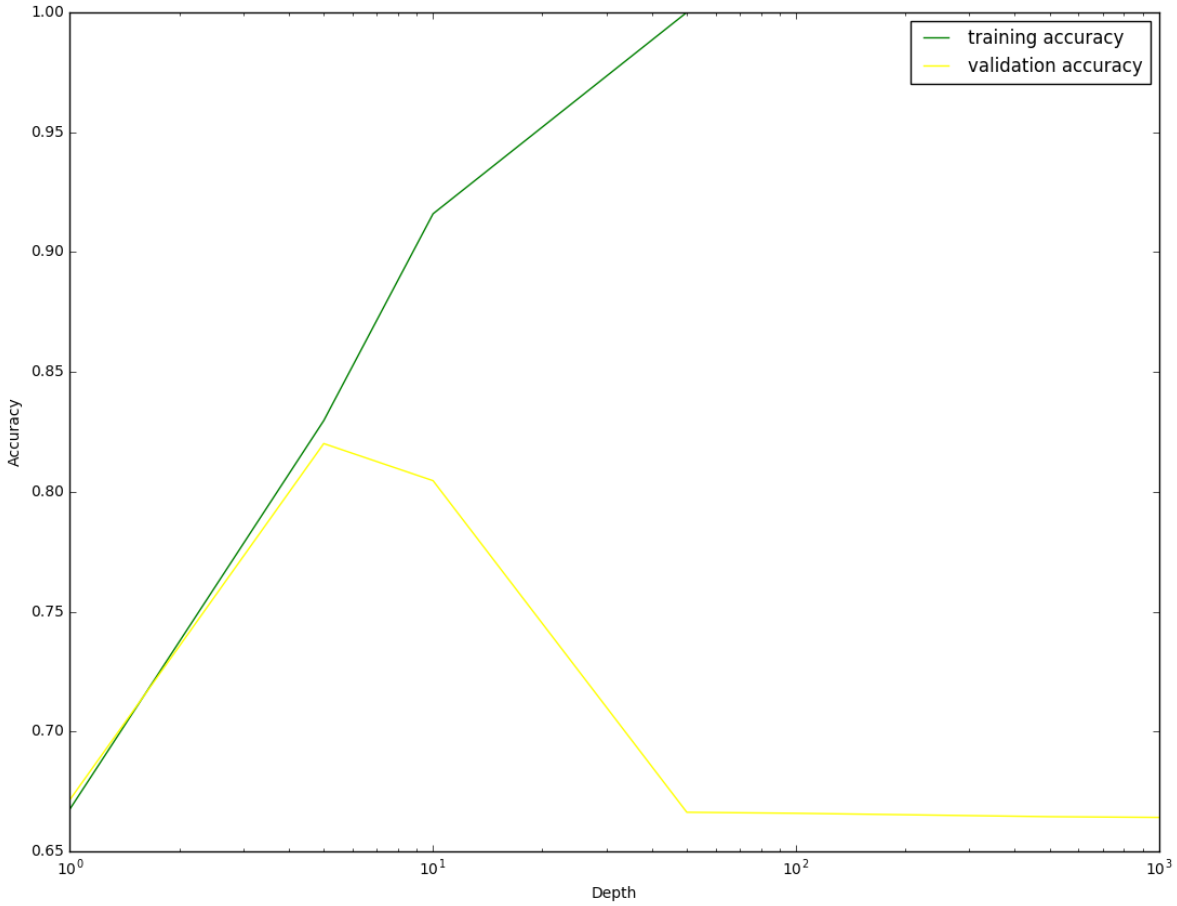
```
print(avg_lgr_train_score_list)
print(avg_lgr_val_score_list)
```

```
[0.66766684893343109, 0.82967684158430965, 0.91595788826755598, 0.9999999917
6104382, 0.99999999176104382, 0.99999999176104382, 0.99999999176104382]
[0.67157686142209716, 0.820100580218179, 0.80462356187259332, 0.666283671475
74319, 0.66586440420098103, 0.66444379510198681, 0.6641456459645253]
```

In [131]:

```
import pylab
plt.figure(figsize=(13, 10))
neighbors_settings = [1, 5, 10, 50, 100, 500, 1000]
plt.plot(neighbors_settings, avg_lgr_train_score_list, label="training accuracy", color='gr
plt.plot(neighbors_settings, avg_lgr_val_score_list, label="validation accuracy", color='ye
plt.xlabel('Depth')
plt.ylabel('Accuracy')
plt.legend()
plt.xscale('log')

plt.show()
```



In [132]:

```
dt_clf = tree.DecisionTreeClassifier(max_depth=bow_best_depth)
dt_clf.fit(avg_tfidf_dict['X_train_avgw2v'],Y_train)
avg_test_proba = dt_clf.predict_proba(avg_tfidf_dict['X_test_avgw2v'])
avg_train_proba = dt_clf.predict_proba(avg_tfidf_dict['X_train_avgw2v'])
avg_test_proba
```

Out[132]:

```
array([[ 0.01212121,  0.98787879],
       [ 0.97647059,  0.02352941],
       [ 0.02313702,  0.97686298],
       ...,
       [ 0.04298643,  0.95701357],
       [ 0.00585294,  0.99414706],
       [ 0.        ,  1.        ]])
```

In [133]:

```
avg_fpr_train, avg_tpr_train, _ = roc_curve(Y_train, avg_train_proba[:, 1])
avg_fpr_test, avg_tpr_test, _ = roc_curve(Y_test, avg_test_proba[:, 1])
avg_test_auc = auc(avg_fpr_test, avg_tpr_test)
avg_train_auc = auc(avg_fpr_train, avg_tpr_train)
print(avg_test_auc)
print(avg_train_auc)
```

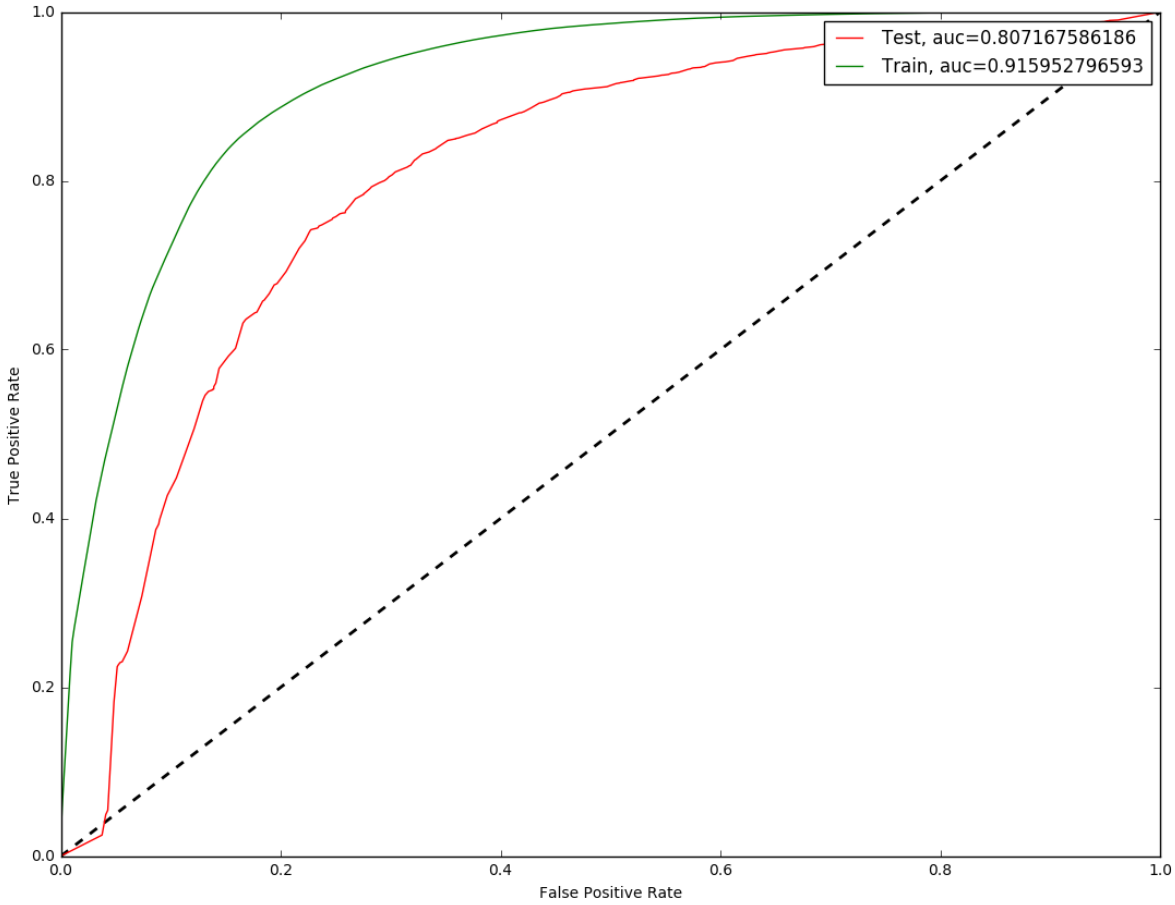
```
0.807167586186
0.915952796593
```

In [134]:

```
import pylab
plt.figure(figsize=(13, 10))
plt.plot([0,1], [0,1], color='black', lw=2, linestyle='--')
plt.plot(avg_fpr_test, avg_tpr_test, label="Test, auc="+str(avg_test_auc), color = 'red')
plt.plot(avg_fpr_train, avg_tpr_train, label="Train, auc="+str(avg_train_auc), color = 'gre

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()

plt.show()
```



In [135]:

```
avg_test_conf = dt_clf.predict(avg_tfidf_dict['X_test_avgw2v'])
avg_train_conf = dt_clf.predict(avg_tfidf_dict['X_train_avgw2v'])
```

In [136]:

```
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)
```

```
[[  943  1733]
 [  823 16501]]
             precision    recall  f1-score   support

          0       0.53      0.35      0.42      2676
          1       0.90      0.95      0.93     17324

avg / total       0.86      0.87      0.86     20000
```

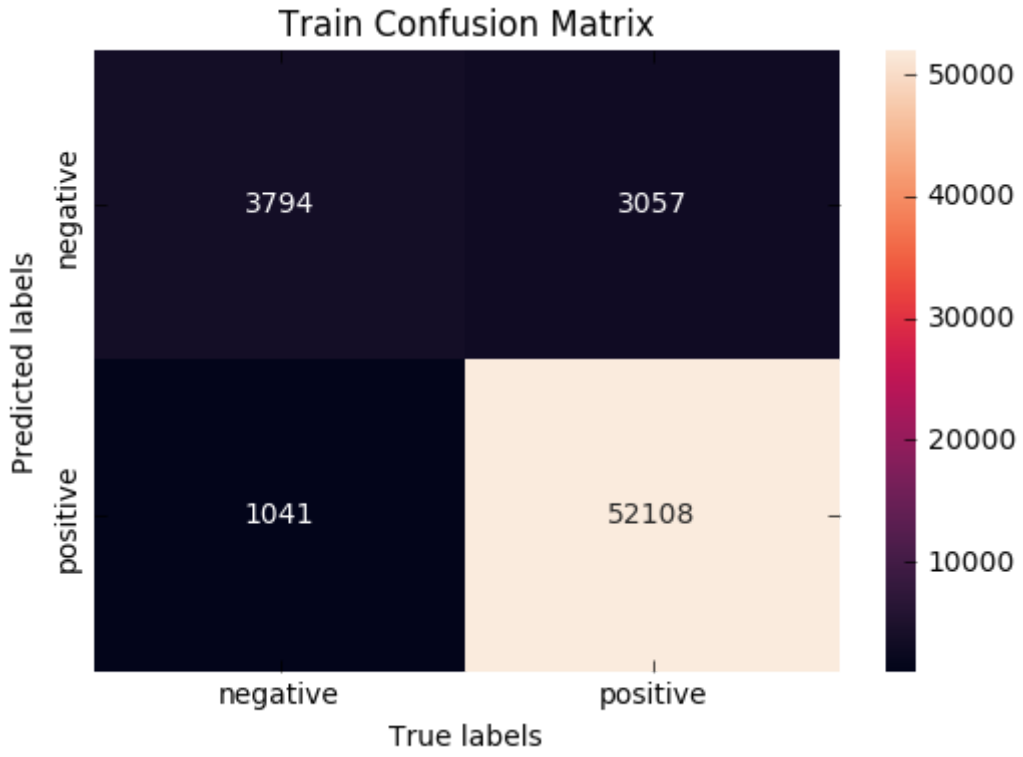
In [137]:

```
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[137]:

```
[<matplotlib.text.Text at 0x2859f2b0>, <matplotlib.text.Text at 0x13fcc400>]
```



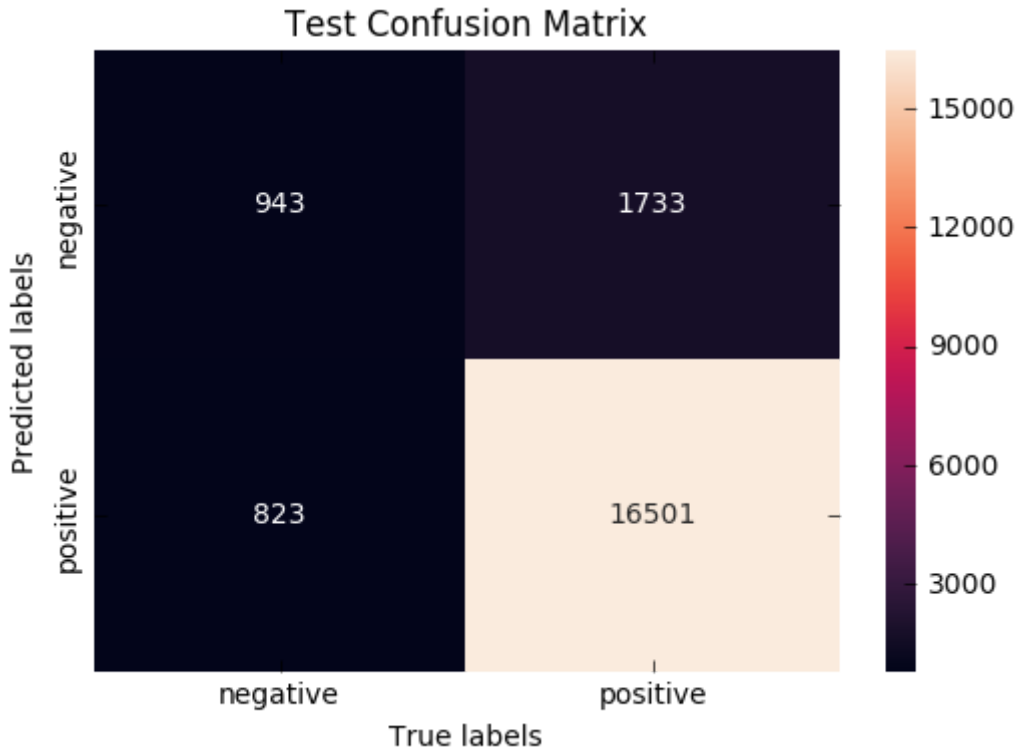
In [138]:

```
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[138]:

```
[<matplotlib.text.Text at 0x212ad8d0>, <matplotlib.text.Text at 0x245f6588>]
```



In [139]:

```
tclassifier = tree.DecisionTreeClassifier(max_depth = 3)
tclassifier.fit(avg_tfidf_dict['X_train_avgw2v'],Y_train)
```

Out[139]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3,
            max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best')
```

In [ ]:

```
from graphviz import Source
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'
graph = Source(tree.export_graphviz(tclassifier, out_file='avg.dot', feature_names=count_ve
png_bytes = graph.pipe(format='png')
with open('tfidf.png','wb') as f:
    f.write(png_bytes)
```

Decision Tree on TFIDF-w2v

In [142]:

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

In [146]:

```
tfidfw2v_lgr_train_score_list = []
tfidfw2v_lgr_val_score_list = []
fpr = dict()
tpr = dict()
fpr_val = dict()
tpr_val = dict()
roc_auc_train = dict()
roc_auc_val = dict()
roc_auc_test = dict()
for depth_val in tqdm([1, 5, 10, 50, 100, 500, 1000]):
    tfidfw2v_clf = tree.DecisionTreeClassifier(max_depth=depth_val)
    tfidfw2v_clf.fit(tfidfw2v_dict['X_train_tfidfw2v'],Y_train)
    train_proba = tfidfw2v_clf.predict_proba(tfidfw2v_dict['X_train_tfidfw2v'])
    val_proba = tfidfw2v_clf.predict_proba(tfidfw2v_dict['X_val_tfidfw2v'])

    fpr[depth_val], tpr[depth_val], _ = roc_curve(Y_train, train_proba[:, 1])
    roc_auc_train[depth_val] = auc(fpr[depth_val], tpr[depth_val])
    tfidfw2v_lgr_train_score_list.append(auc(fpr[depth_val], tpr[depth_val]))
    fpr_val[depth_val], tpr_val[depth_val], _ = roc_curve(Y_val, val_proba[:, 1])
    roc_auc_val[depth_val] = auc(fpr_val[depth_val], tpr_val[depth_val])
    tfidfw2v_lgr_val_score_list.append(auc(fpr_val[depth_val], tpr_val[depth_val]))
print(roc_auc_train)
print(roc_auc_val)
```

```
  0%|
| 0/7 [00:00<?, ?it/s]

 14%|
| 1/7 [00:00<00:03,  1.88it/s]

 29%|
| 2/7 [00:02<00:05,  1.20s/it]

 43%|
| 3/7 [00:05<00:07,  1.97s/it]

 57%|
| 4/7 [0
0:11<00:08,  2.87s/it]

 71%|
| 5/7 [0
0:17<00:06,  3.40s/it]

 86%|
| 6/7 [0
0:22<00:03,  3.77s/it]

100%|
| 7/7 [0
0:27<00:00,  4.00s/it]


{1: 0.65121505282328429, 50: 0.99999999176104382, 100: 0.99999999176104382,
5: 0.80301625656717923, 1000: 0.99999999176104382, 500: 0.99999999176104382,
10: 0.89065852572849813}
{1: 0.64939469230966784, 50: 0.65038567520689972, 100: 0.65115593165042696,
```

```
5: 0.79274812386210203, 1000: 0.65198141458550429, 500: 0.6485145953374396,
10: 0.77410210805959578}
```

In [147]:

```
tfidfw2v_best_depth = max(roc_auc_val, key=roc_auc_val.get)
tfidfw2v_best_depth
```

Out[147]:

```
5
```

In [149]:

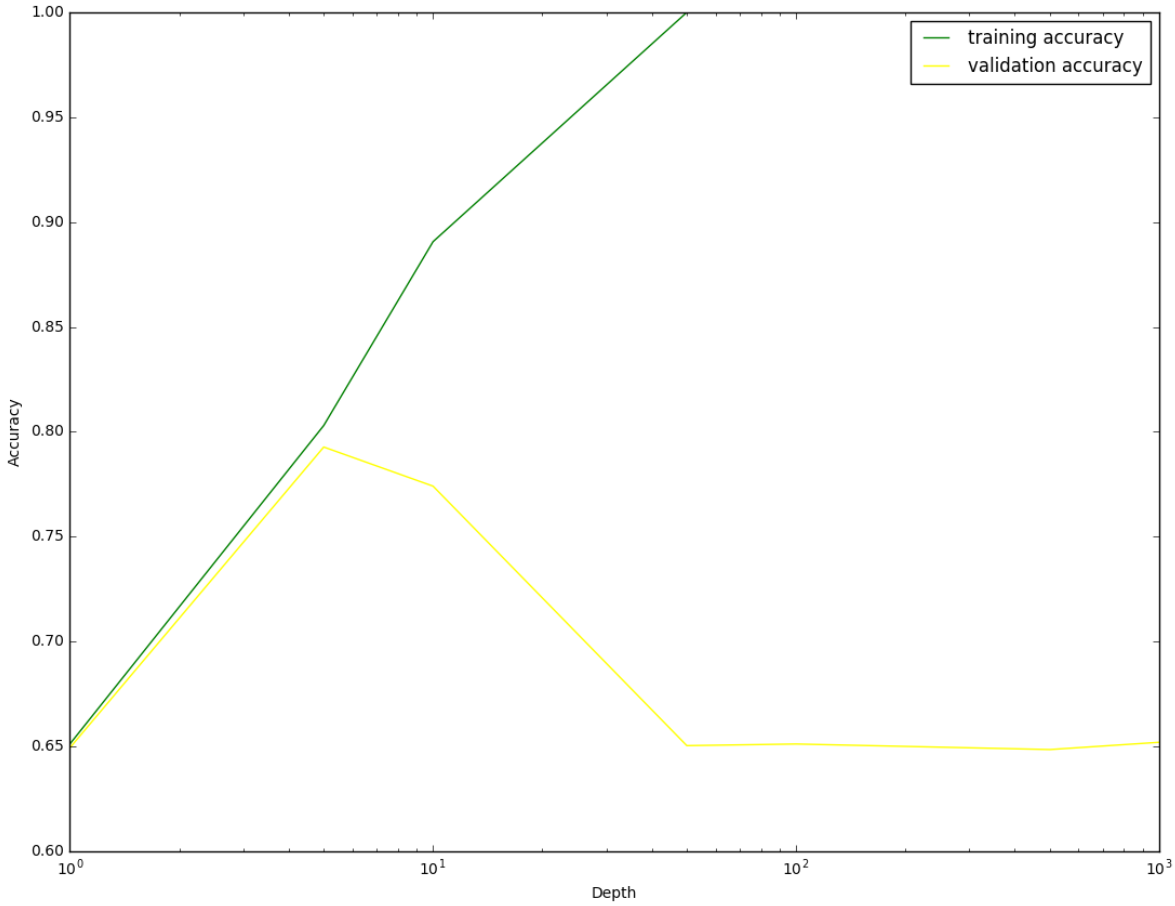
```
print(tfidfw2v_lgr_train_score_list)
print(tfidfw2v_lgr_val_score_list)
```

```
[0.65121505282328429, 0.80301625656717923, 0.89065852572849813, 0.9999999917
6104382, 0.99999999176104382, 0.99999999176104382, 0.99999999176104382]
[0.64939469230966784, 0.79274812386210203, 0.77410210805959578, 0.6503856752
0689972, 0.65115593165042696, 0.6485145953374396, 0.65198141458550429]
```

In [150]:

```
import pylab
plt.figure(figsize=(13, 10))
neighbors_settings = [1, 5, 10, 50, 100, 500, 1000]
plt.plot(neighbors_settings, tfidfw2v_lgr_train_score_list, label="training accuracy", colo
plt.plot(neighbors_settings, tfidfw2v_lgr_val_score_list, label="validation accuracy", colo
plt.xlabel('Depth')
plt.ylabel('Accuracy')
plt.legend()
plt.xscale('log')

plt.show()
```



In [151]:

```
dt_clf = tree.DecisionTreeClassifier(max_depth=tfidfw2v_best_depth)
dt_clf.fit(tfidfw2v_dict['X_train_tfidfw2v'],Y_train)
tfidfw2v_test_proba = dt_clf.predict_proba(tfidfw2v_dict['X_test_tfidfw2v'])
tfidfw2v_train_proba = dt_clf.predict_proba(tfidfw2v_dict['X_train_tfidfw2v'])
tfidfw2v_test_proba
```

Out[151]:

```
array([[ 0.20675944,  0.79324056],
       [ 0.18998979,  0.81001021],
       [ 0.07217125,  0.92782875],
       ...,
       [ 0.04957678,  0.95042322],
       [ 0.04150653,  0.95849347],
       [ 0.22584356,  0.77415644]])
```

In [153]:

```
tfidfw2v_fpr_train, tfidfw2v_tpr_train, _ = roc_curve(Y_train, tfidfw2v_train_proba[:, 1])
tfidfw2v_fpr_test, tfidfw2v_tpr_test, _ = roc_curve(Y_test, tfidfw2v_test_proba[:, 1])
tfidfw2v_test_auc = auc(tfidfw2v_fpr_test, tfidfw2v_tpr_test)
tfidfw2v_train_auc = auc(tfidfw2v_fpr_train, tfidfw2v_tpr_train)
print(tfidfw2v_test_auc)
print(tfidfw2v_train_auc)
```

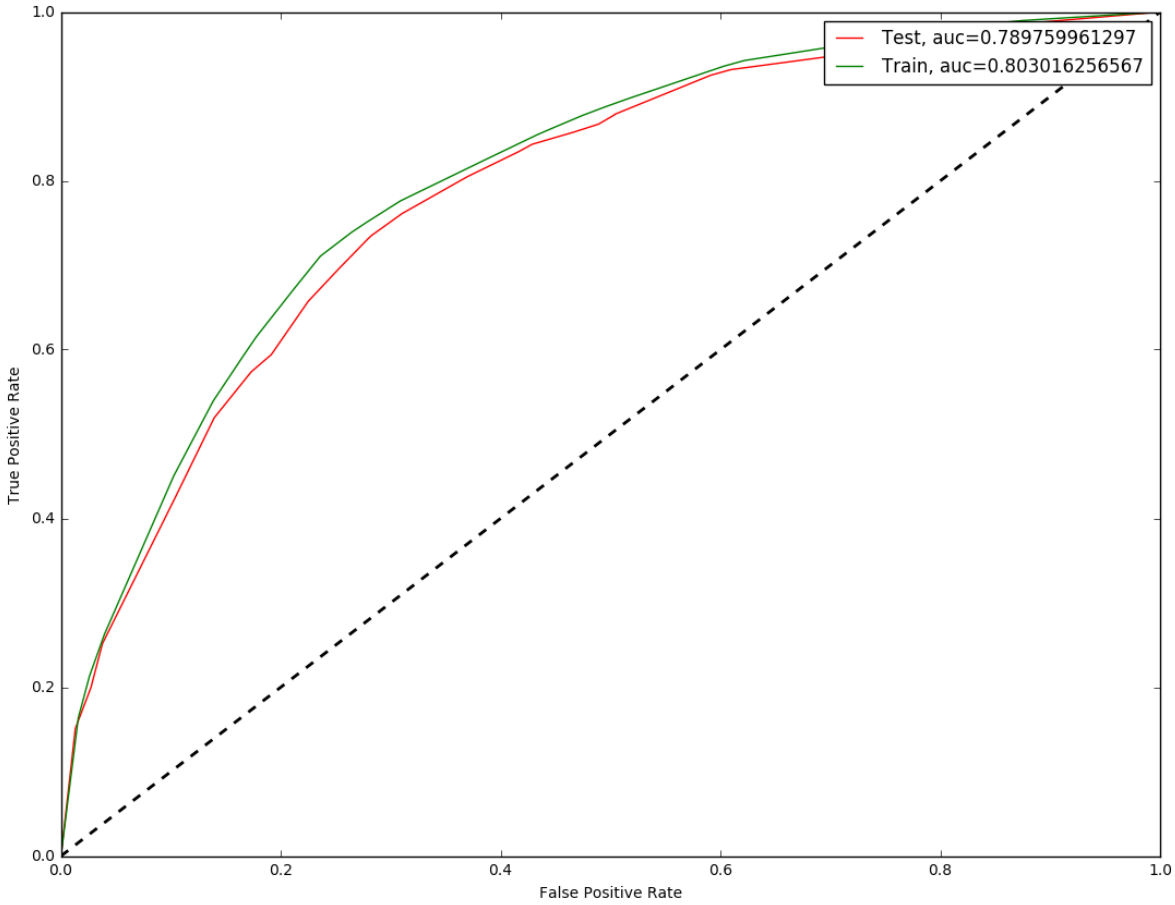
```
0.789759961297
0.803016256567
```

In [154]:

```
import pylab
plt.figure(figsize=(13, 10))
plt.plot([0,1], [0,1], color='black', lw=2, linestyle='--')
plt.plot(tfidfw2v_fpr_test, tfidfw2v_tpr_test, label="Test, auc="+str(tfidfw2v_test_auc), c
plt.plot(tfidfw2v_fpr_train, tfidfw2v_tpr_train, label="Train, auc="+str(tfidfw2v_train_auc

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()

plt.show()
```



In [155]:

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

In [156]:

```
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)
```

```
[[  381  2295]
 [  302 17022]]
             precision    recall  f1-score   support

          0       0.56      0.14      0.23      2676
          1       0.88      0.98      0.93     17324

avg / total       0.84      0.87      0.84     20000
```

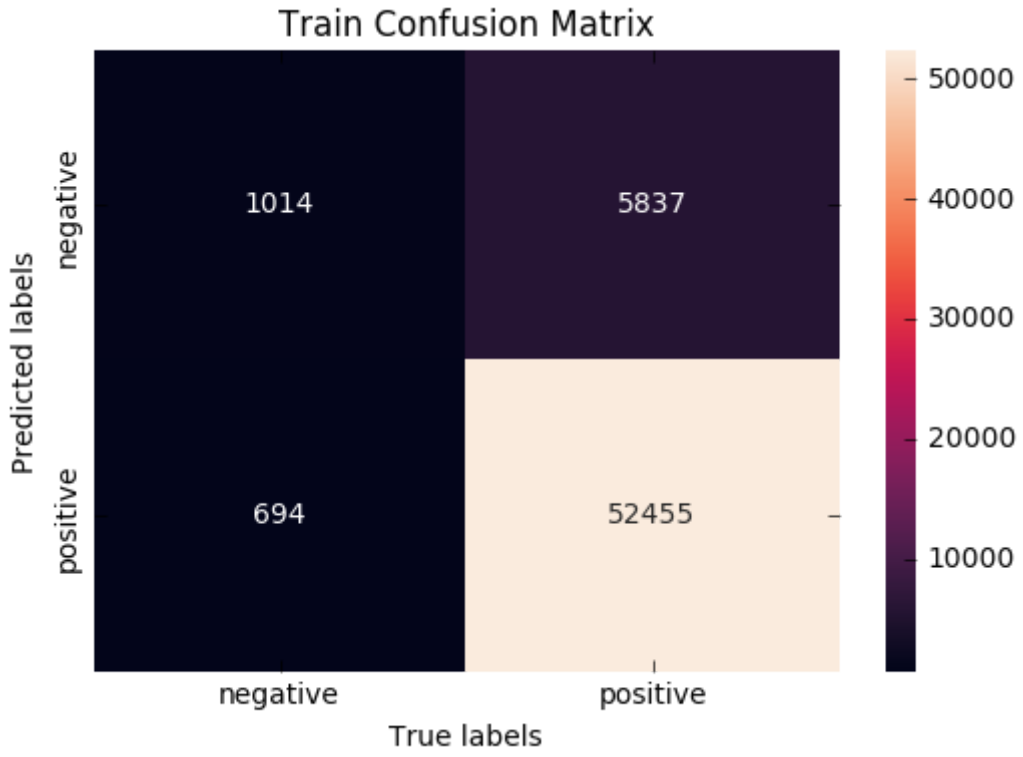
In [157]:

```
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[157]:

```
[<matplotlib.text.Text at 0x329cf240>, <matplotlib.text.Text at 0x32a95390>]
```



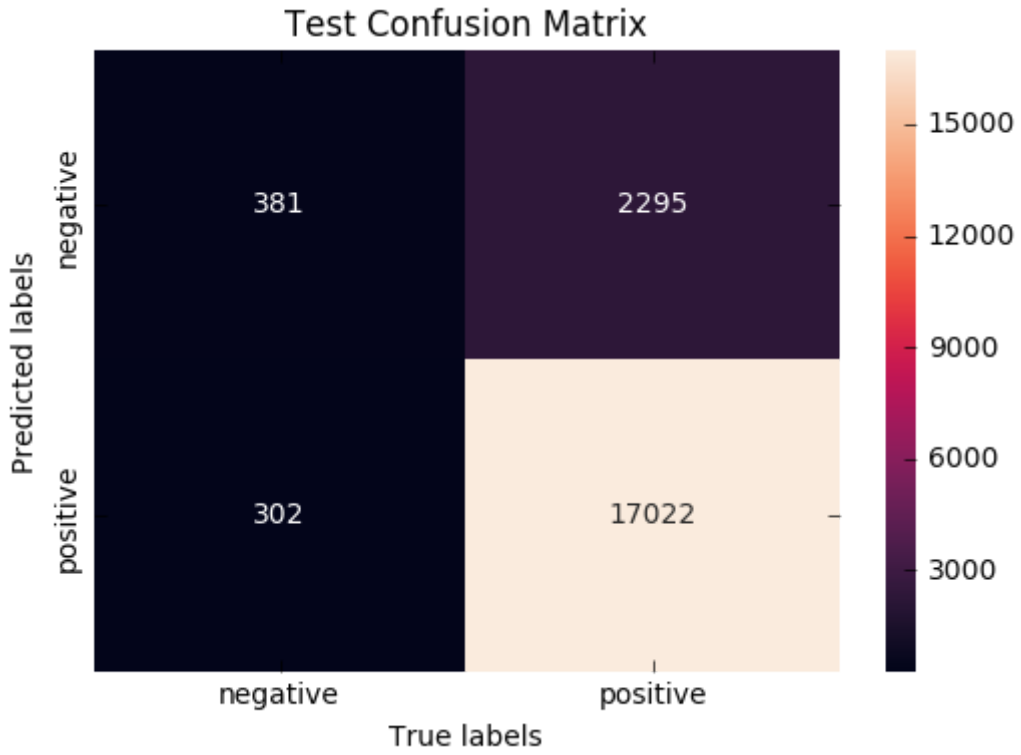
In [158]:

```
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[158]:

```
[<matplotlib.text.Text at 0x100b9c50>, <matplotlib.text.Text at 0x27afc2b0>]
```



In [164]:

```
tclassifier = tree.DecisionTreeClassifier(max_depth = 3)
tclassifier.fit(tfidfw2v_dict['X_train_tfidfw2v'],Y_train)
```

Out[164]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3,
            max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best')
```

In [ ]:

```
from graphviz import Source
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'
graph = Source(tree.export_graphviz(tclassifier, out_file='avg.dot', feature_names=count_ve
png_bytes = graph.pipe(format='png')
with open('tfidf.png','wb') as f:
    f.write(png_bytes)
```

In [163]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Hyperparameter", "Vectorizer", "Train", "Test"]

x.add_row([10, "BoW", 0.8122, 0.7929])
x.add_row([10, "Tf-idf", 0.8171, 0.7982])
x.add_row([5, "Avg-w2v", 0.9159, 0.8071])
x.add_row([5, "tfidf_w2v", 0.8030, 0.7897])
print(x)
```

```
+----------------+------------+--------+--------+
| Hyperparameter | Vectorizer | Train  |  Test  |
+----------------+------------+--------+--------+
|       10       |    BoW     | 0.8122 | 0.7929 |
|       10       |   Tf-idf   | 0.8171 | 0.7982 |
|       5        |  Avg-w2v   | 0.9159 | 0.8071 |
|       5        | tfidf_w2v  | 0.803  | 0.7897 |
+----------------+------------+--------+--------+
```

Steps taken to increase performance: i. Summary and Text columns are appended in single column ii. length of words is taken from appended column and stacked with sparse matrix

Observations: i. Accuracy increased around 2% for each vectorizer.