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
from sklearn import linear_model
from sklearn.calibration import CalibratedClassifierCV
```

In [2]:

```
#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)
```

```
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

---------------------------------------------------------------------------
OperationalError                          Traceback (most recent call last)
/usr/local/lib/python3.6/dist-packages/pandas/io/sql.py in execute(self, *ar
gs, **kwargs)
   1408             else:
-> 1409                 cur.execute(*args)
   1410             return cur

OperationalError: no such table: Reviews

During handling of the above exception, another exception occurred:
```

```
DatabaseError                             Traceback (most recent call last)
<ipython-input-2-3a4070174e29> in <module>()
     13 # for tsne assignment you can take 5k data points
     14
---> 15 filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Sc
ore != 3""", con)
     16
     17 # Give reviews with Score>3 a positive rating(1), and reviews with a
score<3 a negative rating(0).

/usr/local/lib/python3.6/dist-packages/pandas/io/sql.py in read_sql_query(sq
l, con, index_col, coerce_float, params, parse_dates, chunksize)
    330     return pandas_sql.read_query(
    331         sql, index_col=index_col, params=params, coerce_float=coerce
_float,
--> 332         parse_dates=parse_dates, chunksize=chunksize)
    333
    334

/usr/local/lib/python3.6/dist-packages/pandas/io/sql.py in read_query(self,
 sql, index_col, coerce_float, params, parse_dates, chunksize)
   1442
   1443         args = _convert_params(sql, params)
-> 1444         cursor = self.execute(*args)
   1445         columns = [col_desc[0] for col_desc in cursor.description]
   1446

/usr/local/lib/python3.6/dist-packages/pandas/io/sql.py in execute(self, *ar
gs, **kwargs)
   1419             ex = DatabaseError(
   1420                 "Execution failed on sql '%s': %s" % (args[0], exc))
-> 1421             raise_with_traceback(ex)
   1422
   1423     @staticmethod

/usr/local/lib/python3.6/dist-packages/pandas/compat/__init__.py in raise_wi
th_traceback(exc, traceback)
    383         if traceback == Ellipsis:
    384             _, _, traceback = sys.exc_info()
--> 385         raise exc.with_traceback(traceback)
    386 else:
    387     # this version of raise is a syntax error in Python 3

/usr/local/lib/python3.6/dist-packages/pandas/io/sql.py in execute(self, *ar
gs, **kwargs)
   1407                 cur.execute(*args, **kwargs)
   1408             else:
-> 1409                 cur.execute(*args)
   1410             return cur
   1411         except Exception as exc:

DatabaseError: Execution failed on sql ' SELECT * FROM Reviews WHERE Score !
= 3': no such table: Reviews
```

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 [0]:

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

(80668, 7)

Out[4]:

	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 [0]:

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

(100000, 13)

In [0]:

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

Out[28]:

```
100.0
```

In [0]:

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

In [0]:

```
#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()
```

```
(100000, 13)
```

Out[30]:

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

In [0]:

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

Out[31]:

```
117924    EVERY book is educational. this witty little b...
117901    This whole series is great way to spend time w...
298792    Entertainingl Funny!. Beetlejuice is a well wr...
169281    A modern day fairy tale. A twist of rumplestis...
298791    FANTASTIC!. Beetlejuice is an excellent and fu...
Name: cleanReview, dtype: object
```

In [0]:

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

Out[32]:

```
117924    78
117901    90
298792    31
169281    41
298791    44
Name: lengthOfReview, dtype: int64
```

In [0]:

```
#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 [00:01<00:00, 313040.89it/s]

In [0]:

```
#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 [00:01<00:00, 343430.96it/s]

In [0]:

```
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 [02:16<00:00, 2662.25it/s]

In [0]:

```
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 [0]:

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

100%|██| 364171/364171 [00:06<00:00, 53616.87it/s]

In [0]:

```
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 [00:30<00:00, 11914.41it/s]

In [0]:

```
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 [00:20<00:00, 18059.44it/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 [0]:

```
# 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 [03:37<00:00, 1671.64it/s]

In [0]:

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

364171

Out[21]:

'satisfied product advertised use cereal raw vinegar general sweetner'

In [0]:

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

In [0]:

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

364171

Out[23]:

	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 [322]:

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

In [323]:

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

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

In [324]:

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

In [325]:

```
filtered_data.shape
```

Out[325]:

```
(364171, 12)
```

In [326]:

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

In [327]:

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

Out[327]:

	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 [328]:

```
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 [329]:

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

Out[329]:

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

In [330]:

```
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 [331]:

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

Out[331]:

100000

In [332]:

```
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 [333]:

```
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 [334]:

```
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 [335]:

```
X_train_vect.shape
```

Out[335]:

(60000, 47535)

In [337]:

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

Out[337]:

100000

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 [0]:

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

```
(60000, 48271)
(20000, 48271)
(20000, 48271)
```

In [0]:

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

```
48270
```

In [0]:

```
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, 48271)
```

In [0]:

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

[4.3] TF-IDF

In [149]:

```
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]:

```
from scipy.sparse import hstack
tfidf_concat_data_train = hstack((train_tf_idf,np.array(filtered_data['lengthOfReview'][0:6
tfidf_concat_data_val = hstack((cv_tf_idf,np.array(filtered_data['lengthOfReview'][60000:80
tfidf_concat_data_test = hstack((test_tf_idf,np.array(filtered_data['lengthOfReview'][80000
```

In [150]:

```
# tf_idf_dict = {'train_tf_idf': tfidf_concat_data_train, 'cv_tf_idf': tfidf_concat_data_va
tf_idf_dict = {'train_tf_idf': train_tf_idf, 'cv_tf_idf': cv_tf_idf, 'test_tf_idf': test_tf
```

In [151]:

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

In [228]:

```
print(type(train_tf_idf))
```

```
<class 'scipy.sparse.csr.csr_matrix'>
```

[4.4] Word2Vec

In [166]:

```
# 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 [167]:

```
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', 0.8299081325531006), ('terrific', 0.8178765773773193), ('fant
astic', 0.8172010183334351), ('wonderful', 0.7758762240409851), ('awesome',
0.7723352909088135), ('good', 0.7556071877479553), ('perfect', 0.72576248645
78247), ('fabulous', 0.6611760854721069), ('nice', 0.642975926399231), ('inc
redible', 0.6308016777038574)]
==================================================
[('greatest', 0.7711305618286133), ('best', 0.7473520636558533), ('tasties
t', 0.6962459087371826), ('experienced', 0.6544209122657776), ('terrible',
0.631056010723114), ('awful', 0.6309006810188293), ('closest', 0.60654187202
45361), ('disgusting', 0.5986604690551758), ('tasted', 0.5972884893417358),
('nicest', 0.5882304906845093)]
```

In [168]:

```
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  ['lawn', 'satisfy', 'juan', 'misshapen', 'window', 'thermos',
'bubble', 'pooping', 'resembles', 'amadei', 'unwrapped', 'sequence', 'wart
s', 'watches', 'suspected', 'dreaming', 'props', 'pair', 'increases', 'medio
cre', 'thousands', 'skyline', 'pillow', 'equipped', 'whiter', 'convience',
'wash', 'vegeta', 'bites', 'countless', 'testament', 'college', 'bedroom',
'chicory', 'grrrreat', 'finishes', 'flow', 'lesser', 'videos', 'casino', 'gr
eek', 'dashes', 'kilo', 'baking', 'fiasco', 'setting', 'evaluate', 'germinat
e', 'trick', 'clippers']
```

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

[4.4.1.1] Avg W2v

In [169]:

```
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 [221]:

```
# 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 [190]:

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

```
20000
```

In [0]:

```
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 [0]:

```
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 [0]:

```
# 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 [0]:

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

def 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 [0]:

```
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])
```

```
100%|██████████| 60000/60000 [1:28:17<00:00, 11.33it/s]
100%|██████████| 20000/20000 [35:03<00:00,  8.24it/s]
100%|██████████| 20000/20000 [39:32<00:00, 10.91it/s]
```

In [0]:

```
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 [0]:

```
with open('/content/gdrive/My Drive/Colab Notebooks/SVM/tfidf_w2v.pkl', 'wb') as handle:
    pickle.dump(tfidf_w2v_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

[5.1] Linear SVM

SVM on BoW

In [0]:

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

With L1 Regularizer

In [187]:

```
bow_lgr_train_score_list = []
bow_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 all_a in tqdm([0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 1000]):
    tfidfclf = linear_model.SGDClassifier(loss='hinge', penalty='l1', alpha=all_a)
    tfidfclf.fit(BoW_dict['X_train_vect'],Y_train)
    train_proba = tfidfclf.decision_function(BoW_dict['X_train_vect'])
    val_proba = tfidfclf.decision_function(BoW_dict['X_val_vect'])

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

```
100%|██████████| 9/9 [00:04<00:00,  2.24it/s]

{0.0001: 0.9302856073409252, 0.001: 0.889672547879794, 0.01: 0.6549415958389
471, 0.1: 0.4959504734267589, 1: 0.4959504734267589, 10: 0.5, 100: 0.5, 100
0: 0.5}
{0.0001: 0.9285264448517336, 0.001: 0.895371553275592, 0.01: 0.6670513302678
514, 0.1: 0.5107900526592518, 1: 0.5107900526592518, 10: 0.5, 100: 0.5, 100
0: 0.5}
```

With L2 Regularizer

In [188]:

```
bow_lgr_train_score_list = []
bow_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 all_a in tqdm([0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]):
    tfidfclf = linear_model.SGDClassifier(loss='hinge', penalty='l2', alpha=all_a)
    tfidfclf.fit(BoW_dict['X_train_vect'],Y_train)
    train_proba = tfidfclf.decision_function(BoW_dict['X_train_vect'])
    val_proba = tfidfclf.decision_function(BoW_dict['X_val_vect'])

    fpr[all_a], tpr[all_a], _ = roc_curve(Y_train, train_proba)
    roc_auc_train[all_a] = auc(fpr[all_a], tpr[all_a])
    bow_lgr_train_score_list.append(auc(fpr[all_a], tpr[all_a]))
    fpr_val[all_a], tpr_val[all_a], _ = roc_curve(Y_val, val_proba)
    roc_auc_val[all_a] = auc(fpr_val[all_a], tpr_val[all_a])
    bow_lgr_val_score_list.append(auc(fpr_val[all_a], tpr_val[all_a]))
print(roc_auc_train)
print(roc_auc_val)
```

```
100%|██████████| 9/9 [00:03<00:00,  2.58it/s]

{0.0001: 0.8898910683396446, 0.001: 0.9458219840225275, 0.01: 0.940732282374
1053, 0.1: 0.7466119414512644, 1: 0.5388590076750243, 10: 0.4986051872978508
6, 100: 0.49598150682812137, 1000: 0.495340202962125, 10000: 0.4952305905168
2583}
{0.0001: 0.8875006272377816, 0.001: 0.9404786614129061, 0.01: 0.938078093595
8728, 0.1: 0.7514898903630597, 1: 0.5515075628403557, 10: 0.512979049159898
3, 100: 0.5107243933237908, 1000: 0.5101391614662952, 10000: 0.5100366703903
894}
```

In [189]:

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

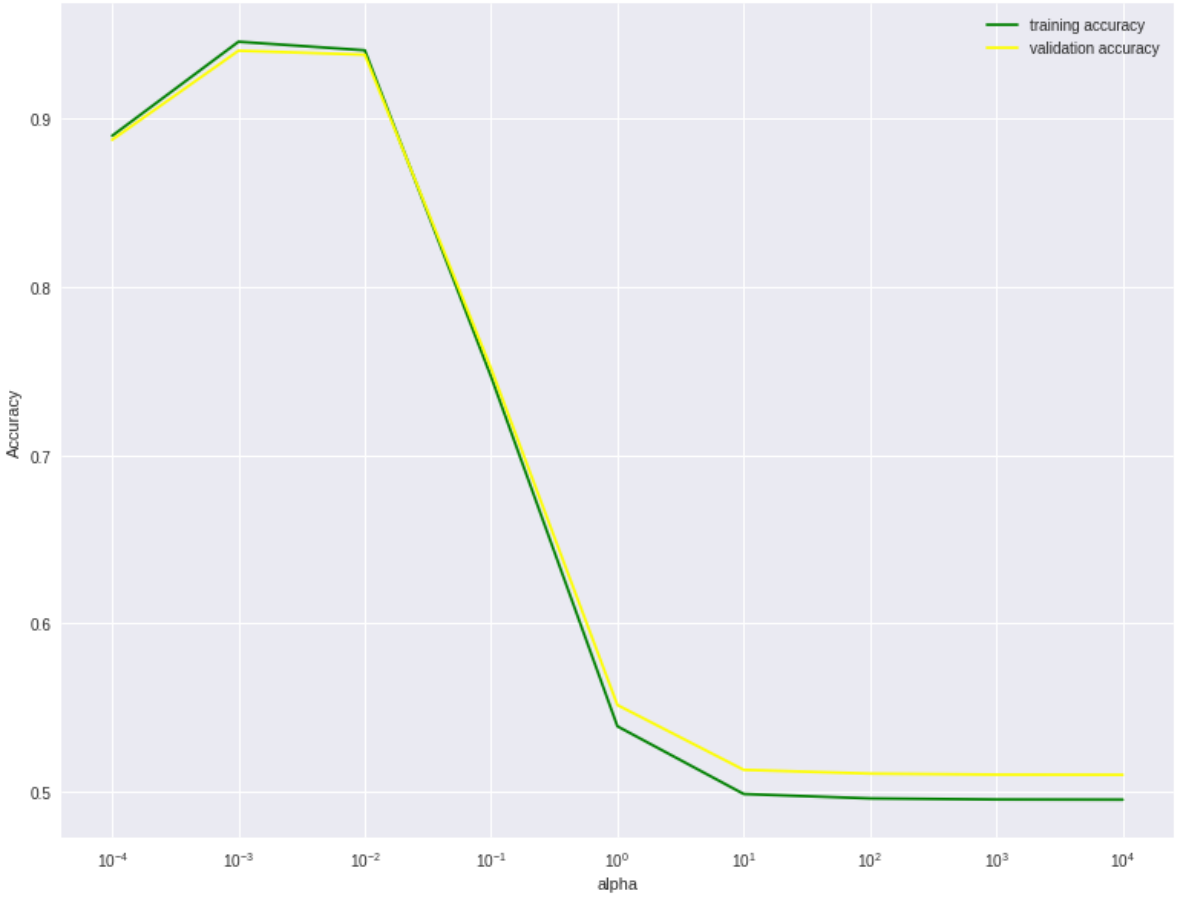
Out[189]:

```
0.001
```

In [190]:

```
import pylab
plt.figure(figsize=(13, 10))
neighbors_settings = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
plt.plot(neighbors_settings, bow_lgr_train_score_list, label="training accuracy", color='gr
plt.plot(neighbors_settings, bow_lgr_val_score_list, label="validation accuracy", color='ye
# plt.plot(neighbors_settings, auc_test, label="test accuracy", color='red')
plt.xlabel('alpha')
plt.ylabel('Accuracy')
plt.legend()
plt.xscale('log')

plt.show()
```



In [191]:

```
bow_svm_linear=linear_model.SGDClassifier(loss='hinge', penalty='l2', alpha=best_a)
bow_svm_linear.fit(BoW_dict['X_train_vect'],Y_train)
f = CalibratedClassifierCV(base_estimator=bow_svm_linear)
f.fit(BoW_dict['X_train_vect'],Y_train)
bow_linear_test_proba = f.predict_proba(BoW_dict['X_test_vect'])
bow_linear_train_proba = f.predict_proba(BoW_dict['X_train_vect'])
bow_linear_test_proba
```

Out[191]:

```
array([[0.06722363, 0.93277637],
       [0.08986817, 0.91013183],
       [0.01579419, 0.98420581],
       ...,
       [0.18866359, 0.81133641],
       [0.04041968, 0.95958032],
       [0.241349  , 0.758651  ]])
```

In [192]:

```
bow_fpr_train, bow_tpr_train, _ = roc_curve(Y_train, bow_linear_train_proba[:, 1])
bow_fpr_test, bow_tpr_test, _ = roc_curve(Y_test, bow_linear_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)
```

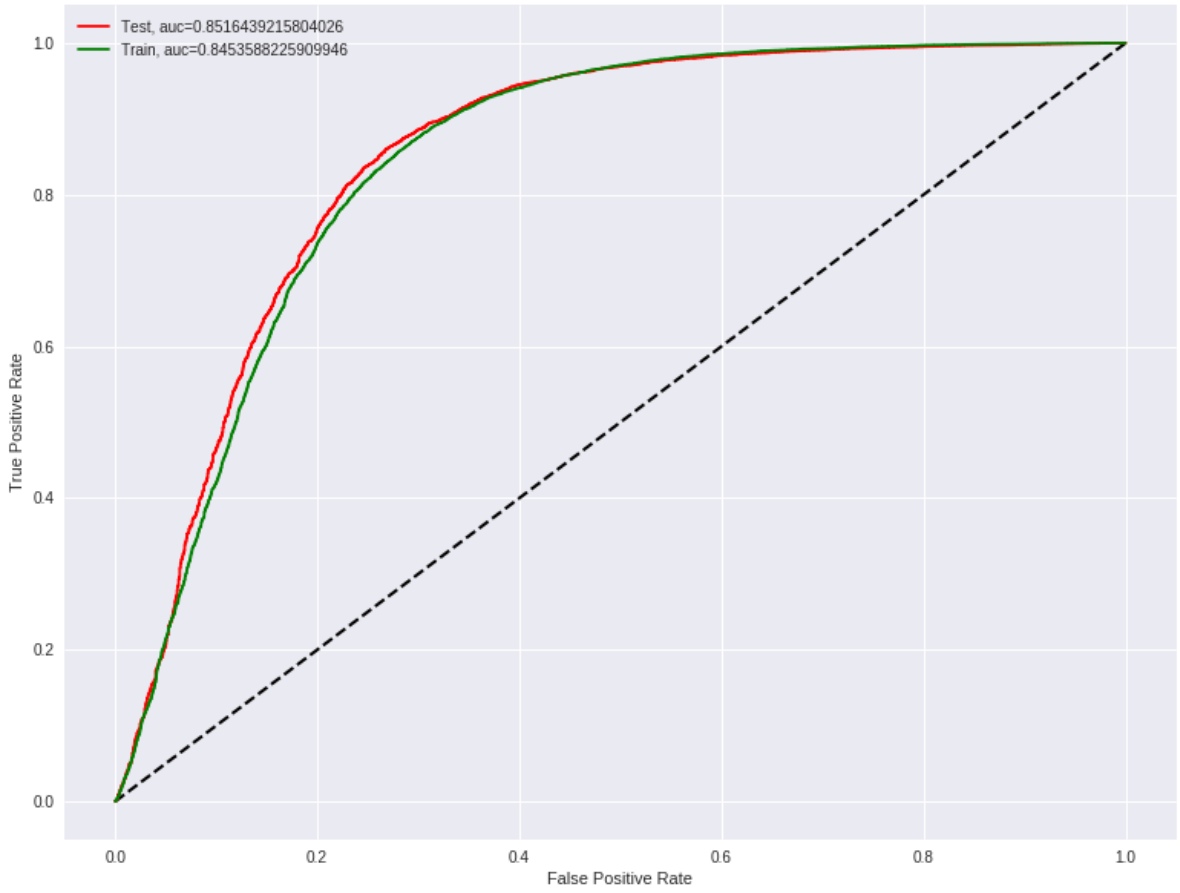
```
0.8516439215804026
0.8453588225909946
```

In [193]:

```
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 [196]:

```
#https://stackoverflow.com/questions/26976362/how-to-get-most-informative-features-for-scik
neg_features_labels = []
neg_features_coeff = []
neg_features_feat = []

pos_features_labels = []
pos_features_coeff = []
pos_features_feat = []
def most_informative_feature_for_binary_classification(vectorizer, classifier, n=10):
    class_labels = classifier.classes_
    feature_names = vectorizer.get_feature_names()
    topn_class1 = sorted(zip(classifier.coef_[0], feature_names))[:n]
    topn_class2 = sorted(zip(classifier.coef_[0], feature_names))[-n:]

    for coef, feat in topn_class1:
        neg_features_labels.append(class_labels[0])
        neg_features_coeff.append(coef)
        neg_features_feat.append(feat)

    for coef, feat in reversed(topn_class2):
        pos_features_labels.append(class_labels[1])
        pos_features_coeff.append(coef)
        pos_features_feat.append(feat)

    neg_df = pd.DataFrame({'Labels': neg_features_labels,'Coeff':neg_features_coeff ,'Negat
    pos_df = pd.DataFrame({'Labels': pos_features_labels,'Coeff':pos_features_coeff ,'Posit
    print("Top 10 featues for negative class \n", neg_df)
    print("Top 10 featues for positive class \n", pos_df)

f = most_informative_feature_for_binary_classification(count_vect, bow_svm_linear)
```

```
Top 10 featues for negative class
       Coeff  Labels Negative features
0 -2.668427       0             crabs
1 -2.068781       0            weignt
2 -2.048792       0            manged
3 -2.015479       0           grabbed
4 -2.012147       0         amplified
5 -2.005485       0        surpassing
6 -1.642365       0        b0006muf6g
7 -1.602389       0             crack
8 -1.559081       0           smelled
9 -1.405838       0           anguish
Top 10 featues for positive class
       Coeff  Labels Positive features
0  3.521258       1              food
1  3.234760       1        attendance
2  3.148144       1           concurs
3  2.761706       1            divina
4  2.271994       1          orderred
5  2.235349       1            waxier
6  2.158728       1            kidney
7  1.945520       1            mimics
8  1.922200       1              sued
9  1.902212       1           wizards
```

In [0]:

```
bow_test_conf = bow_svm_linear.predict(BoW_dict['X_test_vect'])
```

In [0]:

```
bow_train_conf = bow_svm_linear.predict(BoW_dict['X_train_vect'])
```

In [199]:

```
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)
```

```
[[  660  2016]
 [   98 17226]]
              precision    recall  f1-score   support

           0       0.87      0.25      0.38      2676
           1       0.90      0.99      0.94     17324

   micro avg       0.89      0.89      0.89     20000
   macro avg       0.88      0.62      0.66     20000
weighted avg       0.89      0.89      0.87     20000
```

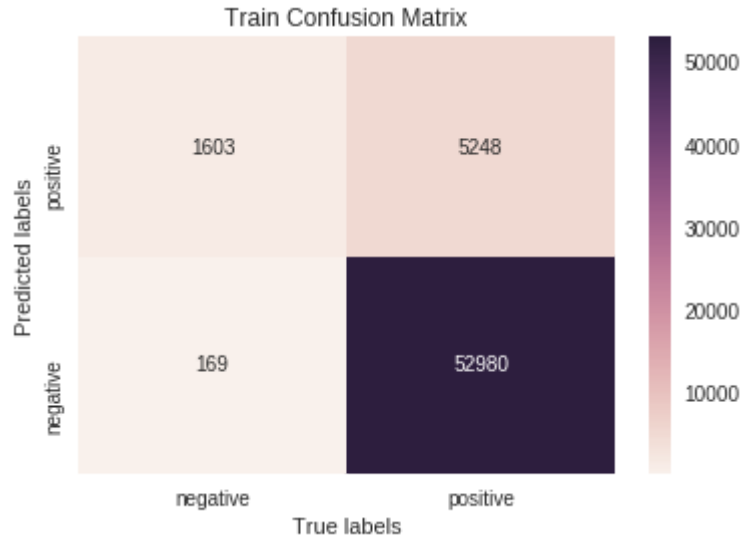
In [200]:

```
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[200]:

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



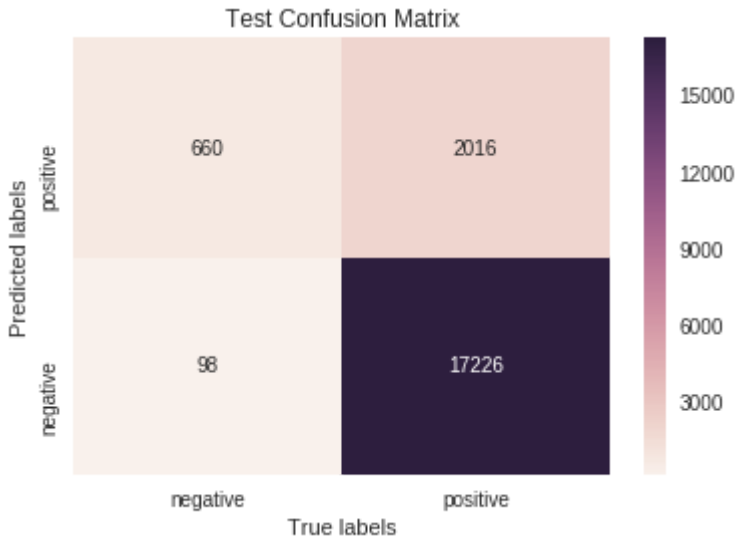
In [201]:

```
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[201]:

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



SVM on TF-IDF (Linear)

In [152]:

```
import pickle
# with open(r"/content/gdrive/My Drive/Colab Notebooks/SVM/tf_idf.pkl", "rb") as input_file
with open(r"new_tf_idf.pkl", "rb") as input_file:
    tfidf_dict = pickle.load(input_file)
```

In [153]:

```
tfidf_lgr_train_score_list = []
tfidf_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 all_a in tqdm([0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 1000]):
    tfidfclf = linear_model.SGDClassifier(loss='hinge', penalty='l2', alpha=all_a, class_we
    tfidfclf.fit(tfidf_dict['train_tf_idf'],Y_train)
    train_proba = tfidfclf.decision_function(tfidf_dict['train_tf_idf'])
    val_proba = tfidfclf.decision_function(tfidf_dict['cv_tf_idf'])

    fpr[all_a], tpr[all_a], _ = roc_curve(Y_train, train_proba)
    roc_auc_train[all_a] = auc(fpr[all_a], tpr[all_a])
    tfidf_lgr_train_score_list.append(auc(fpr[all_a], tpr[all_a]))
    fpr_val[all_a], tpr_val[all_a], _ = roc_curve(Y_val, val_proba)
    roc_auc_val[all_a] = auc(fpr_val[all_a], tpr_val[all_a])
    tfidf_lgr_val_score_list.append(auc(fpr_val[all_a], tpr_val[all_a]))
print(roc_auc_train)
print(roc_auc_val)
```

```
100%|████████████████████████████████████████████████████████████| 9/9 [0
0:01<00:00,  6.40it/s]

{0.1: 0.92735916857771761, 1: 0.92735916995087708, 100: 0.85135541360206457,
1000: 0.53044372142233964, 0.0001: 0.98819712687881733, 10: 0.92735895299169
946, 0.01: 0.92735585239788187, 0.001: 0.96068148926458941}
{0.1: 0.92797882614202032, 1: 0.9279788050229032, 100: 0.92797212082230074,
1000: 0.59572024468440277, 0.0001: 0.97296417200895857, 10: 0.92797902677363
42, 0.01: 0.92797858327217242, 0.001: 0.9562084967108242}
```

In [154]:

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

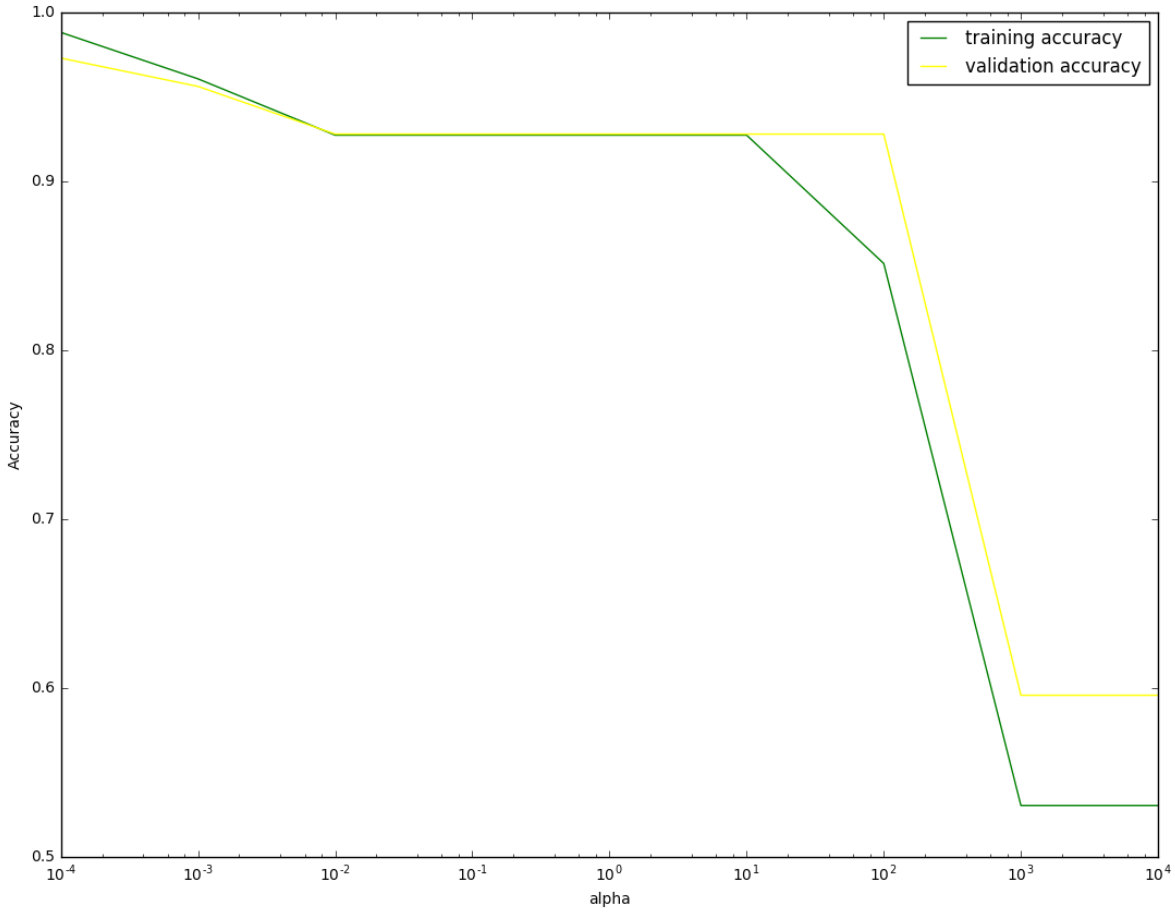
Out[154]:

```
0.0001
```

In [155]:

```
import pylab
plt.figure(figsize=(13, 10))
neighbors_settings = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
plt.plot(neighbors_settings, tfidf_lgr_train_score_list, label="training accuracy", color='
plt.plot(neighbors_settings, tfidf_lgr_val_score_list, label="validation accuracy", color='
# plt.plot(neighbors_settings, auc_test, label="test accuracy", color='red')
plt.xlabel('alpha')
plt.ylabel('Accuracy')
plt.legend()
plt.xscale('log')

plt.show()
```



In [156]:

```
tfidf_svm_linear=linear_model.SGDClassifier(loss='hinge', penalty='l2', alpha=tfidf_best_a)
tfidf_svm_linear.fit(tfidf_dict['train_tf_idf'], Y_train)
tf = CalibratedClassifierCV(base_estimator=tfidf_svm_linear)
tf.fit(tfidf_dict['train_tf_idf'], Y_train)
tfidf_linear_test_proba = tf.predict_proba(tfidf_dict['test_tf_idf'])
tfidf_linear_train_proba = tf.predict_proba(tfidf_dict['train_tf_idf'])
tfidf_linear_test_proba
```

Out[156]:

```
array([[  2.17118387e-03,   9.97828816e-01],
       [  8.32957417e-01,   1.67042583e-01],
       [  1.61635510e-02,   9.83836449e-01],
       ...,
       [  7.38629895e-02,   9.26137010e-01],
       [  1.31364681e-04,   9.99868635e-01],
       [  9.95505166e-01,   4.49483419e-03]])
```

In [157]:

```
#https://stackoverflow.com/questions/26976362/how-to-get-most-informative-features-for-scik
neg_features_labels = []
neg_features_coeff = []
neg_features_feat = []

pos_features_labels = []
pos_features_coeff = []
pos_features_feat = []
def most_informative_feature_for_binary_classification(vectorizer, classifier, n=10):
    class_labels = classifier.classes_
    feature_names = vectorizer.get_feature_names()
    topn_class1 = sorted(zip(classifier.coef_[0], feature_names))[:n]
    topn_class2 = sorted(zip(classifier.coef_[0], feature_names))[-n:]

    for coef, feat in topn_class1:
        neg_features_labels.append(class_labels[0])
        neg_features_coeff.append(coef)
        neg_features_feat.append(feat)

    for coef, feat in reversed(topn_class2):
        pos_features_labels.append(class_labels[1])
        pos_features_coeff.append(coef)
        pos_features_feat.append(feat)

    neg_df = pd.DataFrame({'Labels': neg_features_labels,'Coeff':neg_features_coeff ,'Negat
    pos_df = pd.DataFrame({'Labels': pos_features_labels,'Coeff':pos_features_coeff ,'Posit
    print("Top 10 featues for negative class \n", neg_df)
    print("Top 10 featues for positive class \n", pos_df)

f = most_informative_feature_for_binary_classification(tf_idf_vect, tfidf_svm_linear)
```

```
Top 10 featues for negative class
       Coeff  Labels Negative features
0 -3.815412       0      disappointed
1 -3.590785       0             worst
2 -3.380196       0               not
3 -3.187013       0          terrible
4 -3.165205       0             awful
5 -3.068079       0          horrible
6 -3.009071       0         not worth
7 -2.821293       0          not good
8 -2.778394       0     disappointing
9 -2.325796       0              yuck
Top 10 featues for positive class
       Coeff  Labels Positive features
0  3.638751       1             great
1  3.010031       1              best
2  2.727139       1              good
3  2.459756       1         delicious
4  2.320165       1         excellent
5  2.122157       1  not disappointed
6  1.989215       1              love
7  1.963226       1         wonderful
8  1.783980       1             loves
9  1.739815       1             yummy
```

In [158]:

```
tfidf_fpr_train, tfidf_tpr_train, _ = roc_curve(Y_train, tfidf_linear_train_proba[:, 1])
tfidf_fpr_test, tfidf_tpr_test, _ = roc_curve(Y_test, tfidf_linear_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)
```

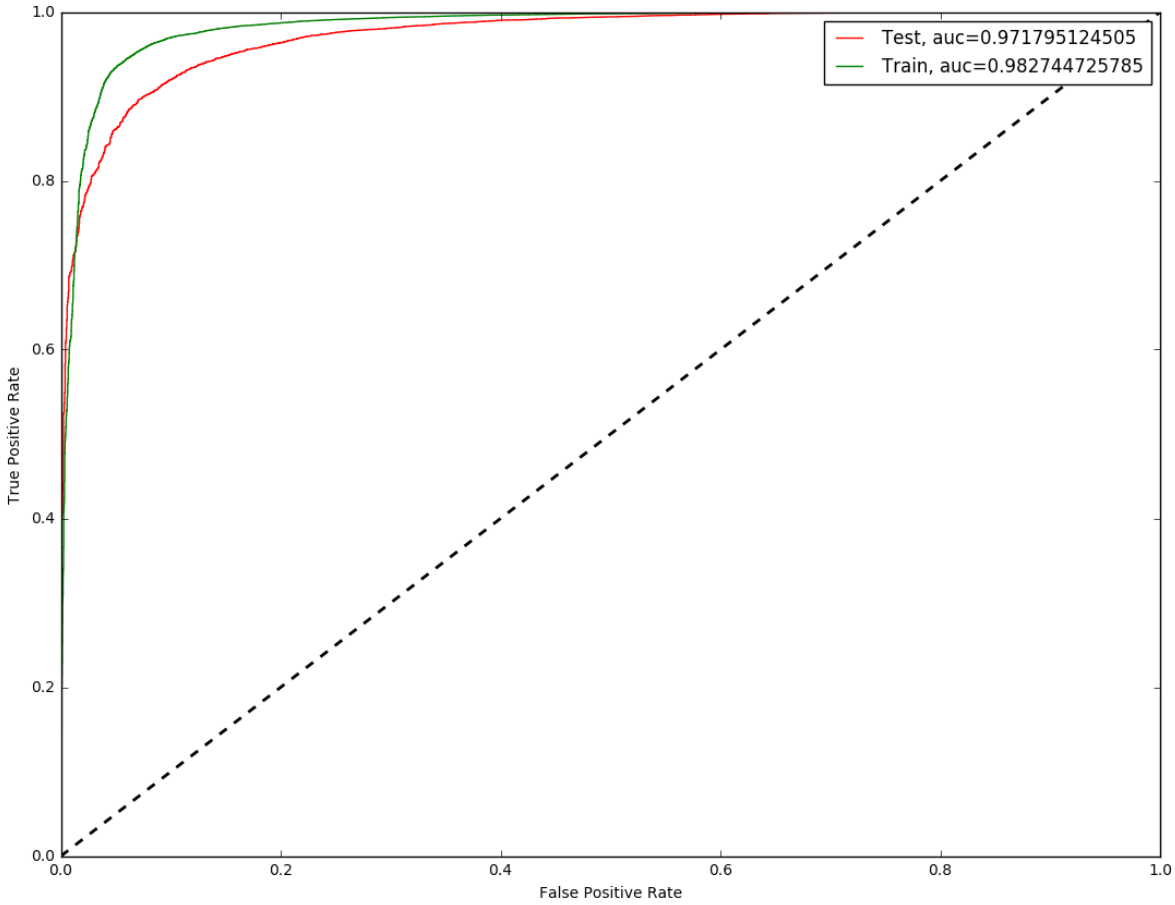
```
0.971795124505
0.982744725785
```

In [159]:

```
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 [160]:

```
tfidf_svm=linear_model.SGDClassifier(loss='hinge', penalty='l2', alpha=tfidf_best_a)
tfidf_svm.fit(tfidf_dict['train_tf_idf'], Y_train)
tfidf_test_conf = tfidf_svm.predict(tfidf_dict['test_tf_idf'])
tfidf_train_conf = tfidf_svm.predict(tfidf_dict['train_tf_idf'])
```

In [161]:

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

```
[[ 1197  1479]
 [   69 17255]]
             precision    recall  f1-score   support

          0       0.95      0.45      0.61      2676
          1       0.92      1.00      0.96     17324

avg / total       0.92      0.92      0.91     20000
```

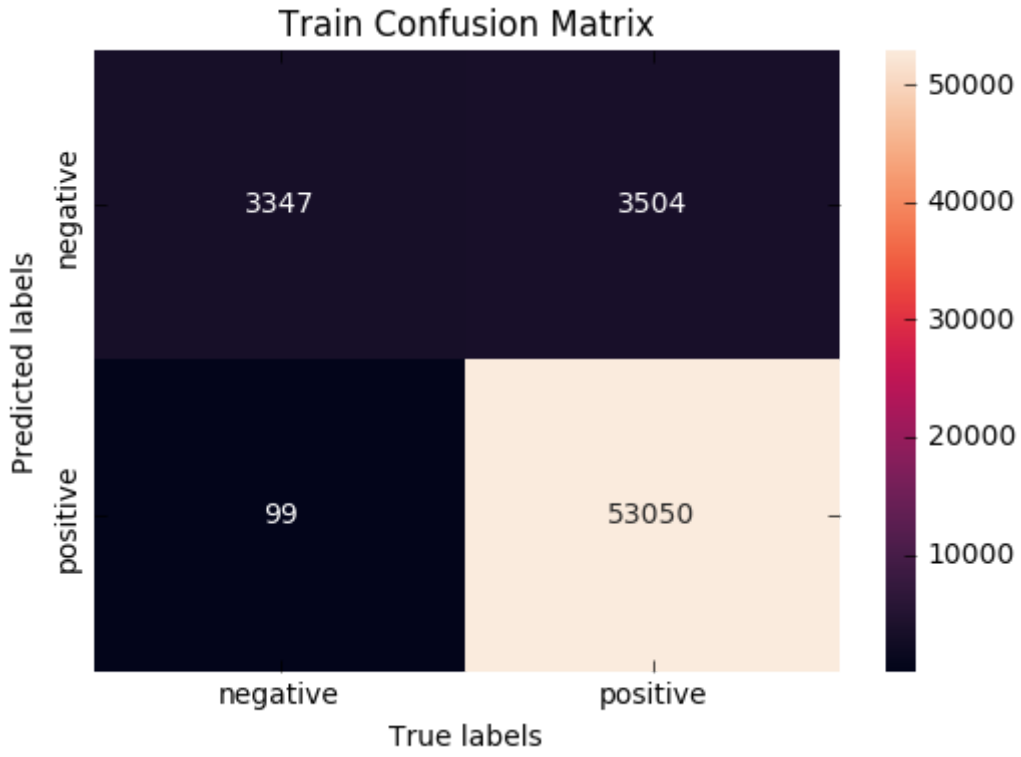
In [162]:

```
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[162]:

```
[<matplotlib.text.Text at 0x2fb21518>, <matplotlib.text.Text at 0x3c57b780>]
```



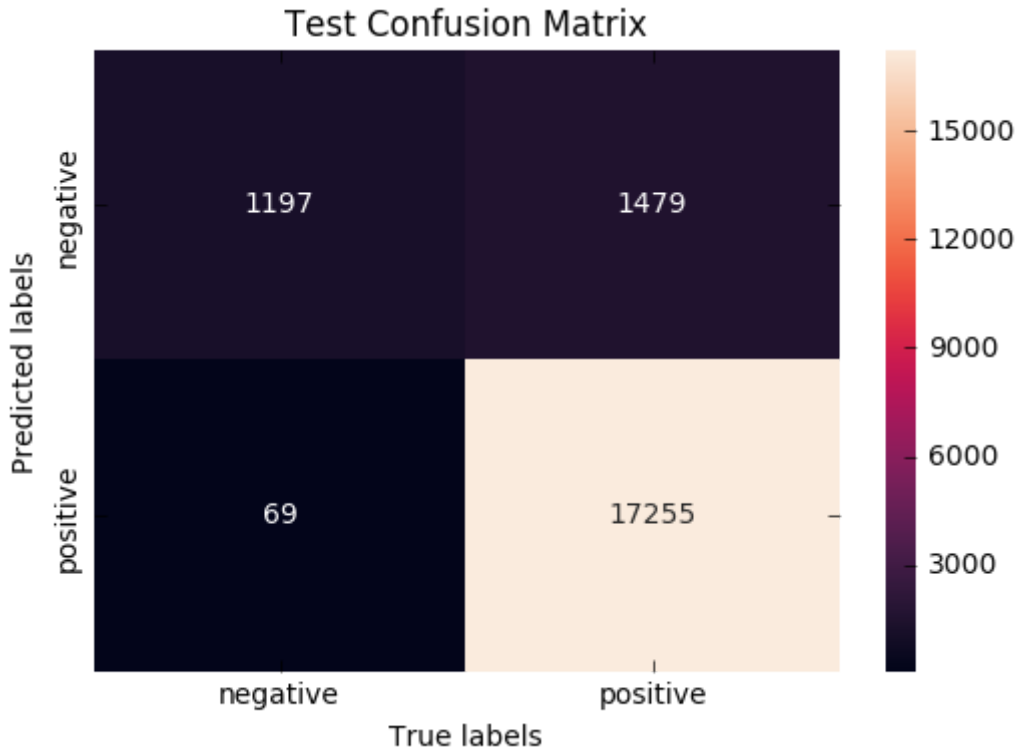
In [163]:

```
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[163]:

```
[<matplotlib.text.Text at 0x2f5aa080>, <matplotlib.text.Text at 0x38288160>]
```



SVM on Avg-W2V

In [302]:

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

In [304]:

```
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 all_a in tqdm([0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 1000]):
    tfidfclf = linear_model.SGDClassifier(loss='hinge', penalty='l2', alpha=all_a)
    tfidfclf.fit(avg_tfidf_dict['X_train_avgw2v'],Y_train)
    train_proba = tfidfclf.decision_function(avg_tfidf_dict['X_train_avgw2v'])
    val_proba = tfidfclf.decision_function(avg_tfidf_dict['X_val_avgw2v'])

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

```
  0%|
| 0/9 [00:00<?, ?it/s]

 11%|
| 1/9 [00:00<00:01,  4.29it/s]

 22%|
| 2/9 [00:00<00:01,  4.55it/s]

 33%|
| 3/9 [00:00<00:01,  4.58it/s]

 44%|
| 4/9 [00:00<00:01,  4.50it/s]

 56%|
                                                                | 5/9 [0
0:01<00:00,  4.24it/s]

 67%|
                                                                | 6/9 [0
0:01<00:00,  3.97it/s]

 78%|
                                                                | 7/9 [0
0:01<00:00,  3.68it/s]

 89%|
                                                                | 8/9 [0
0:02<00:00,  3.79it/s]

100%|
                                                               | 9/9 [0
```

```
0:02<00:00,  3.80it/s]


{0.1: 0.91895987001937218, 1: 0.91757792793983239, 100: 0.615484306753594, 1
000: 0.61548424633458254, 0.0001: 0.91154149745647362, 10: 0.754133467117868
88, 0.01: 0.91956099249640089, 0.001: 0.92060980062442999}
{0.1: 0.92621661532186894, 1: 0.92477350380346846, 100: 0.62415214136021979,
1000: 0.62415206744330953, 0.0001: 0.92054472792494701, 10: 0.77170388446457
461, 0.01: 0.92673613504611063, 0.001: 0.92732202159624022}
```

In [305]:

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

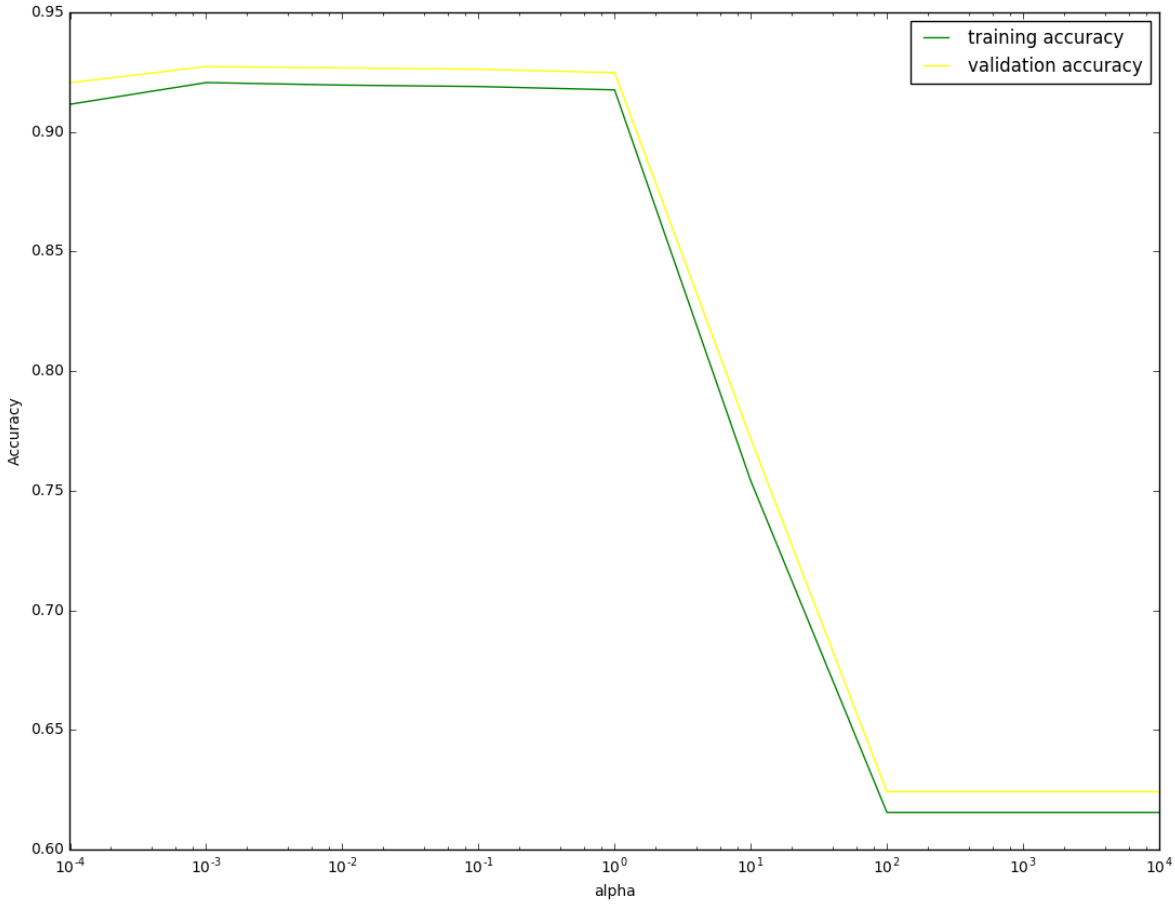
Out[305]:

```
0.001
```

In [306]:

```
import pylab
plt.figure(figsize=(13, 10))
neighbors_settings = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
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('alpha')
plt.ylabel('Accuracy')
plt.legend()
plt.xscale('log')

plt.show()
```



In [307]:

```
avg_svm_linear=linear_model.SGDClassifier(loss='hinge', penalty='l2', alpha=best_a)
avg_svm_linear.fit(avg_tfidf_dict['X_train_avgw2v'], Y_train)
f = CalibratedClassifierCV(base_estimator=avg_svm_linear)
f.fit(avg_tfidf_dict['X_train_avgw2v'], Y_train)
avg_linear_test_proba = f.predict_proba(avg_tfidf_dict['X_test_avgw2v'])
avg_linear_train_proba = f.predict_proba(avg_tfidf_dict['X_train_avgw2v'])
avg_linear_test_proba
```

Out[307]:

```
array([[  2.10623073e-03,   9.97893769e-01],
       [  8.85022931e-01,   1.14977069e-01],
       [  2.10171234e-02,   9.78982877e-01],
       ...,
       [  8.07373824e-02,   9.19262618e-01],
       [  8.99828020e-06,   9.99991002e-01],
       [  2.66416577e-01,   7.33583423e-01]])
```

In [308]:

```
avg_fpr_train, avg_tpr_train, _ = roc_curve(Y_train, avg_linear_train_proba[:, 1])
avg_fpr_test, avg_tpr_test, _ = roc_curve(Y_test, avg_linear_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)
```

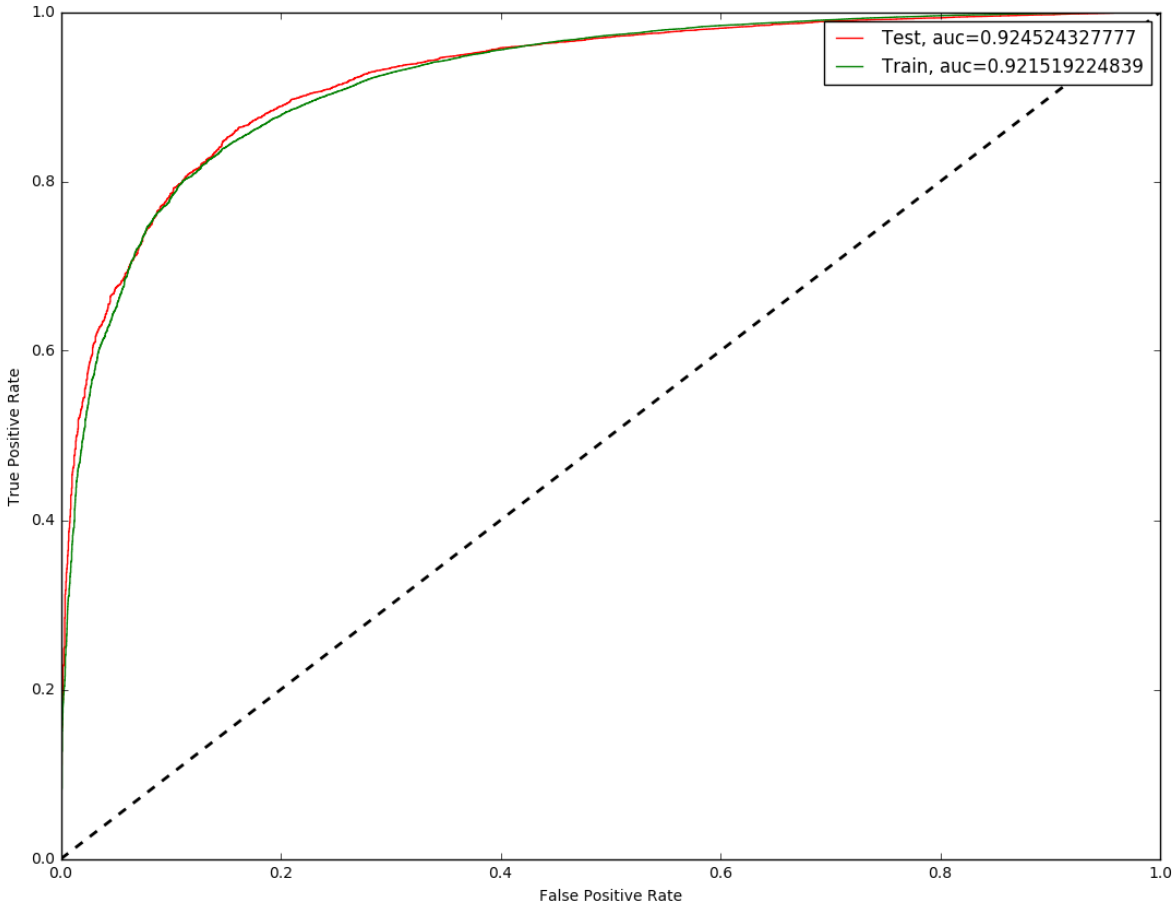
```
0.924524327777
0.921519224839
```

In [309]:

```
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 [310]:

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

In [311]:

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

```
[[  852  1824]
 [  224 17100]]
             precision    recall  f1-score   support

          0       0.79      0.32      0.45      2676
          1       0.90      0.99      0.94     17324

avg / total       0.89      0.90      0.88     20000
```

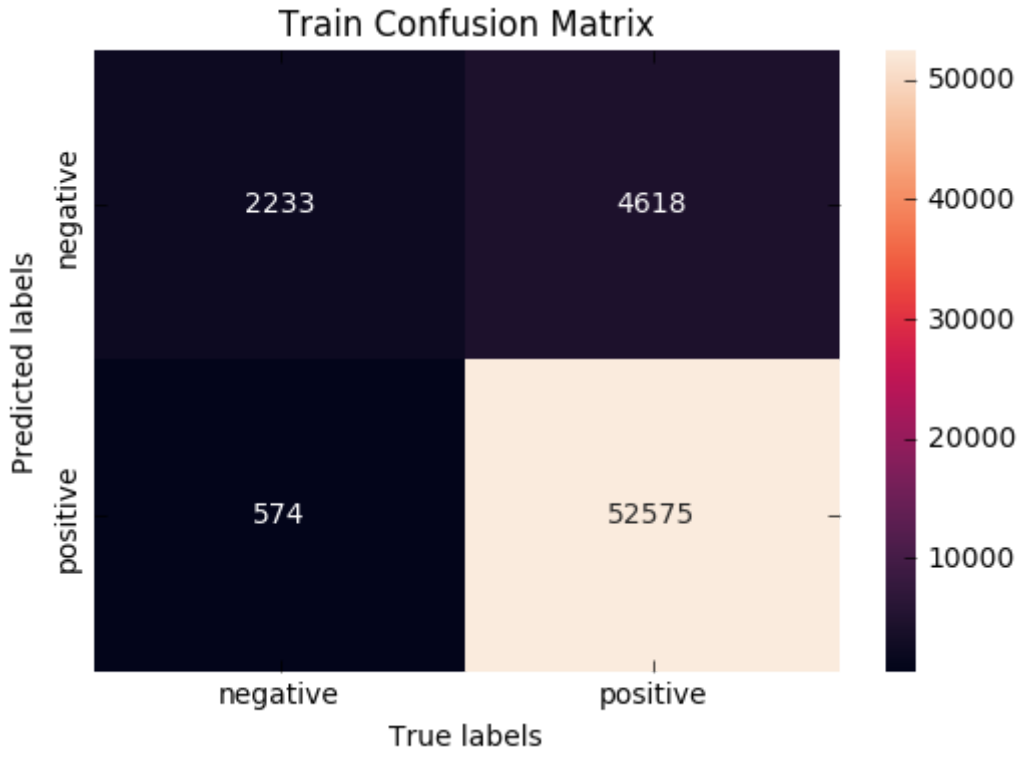
In [312]:

```
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[312]:

```
[<matplotlib.text.Text at 0x3344c128>, <matplotlib.text.Text at 0x2f9fdc18>]
```



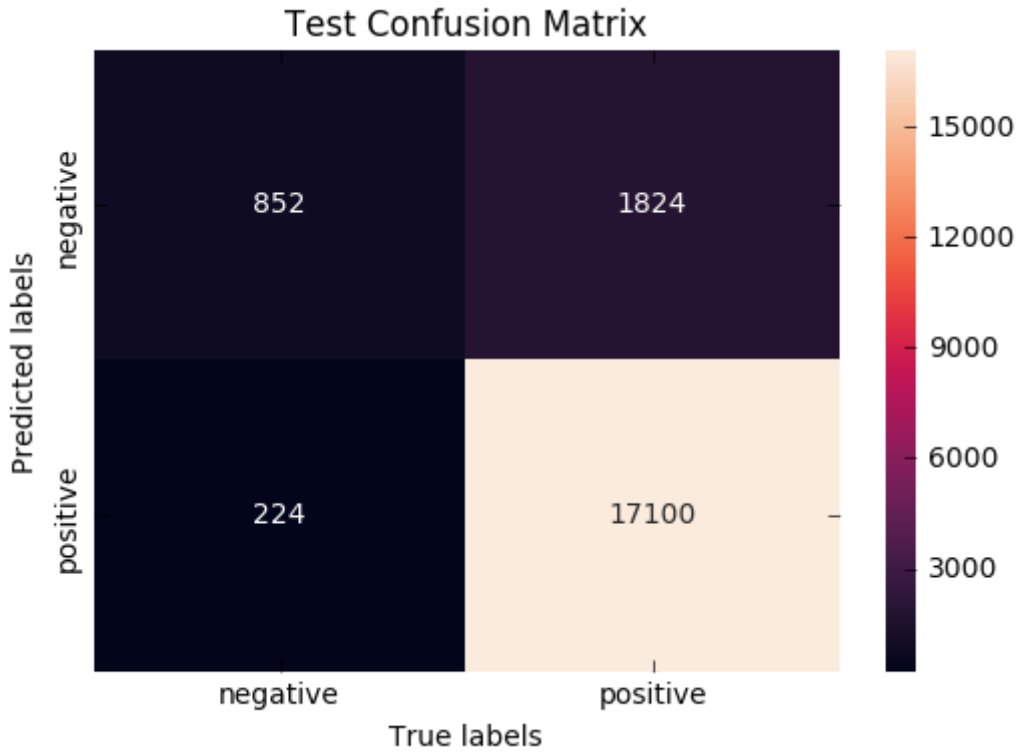
In [313]:

```
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[313]:

```
[<matplotlib.text.Text at 0x3c1db470>, <matplotlib.text.Text at 0x3c1bdfd0>]
```



SVM on tfidf_w2v

In [271]:

```
import pickle
# with open(r"/content/gdrive/My Drive/Colab Notebooks/SVM/tfidf_w2v.pkl", "rb") as input_f
with open(r"tfidf_w2v.pkl", "rb") as input_file:
    tfidfw2v_dict = pickle.load(input_file)
```

In [272]:

```
print(type(tfidfw2v_dict['X_train_tfidfw2v']))
```

```
<class 'list'>
```

In [291]:

```
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 all_a in tqdm([0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 1000]):
    tfidfclf = linear_model.SGDClassifier(loss='hinge', penalty='l2', alpha=all_a, class_we
    tfidfclf.fit(tfidfw2v_dict['X_train_tfidfw2v'],Y_train)
    train_proba = tfidfclf.decision_function(tfidfw2v_dict['X_train_tfidfw2v'])
    val_proba = tfidfclf.decision_function(tfidfw2v_dict['X_val_tfidfw2v'])

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

```
  0%|
| 0/9 [00:00<?, ?it/s]

 11%|
| 1/9 [00:00<00:01,  4.52it/s]

 22%|
| 2/9 [00:00<00:02,  3.00it/s]

 33%|
| 3/9 [00:00<00:01,  3.40it/s]

 44%|
| 4/9 [00:01<00:01,  3.63it/s]

 56%|
                                                                | 5/9 [0
0:01<00:01,  3.68it/s]

 67%|
                                                                | 6/9 [0
0:01<00:00,  3.89it/s]

 78%|
                                                                | 7/9 [0
0:01<00:00,  3.86it/s]

 89%|
                                                                | 8/9 [0
0:02<00:00,  3.88it/s]

100%|
                                                               | 9/9 [0
```

```
0:02<00:00,  3.70it/s]


{0.1: 0.83003216304463523, 1: 0.79175601482725377, 100: 0.79243478809249712,
1000: 0.79243478534617839, 0.0001: 0.81650335357508452, 10: 0.79243621480506
41, 0.01: 0.84109241648332911, 0.001: 0.83876241360428083}
{0.1: 0.8357907960521781, 1: 0.7948151046629659, 100: 0.79592297131449441, 1
000: 0.79592234830053621, 0.0001: 0.81665152214770276, 10: 0.795923045231404
78, 0.01: 0.84611929040441924, 0.001: 0.8417343280944406}
```

In [292]:

```
tfidf_w2v_best_a = max(roc_auc_val, key=roc_auc_val.get)
tfidf_w2v_best_a
```

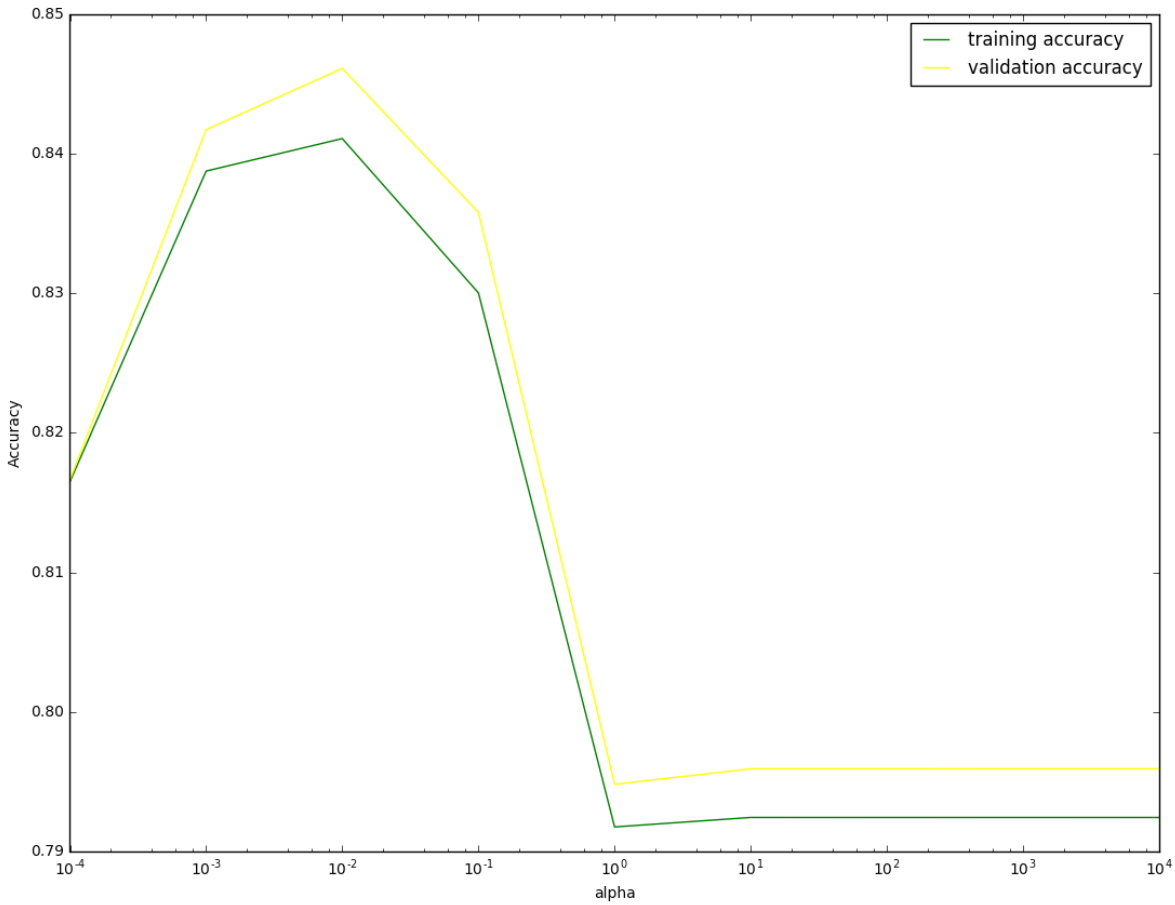
Out[292]:

```
0.01
```

In [293]:

```
import pylab
plt.figure(figsize=(13, 10))
neighbors_settings = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
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.plot(neighbors_settings, auc_test, label="test accuracy", color='red')
plt.xlabel('alpha')
plt.ylabel('Accuracy')
plt.legend()
plt.xscale('log')

plt.show()
```



In [294]:

```
tfidfw2v_svm_linear=linear_model.SGDClassifier(loss='hinge', penalty='l2', alpha=tfidf_w2v_
tfidfw2v_svm_linear.fit(tfidfw2v_dict['X_train_tfidfw2v'], Y_train)
f = CalibratedClassifierCV(base_estimator=tfidfw2v_svm_linear)
f.fit(tfidfw2v_dict['X_train_tfidfw2v'], Y_train)
tfidfw2v_test_proba = f.predict_proba(tfidfw2v_dict['X_test_tfidfw2v'])
tfidfw2v_train_proba = f.predict_proba(tfidfw2v_dict['X_train_tfidfw2v'])
tfidfw2v_test_proba
```

Out[294]:

```
array([[ 0.18381422,  0.81618578],
       [ 0.20552587,  0.79447413],
       [ 0.06864009,  0.93135991],
       ...,
       [ 0.13215745,  0.86784255],
       [ 0.00328004,  0.99671996],
       [ 0.607836  ,  0.392164  ]])
```

In [295]:

```
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)
```

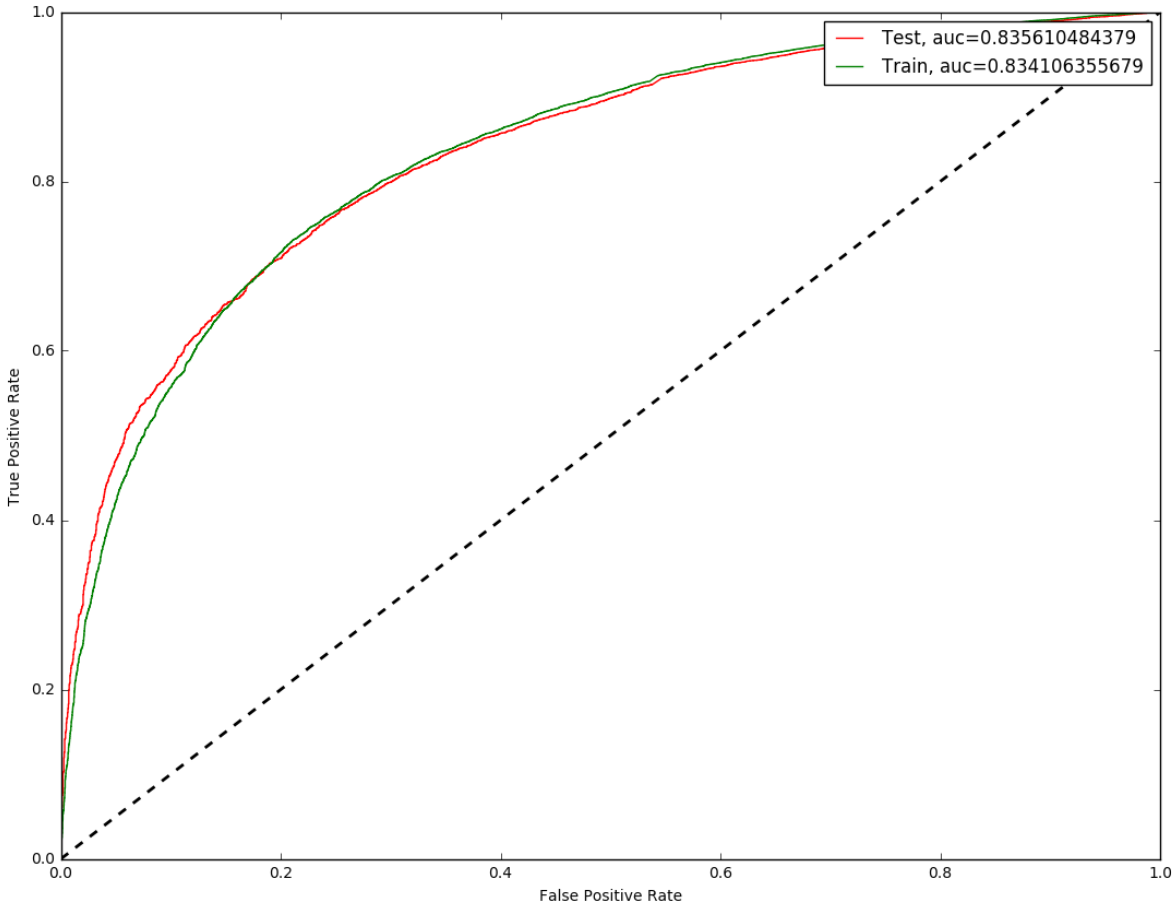
```
0.835610484379
0.834106355679
```

In [296]:

```
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 [297]:

```
tfidfw2v_test_conf = tfidfw2v_svm_linear.predict(tfidfw2v_dict['X_test_tfidfw2v'])
tfidfw2v_train_conf = tfidfw2v_svm_linear.predict(tfidfw2v_dict['X_train_tfidfw2v'])
```

In [319]:

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

```
[[ 2233  4618]
 [  574 52575]]
             precision    recall  f1-score   support

          0       0.79      0.32      0.45      2676
          1       0.90      0.99      0.94     17324

avg / total       0.89      0.90      0.88     20000
```

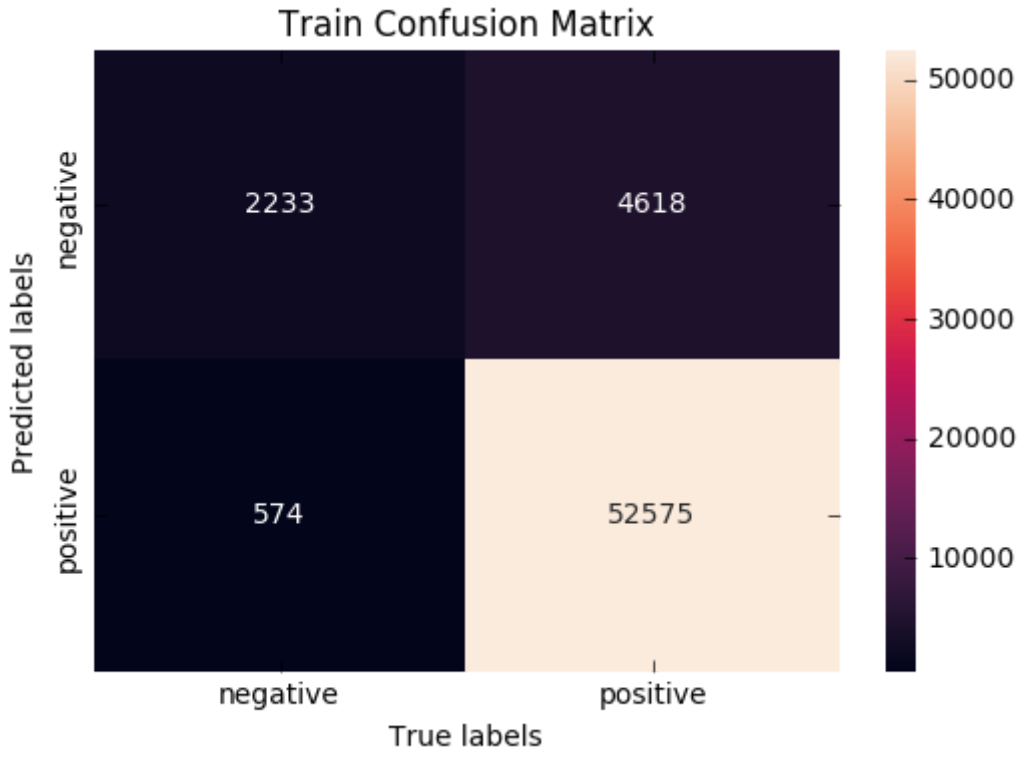
In [315]:

```
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[315]:

```
[<matplotlib.text.Text at 0x37ff5b70>, <matplotlib.text.Text at 0x37f61898>]
```



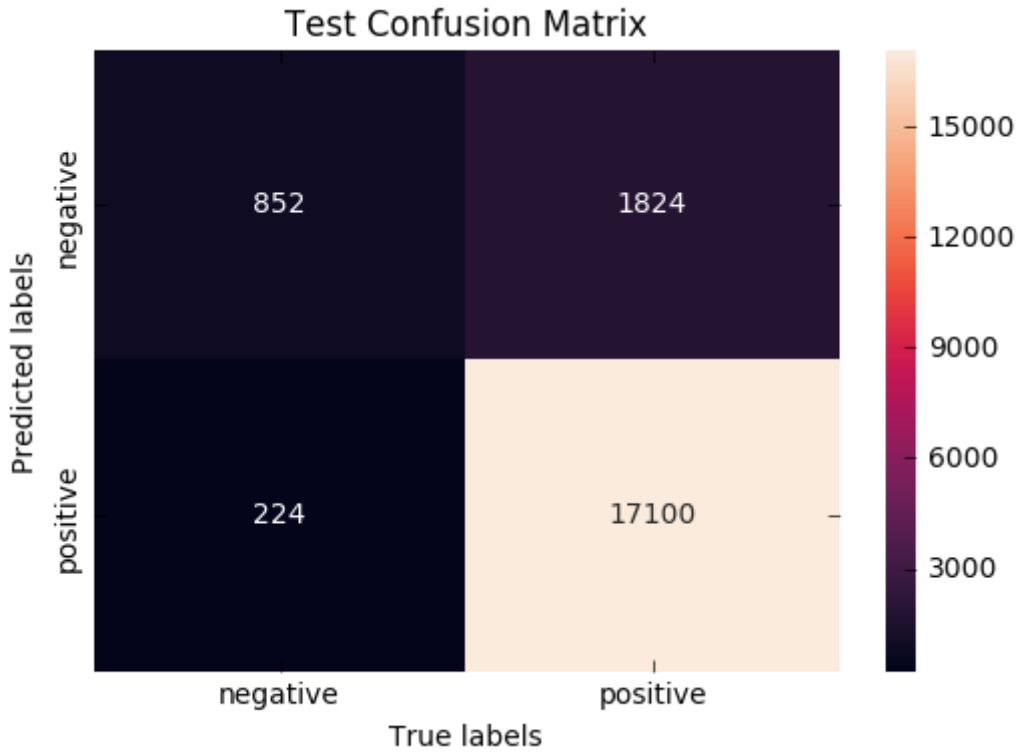
In [316]:

```
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[316]:

```
[<matplotlib.text.Text at 0x3afdbda0>, <matplotlib.text.Text at 0x397accc0>]
```



RBF Kernel

In [66]:

```
#Data

X_train = X[0:20000]
Y_train = y[0:20000]
X_val = X[20000:25000]
Y_val = y[20000:25000]
X_test = X[25000:30000]
Y_test = y[25000:30000]
```

In [67]:

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

```
20000 5000 5000
20000 5000 5000
```

BoW on 20k

In [0]:

```
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)
```

```
(20000, 29706)
```

In [0]:

```
from scipy.sparse import hstack
# len_review = final['lengthOfReview'].to_sparse()
concat_data = hstack((X_train_vect,np.array(final['lengthOfReview'][0:20000])[:,None]))
concat_data_val = hstack((X_val_vect,np.array(final['lengthOfReview'][20000:25000])[:,None]
concat_data_test = hstack((X_test_vect,np.array(final['lengthOfReview'][25000:30000])[:,Nor
```

In [0]:

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

```
(20000, 29707)
(5000, 29707)
(5000, 29707)
```

In [0]:

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

```
(20000, 29707)
```

In [0]:

```
import pickle
with open('/content/gdrive/My Drive/Colab Notebooks/SVM/30kBoW.pkl', 'wb') as handle:
    pickle.dump(BoW_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

[4.3] TF-IDF

In [0]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10, max_features=500)
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  (20000, 500)
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the number of unique words including both unigrams and bigrams  500
```

In [0]:

```
tfidf_concat_data_train = hstack((train_tf_idf,np.array(final['lengthOfReview'][0:20000])[:
tfidf_concat_data_val = hstack((cv_tf_idf,np.array(final['lengthOfReview'][20000:25000])[:,
tfidf_concat_data_test = hstack((test_tf_idf,np.array(final['lengthOfReview'][25000:30000])
```

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/SVM/30ktf_idf.pkl', 'wb') as handle:
    pickle.dump(tf_idf_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

[4.4] Word2Vec

In [0]:

```
# 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 [0]:

```
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
```

```
[('wonderful', 0.9070028066635132), ('good', 0.8153142929077148), ('perfec
t', 0.7934567332267761), ('nice', 0.7591120600700378), ('fantastic', 0.73220
80731391907), ('delicious', 0.7165140509605408), ('special', 0.6991521120071
411), ('excellent', 0.6706159710884094), ('decent', 0.6646473407745361), ('g
reat.', 0.6598770022392273)]
==================================================
[('best.', 0.8320965766906738), ('best', 0.8228322267532349), ('best-tastin
g', 0.8215420842170715), ('nicest', 0.810248076915741), ('greatest', 0.79618
10231208801), ('best!', 0.7789650559425354), ('had.', 0.7666232585906982),
('Best', 0.7660649418830872), ('tried.', 0.7629457116127014), ('made.', 0.75
9280800819397)]
```

In [0]:

```
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  16103
sample words  ['EVERY', 'book', 'is', 'this', 'little', 'makes', 'my', 'so
n', 'laugh', 'at', 'i', 'it', 'in', 'the', 'car', 'as', "we're", 'driving',
'along', 'and', 'he', 'always', 'can', 'sing', "he's", 'learned', 'about',
'India,', 'love', 'all', 'new', 'words', 'of', 'all.', 'a', 'classic', 'am',
'willing', 'to', 'bet', 'will', 'STILL', 'be', 'able', 'from', 'memory', 'wh
en', 'college', 'This', 'whole']
```

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

[4.4.1.1] Avg W2v

In [0]:

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

```
EVERY book is educational. this witty little book makes my son laugh at lou
d. i recite it in the car as we're driving along and he always can sing the
refrain. he's learned about whales, India, drooping roses:  i love all the n
ew words this book  introduces and the silliness of it all.  this is a class
ic book i am  willing to bet my son will STILL be able to recite from memory
when he is  in college
5000
5000
```

In [0]:

```
# 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 [0]:

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

```
5000
```

In [0]:

```
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])
```

```
100%|██████████| 20000/20000 [01:54<00:00, 175.21it/s]
  1%|          | 28/5000 [00:00<00:17, 276.46it/s]

20000
50

100%|██████████| 5000/5000 [00:27<00:00, 183.63it/s]
  0%|          | 25/5000 [00:00<00:20, 247.93it/s]

5000
50

100%|██████████| 5000/5000 [00:30<00:00, 165.08it/s]

5000
50
```

In [0]:

```
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 [0]:

```
import pickle
with open('/content/gdrive/My Drive/Colab Notebooks/SVM/30kavg_w2v.pkl', 'wb') as handle:
    pickle.dump(Avg_w2v_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

TFIDF-w2v

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer(min_df=10, max_features=500)
tf_idf_matrix = model.fit_transform(X_train)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [0]:

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

def 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 [0]:

```
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])
```

```
100%|██████████| 20000/20000 [02:07<00:00, 156.36it/s]
100%|██████████| 5000/5000 [00:29<00:00, 168.92it/s]
100%|██████████| 5000/5000 [00:34<00:00, 145.98it/s]
```

In [0]:

```
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 [0]:

```
with open('/content/gdrive/My Drive/Colab Notebooks/SVM/30ktfidf_w2v.pkl', 'wb') as handle:
    pickle.dump(tfidf_w2v_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

SVM on BoW (RBF)

In [52]:

```
from sklearn.svm import SVC
```

In [0]:

```
import pickle
with open(r"/content/gdrive/My Drive/Colab Notebooks/SVM/30kBoW.pkl", "rb") as input_file:
# with open(r"30kBoW.pkl", "rb") as input_file:
    small_BoW_dict = pickle.load(input_file)
```

In [24]:

```
from tqdm import tqdm
bow_lgr_train_score_list = []
bow_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 c_value in tqdm([0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]):
    tfidfclf = SVC(C=c_value, kernel='rbf')
    tfidfclf.fit(small_BoW_dict['X_train_vect'],Y_train)
    train_proba = tfidfclf.decision_function(small_BoW_dict['X_train_vect'])
    val_proba = tfidfclf.decision_function(small_BoW_dict['X_val_vect'])

    fpr[c_value], tpr[c_value], _ = roc_curve(Y_train, train_proba)
    roc_auc_train[c_value] = auc(fpr[c_value], tpr[c_value])
    bow_lgr_train_score_list.append(auc(fpr[c_value], tpr[c_value]))
    fpr_val[c_value], tpr_val[c_value], _ = roc_curve(Y_val, val_proba)
    roc_auc_val[c_value] = auc(fpr_val[c_value], tpr_val[c_value])
    bow_lgr_val_score_list.append(auc(fpr_val[c_value], tpr_val[c_value]))
print(roc_auc_train)
print(roc_auc_val)
```

```
100%|██████████| 9/9 [22:31<00:00, 167.94s/it]

{0.0001: 0.5675004312940621, 0.001: 0.7552593158291232, 0.01: 0.908095604450
7223, 0.1: 0.9637019871226937, 1: 0.9651030635985978, 10: 0.965025284816066
3, 100: 0.9759159148631846, 1000: 0.9935561324155764, 10000: 0.9998860036164
423}
{0.0001: 0.5956686915631986, 0.001: 0.7624766466377566, 0.01: 0.891443902806
5199, 0.1: 0.9385242034226875, 1: 0.939254657372051, 10: 0.9396224085097692,
100: 0.9490235566544509, 1000: 0.9522995319425067, 10000: 0.938835646308062
3}
```

In [25]:

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

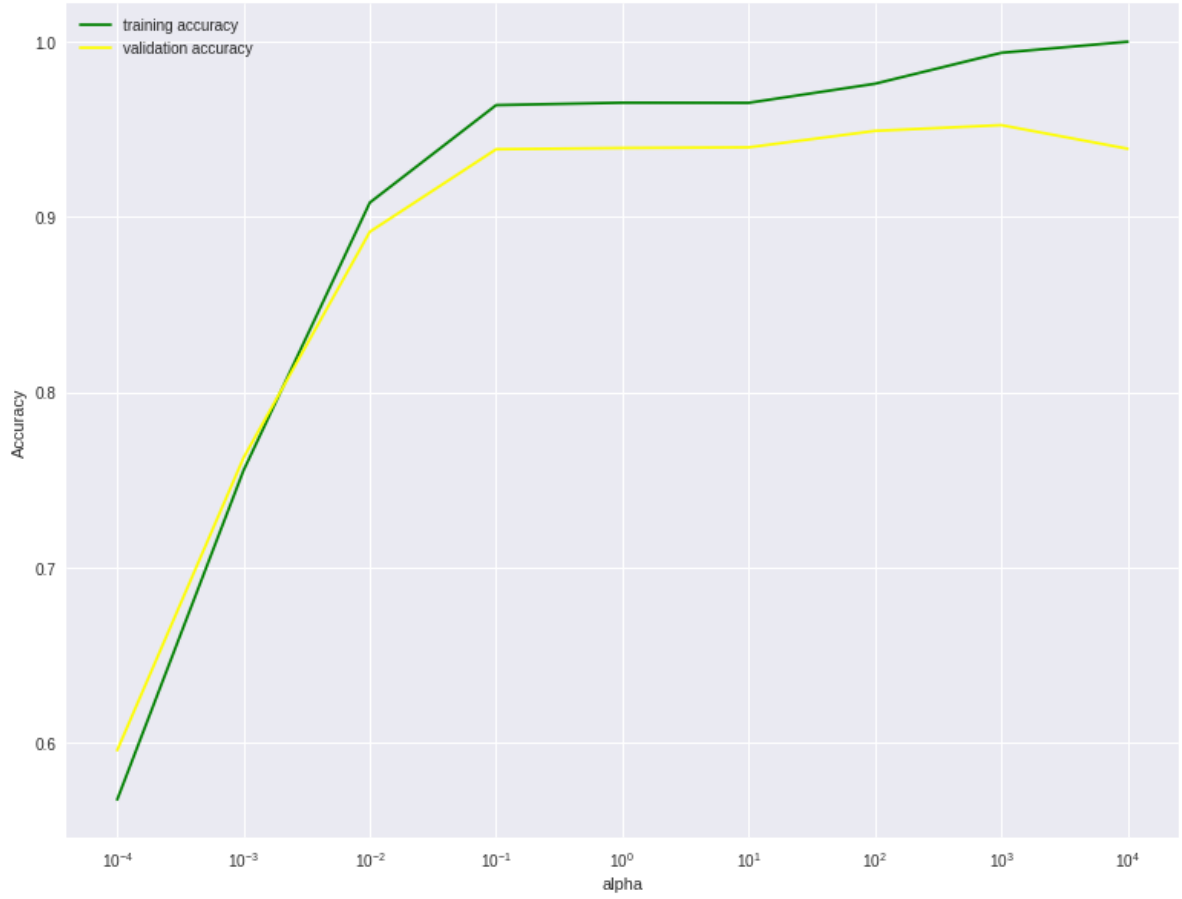
Out[25]:

```
1000
```

In [26]:

```
import pylab
plt.figure(figsize=(13, 10))
neighbors_settings = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
plt.plot(neighbors_settings, bow_lgr_train_score_list, label="training accuracy", color='gr
plt.plot(neighbors_settings, bow_lgr_val_score_list, label="validation accuracy", color='ye
# plt.plot(neighbors_settings, auc_test, label="test accuracy", color='red')
plt.xlabel('alpha')
plt.ylabel('Accuracy')
plt.legend()
plt.xscale('log')

plt.show()
```



In [27]:

```
bow_svm_linear=SVC(C=rbf_best_c, kernel='rbf')
bow_svm_linear.fit(small_BoW_dict['X_train_vect'],Y_train)
f = CalibratedClassifierCV(base_estimator=bow_svm_linear)
f.fit(small_BoW_dict['X_train_vect'],Y_train)
bow_linear_test_proba = f.predict_proba(small_BoW_dict['X_test_vect'])
bow_linear_train_proba = f.predict_proba(small_BoW_dict['X_train_vect'])
bow_linear_test_proba
```

Out[27]:

```
array([[0.1148381 , 0.8851619 ],
       [0.05897012, 0.94102988],
       [0.03864145, 0.96135855],
       ...,
       [0.02261815, 0.97738185],
       [0.02777235, 0.97222765],
       [0.1101549 , 0.8898451 ]])
```

In [28]:

```
bow_fpr_train, bow_tpr_train, _ = roc_curve(Y_train, bow_linear_train_proba[:, 1])
bow_fpr_test, bow_tpr_test, _ = roc_curve(Y_test, bow_linear_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)
```

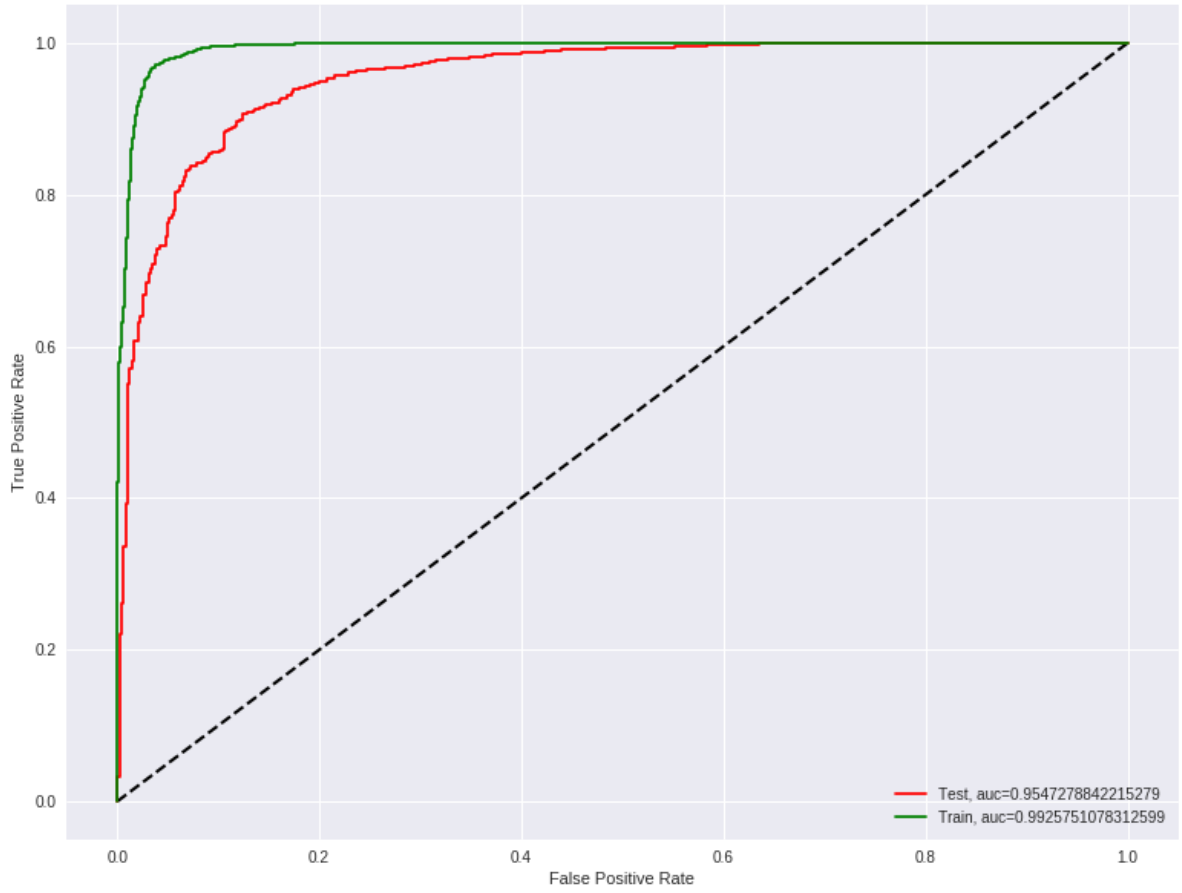
```
0.9547278842215279
0.9925751078312599
```

In [30]:

```
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 [0]:

```
bow_test_conf = bow_svm_linear.predict(small_BoW_dict['X_test_vect'])
```

In [0]:

```
bow_train_conf = bow_svm_linear.predict(small_BoW_dict['X_train_vect'])
```

In [34]:

```
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)
```

```
[[ 300  182]
 [  78 4440]]
              precision    recall  f1-score   support

           0       0.79      0.62      0.70       482
           1       0.96      0.98      0.97      4518

   micro avg       0.95      0.95      0.95      5000
   macro avg       0.88      0.80      0.83      5000
weighted avg       0.94      0.95      0.95      5000
```

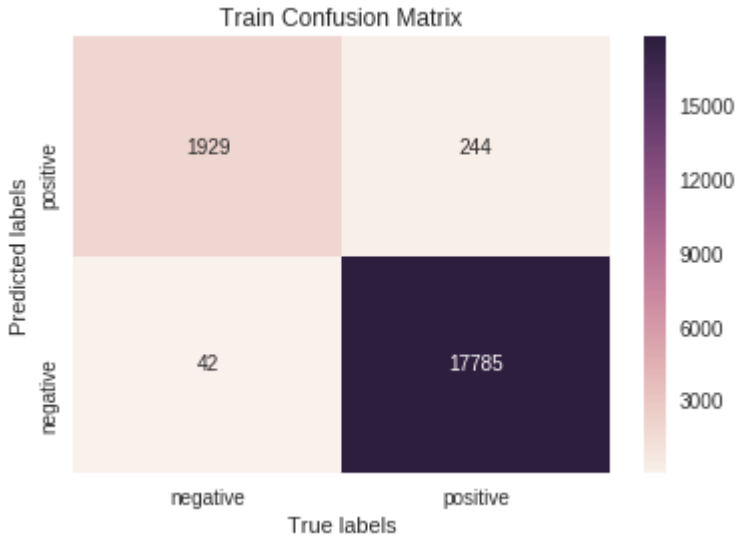
In [35]:

```
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[35]:

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



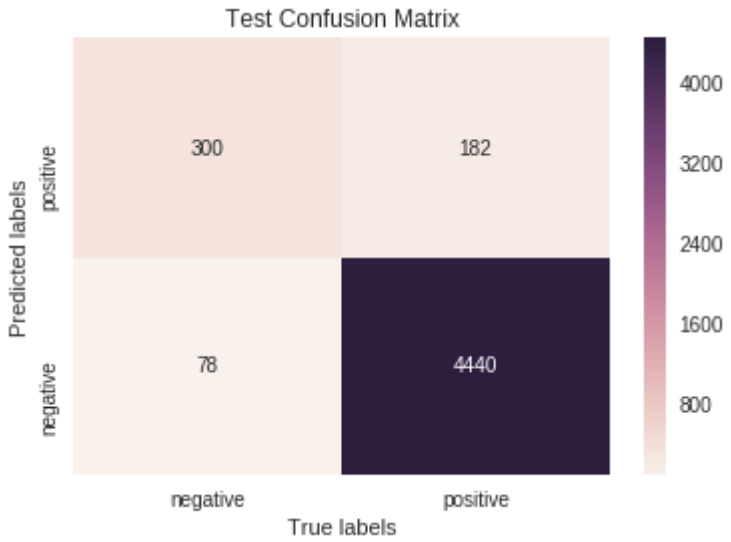
In [36]:

```
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[36]:

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



SVM on TF-IDF (RBF)

In [0]:

```
import pickle
with open(r"/content/gdrive/My Drive/Colab Notebooks/SVM/30ktf_idf.pkl", "rb") as input_fil
    small_tfidf_dict = pickle.load(input_file)
```

In [42]:

```
from tqdm import tqdm
tfidf_lgr_train_score_list = []
tfidf_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 c_value in tqdm([0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]):
    tfidfclf = SVC(C=c_value, kernel='rbf')
    tfidfclf.fit(small_tfidf_dict['train_tf_idf'],Y_train)
    train_proba = tfidfclf.decision_function(small_tfidf_dict['train_tf_idf'])
    val_proba = tfidfclf.decision_function(small_tfidf_dict['cv_tf_idf'])

    fpr[c_value], tpr[c_value], _ = roc_curve(Y_train, train_proba)
    roc_auc_train[c_value] = auc(fpr[c_value], tpr[c_value])
    tfidf_lgr_train_score_list.append(auc(fpr[c_value], tpr[c_value]))
    fpr_val[c_value], tpr_val[c_value], _ = roc_curve(Y_val, val_proba)
    roc_auc_val[c_value] = auc(fpr_val[c_value], tpr_val[c_value])
    tfidf_lgr_val_score_list.append(auc(fpr_val[c_value], tpr_val[c_value]))
print(roc_auc_train)
print(roc_auc_val)
```

```
100%|██████████| 9/9 [22:03<00:00, 194.72s/it]

{0.0001: 0.7571939242922034, 0.001: 0.7572555174469064, 0.01: 0.722039437637
46, 0.1: 0.9395271385609263, 1: 0.9490204610343143, 10: 0.9492186639856177,
100: 0.9514810894946214, 1000: 0.9721998805774297, 10000: 0.986526691016700
3}
{0.0001: 0.733338873547587, 0.001: 0.7333369318837131, 0.01: 0.6728579855392
641, 0.1: 0.8820435313273874, 1: 0.8959741929571192, 10: 0.8963427207603866,
100: 0.8982750646476987, 1000: 0.8926345310939995, 10000: 0.863993047290000
4}
```

In [44]:

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

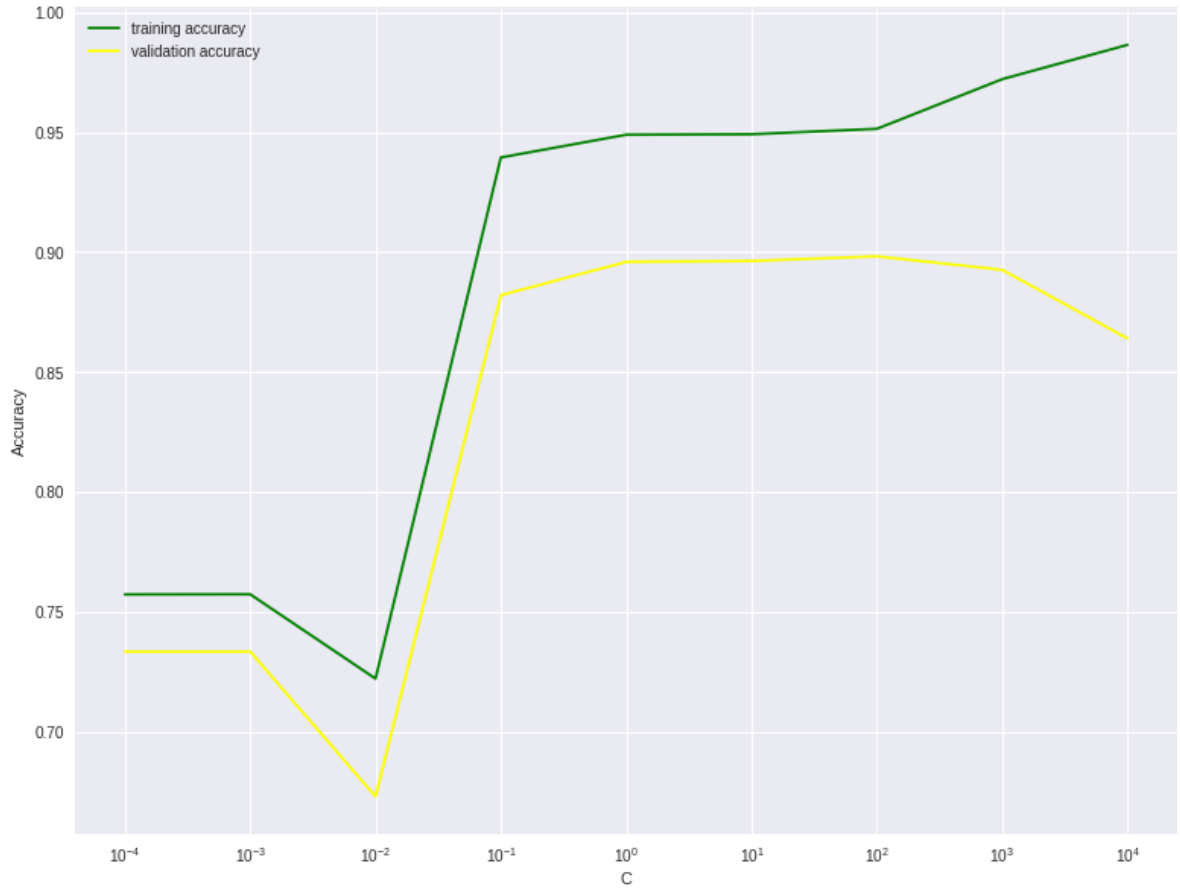
Out[44]:

```
100
```

In [45]:

```
import pylab
plt.figure(figsize=(13, 10))
neighbors_settings = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
plt.plot(neighbors_settings, tfidf_lgr_train_score_list, label="training accuracy", color='
plt.plot(neighbors_settings, tfidf_lgr_val_score_list, label="validation accuracy", color='
# plt.plot(neighbors_settings, auc_test, label="test accuracy", color='red')
plt.xlabel('C')
plt.ylabel('Accuracy')
plt.legend()
plt.xscale('log')

plt.show()
```



In [46]:

```
tfidf_svm_linear=SVC(C=tfidf_best_c, kernel='rbf')
tfidf_svm_linear.fit(tfidf_dict['train_tf_idf'], Y_train)
f = CalibratedClassifierCV(base_estimator=bow_svm_linear)
f.fit(tfidf_dict['train_tf_idf'], Y_train)
tfidf_linear_test_proba = f.predict_proba(tfidf_dict['test_tf_idf'])
tfidf_linear_train_proba = f.predict_proba(tfidf_dict['train_tf_idf'])
tfidf_linear_test_proba
```

Out[46]:

```
array([[0.06192611, 0.93807389],
       [0.05750108, 0.94249892],
       [0.09716267, 0.90283733],
       ...,
       [0.06753958, 0.93246042],
       [0.01556645, 0.98443355],
       [0.13025602, 0.86974398]])
```

In [47]:

```
tfidf_fpr_train, tfidf_tpr_train, _ = roc_curve(Y_train, tfidf_linear_train_proba[:, 1])
tfidf_fpr_test, tfidf_tpr_test, _ = roc_curve(Y_test, tfidf_linear_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)
```

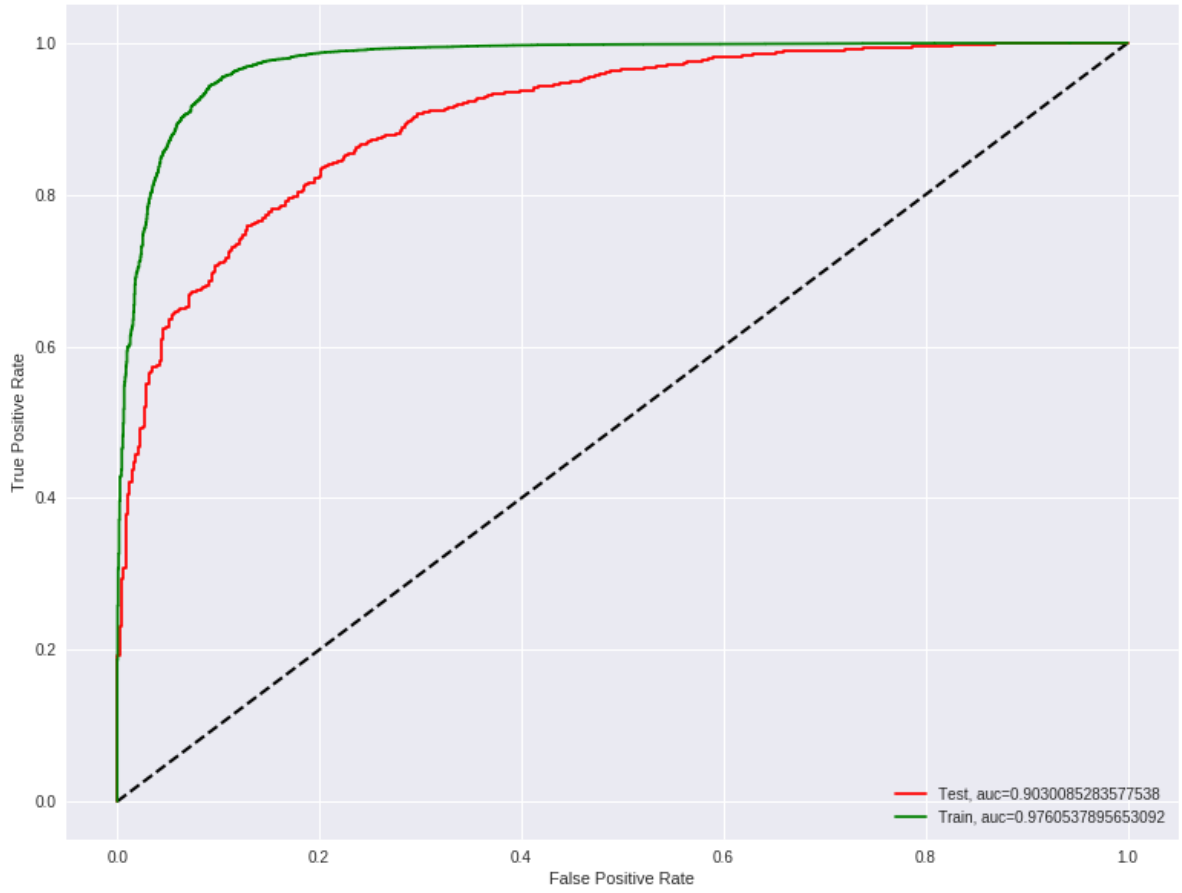
```
0.9030085283577538
0.9760537895653092
```

In [48]:

```
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 [0]:

```
tfidf_test_conf = tfidf_svm_linear.predict(tfidf_dict['test_tf_idf'])
```

In [0]:

```
tfidf_train_conf = tfidf_svm_linear.predict(tfidf_dict['train_tf_idf'])
```

In [51]:

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

```
[[  65  417]
 [  14 4504]]
              precision    recall  f1-score   support

           0       0.82      0.13      0.23       482
           1       0.92      1.00      0.95      4518

   micro avg       0.91      0.91      0.91      5000
   macro avg       0.87      0.57      0.59      5000
weighted avg       0.91      0.91      0.88      5000
```

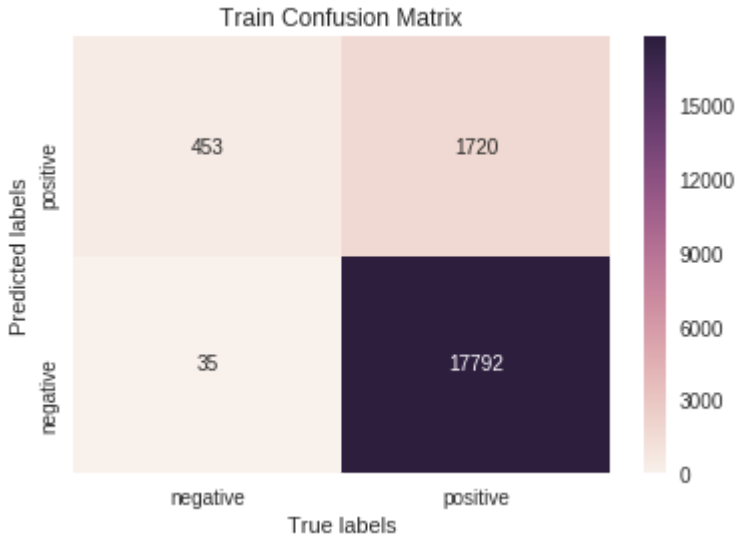
In [52]:

```
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[52]:

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



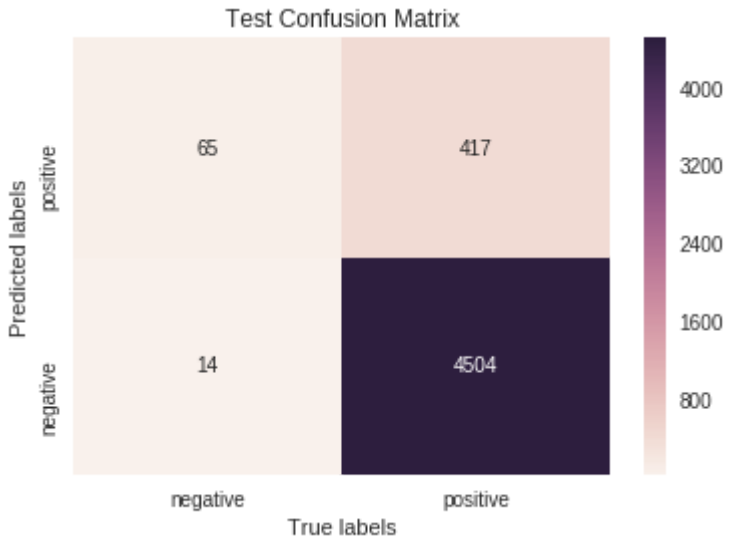
In [53]:

```
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[53]:

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



SVM on Avg-W2V (RBF)

In [0]:

```
import pickle
with open(r"/content/gdrive/My Drive/Colab Notebooks/SVM/30kavg_w2v.pkl", "rb") as input_fi
    small_avg_tfidf_dict = pickle.load(input_file)
```

In [55]:

```
from tqdm import tqdm
tfidf_lgr_train_score_list = []
tfidf_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 c_value in tqdm([0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]):
    tfidfclf = SVC(C=c_value, kernel='rbf')
    tfidfclf.fit(small_avg_tfidf_dict['X_train_avgw2v'],Y_train)
    train_proba = tfidfclf.decision_function(small_avg_tfidf_dict['X_train_avgw2v'])
    val_proba = tfidfclf.decision_function(small_avg_tfidf_dict['X_val_avgw2v'])

    fpr[c_value], tpr[c_value], _ = roc_curve(Y_train, train_proba)
    roc_auc_train[c_value] = auc(fpr[c_value], tpr[c_value])
    tfidf_lgr_train_score_list.append(auc(fpr[c_value], tpr[c_value]))
    fpr_val[c_value], tpr_val[c_value], _ = roc_curve(Y_val, val_proba)
    roc_auc_val[c_value] = auc(fpr_val[c_value], tpr_val[c_value])
    tfidf_lgr_val_score_list.append(auc(fpr_val[c_value], tpr_val[c_value]))
print(roc_auc_train)
print(roc_auc_val)
```

```
100%|██████████| 9/9 [14:19<00:00, 205.54s/it]

{0.0001: 0.8143906546095184, 0.001: 0.852266004675349, 0.01: 0.8754321814320
594, 0.1: 0.8915572486817942, 1: 0.89222723557918, 10: 0.8945084023414589, 1
00: 0.91105437335793, 1000: 0.93867206242665, 10000: 0.9701042677112136}
{0.0001: 0.8123203232792684, 0.001: 0.845005127934291, 0.01: 0.8622836064154
128, 0.1: 0.8752647944108041, 1: 0.8756251672258012, 10: 0.8758985534992472,
100: 0.872082018988696, 1000: 0.8607469736256029, 10000: 0.8369375145382083}
```

In [56]:

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

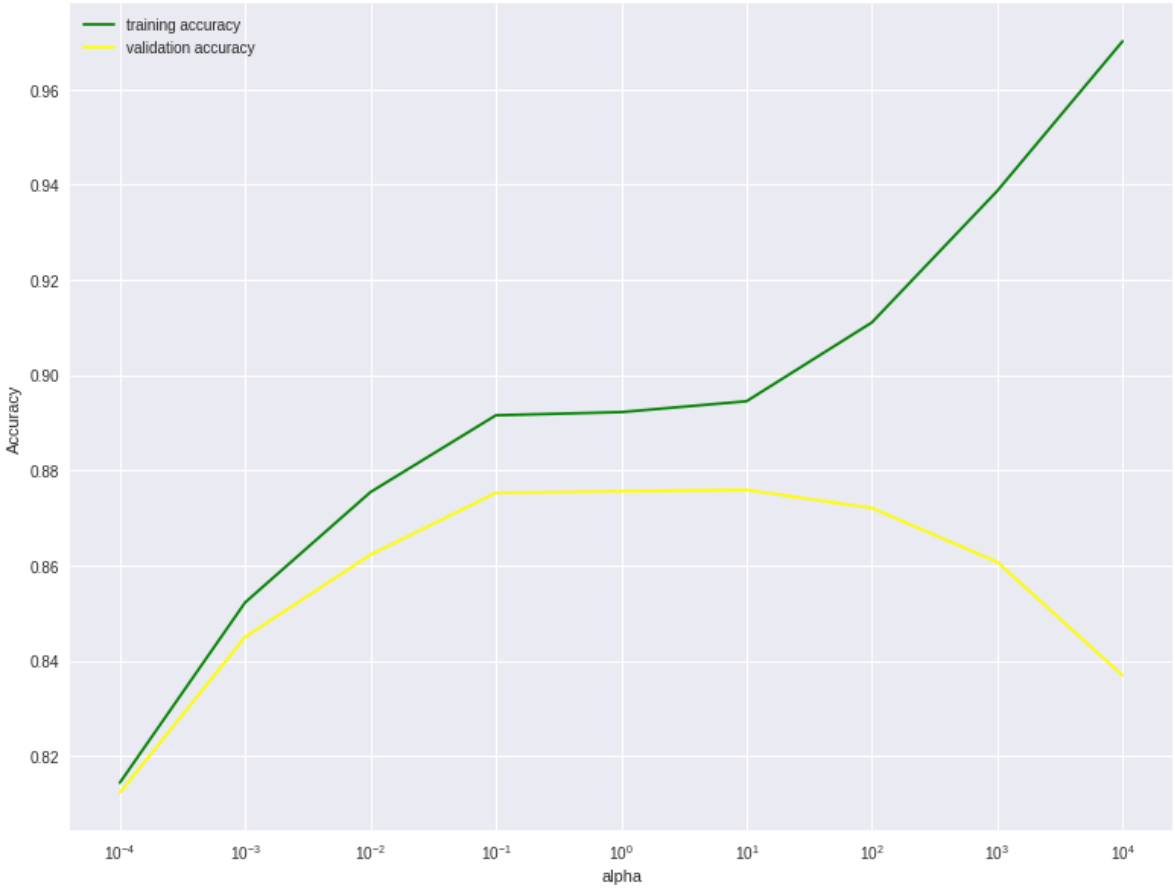
Out[56]:

```
10
```

In [57]:

```
import pylab
plt.figure(figsize=(13, 10))
neighbors_settings = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
plt.plot(neighbors_settings, tfidf_lgr_train_score_list, label="training accuracy", color='
plt.plot(neighbors_settings, tfidf_lgr_val_score_list, label="validation accuracy", color='
# plt.plot(neighbors_settings, auc_test, label="test accuracy", color='red')
plt.xlabel('alpha')
plt.ylabel('Accuracy')
plt.legend()
plt.xscale('log')

plt.show()
```



In [58]:

```
avg_svm_linear=SVC(C=best_c, kernel='rbf')
avg_svm_linear.fit(small_avg_tfidf_dict['X_train_avgw2v'], Y_train)
f = CalibratedClassifierCV(base_estimator=bow_svm_linear)
f.fit(small_avg_tfidf_dict['X_train_avgw2v'], Y_train)
avg_linear_test_proba = f.predict_proba(small_avg_tfidf_dict['X_test_avgw2v'])
avg_linear_train_proba = f.predict_proba(small_avg_tfidf_dict['X_train_avgw2v'])
avg_linear_test_proba
```

Out[58]:

```
array([[0.08696042, 0.91303958],
       [0.19738614, 0.80261386],
       [0.31957795, 0.68042205],
       ...,
       [0.06537385, 0.93462615],
       [0.05504791, 0.94495209],
       [0.09057401, 0.90942599]])
```

In [59]:

```
avg_fpr_train, avg_tpr_train, _ = roc_curve(Y_train, avg_linear_train_proba[:, 1])
avg_fpr_test, avg_tpr_test, _ = roc_curve(Y_test, avg_linear_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)
```

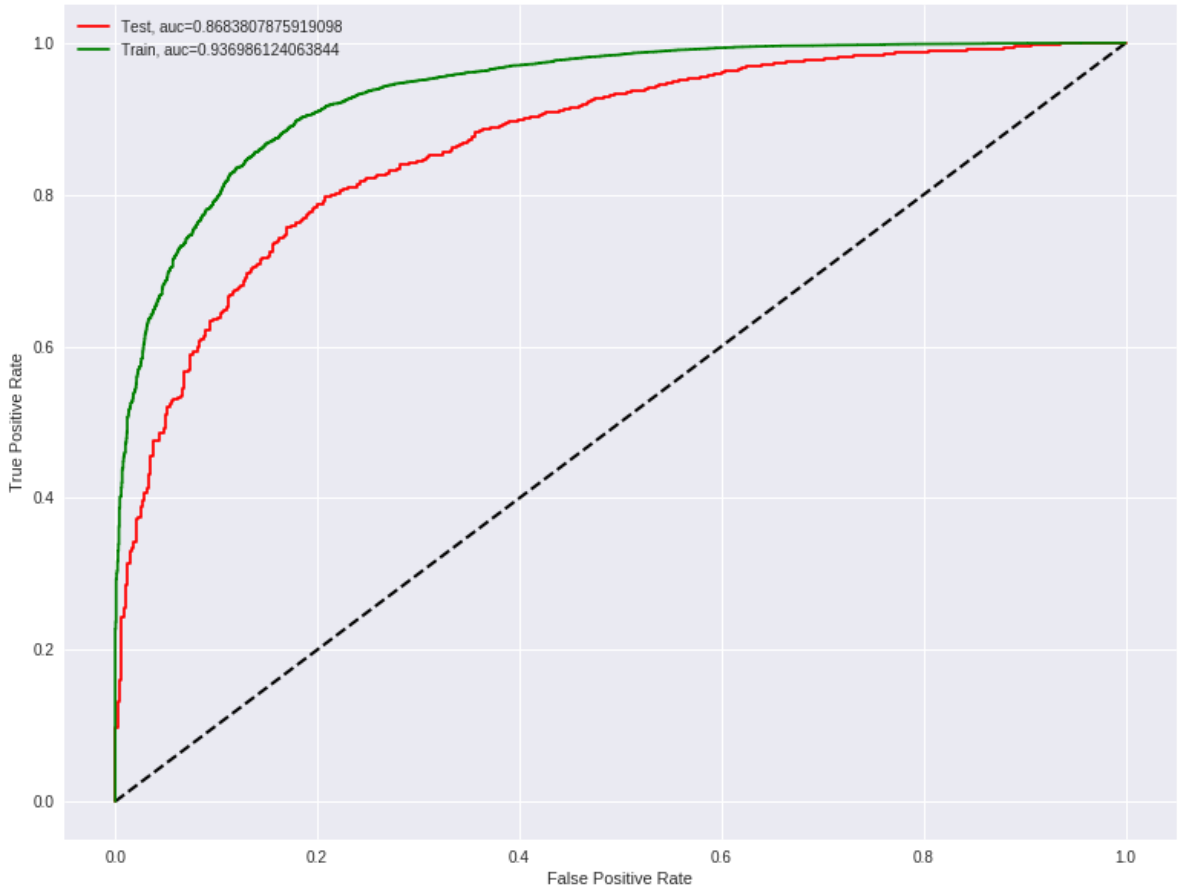
```
0.8683807875919098
0.936986124063844
```

In [63]:

```
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 [0]:

```
avg_test_conf = avg_svm_linear.predict(small_avg_tfidf_dict['X_test_avgw2v'])
avg_train_conf = avg_svm_linear.predict(small_avg_tfidf_dict['X_train_avgw2v'])
```

In [65]:

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

```
[[   7  475]
 [   0 4518]]
              precision    recall  f1-score   support

           0       1.00      0.01      0.03       482
           1       0.90      1.00      0.95      4518

   micro avg       0.91      0.91      0.91      5000
   macro avg       0.95      0.51      0.49      5000
weighted avg       0.91      0.91      0.86      5000
```

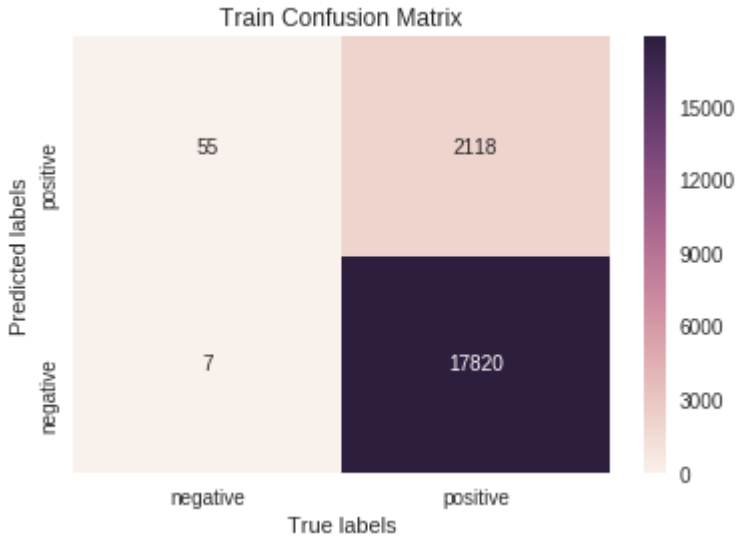
In [66]:

```
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[66]:

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



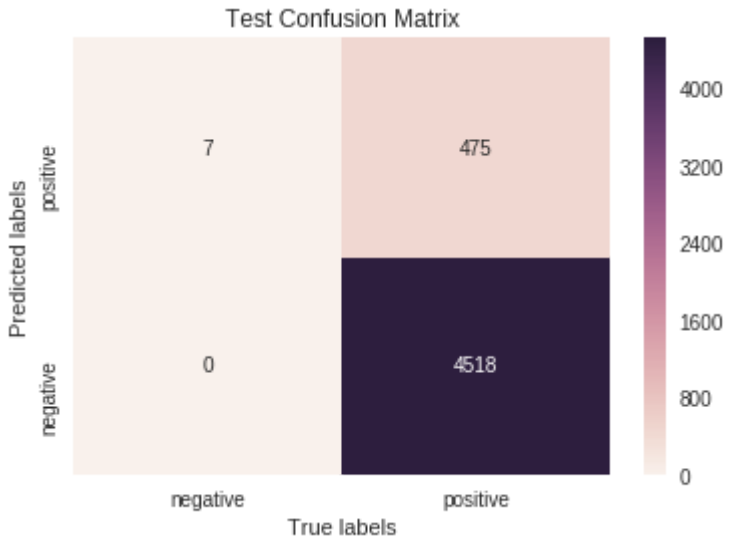
In [67]:

```
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[67]:

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



SVM on tfidf_w2v (RBF)

In [68]:

```
import pickle
# with open(r"/content/gdrive/My Drive/Colab Notebooks/SVM/30ktfidf_w2v.pkl", "rb") as inpu
with open(r"30ktfidf_w2v.pkl", "rb") as input_file:
    tfidfw2v_dict = pickle.load(input_file)
```

In [69]:

```
from tqdm import tqdm
tfidf_lgr_train_score_list = []
tfidf_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 c_value in tqdm([0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]):
    tfidfclf = SVC(C=c_value, kernel='rbf', class_weight='auto')
    tfidfclf.fit(tfidfw2v_dict['X_train_tfidfw2v'],Y_train)
    train_proba = tfidfclf.decision_function(tfidfw2v_dict['X_train_tfidfw2v'])
    val_proba = tfidfclf.decision_function(tfidfw2v_dict['X_val_tfidfw2v'])

    fpr[c_value], tpr[c_value], _ = roc_curve(Y_train, train_proba)
    roc_auc_train[c_value] = auc(fpr[c_value], tpr[c_value])
    tfidf_lgr_train_score_list.append(auc(fpr[c_value], tpr[c_value]))
    fpr_val[c_value], tpr_val[c_value], _ = roc_curve(Y_val, val_proba)
    roc_auc_val[c_value] = auc(fpr_val[c_value], tpr_val[c_value])
    tfidf_lgr_val_score_list.append(auc(fpr_val[c_value], tpr_val[c_value]))
print(roc_auc_train)
print(roc_auc_val)
```

```
  0%|
| 0/9 [00:00<?, ?it/s]C:\Program Files\Anaconda3\lib\site-packages\sklearn\u
tils\class_weight.py:62: DeprecationWarning: The class_weight='auto' heurist
ic is deprecated in 0.17 in favor of a new heuristic class_weight='balance
d'. 'auto' will be removed in 0.19
  " 0.19", DeprecationWarning)
 11%|
| 1/9 [01:19<10:35, 79.44s/it]C:\Program Files\Anaconda3\lib\site-packages\s
klearn\utils\class_weight.py:62: DeprecationWarning: The class_weight='auto'
heuristic is deprecated in 0.17 in favor of a new heuristic class_weight='ba
lanced'. 'auto' will be removed in 0.19
  " 0.19", DeprecationWarning)
 22%|
| 2/9 [02:38<09:16, 79.44s/it]C:\Program Files\Anaconda3\lib\site-packages\s
klearn\utils\class_weight.py:62: DeprecationWarning: The class_weight='auto'
heuristic is deprecated in 0.17 in favor of a new heuristic class_weight='ba
lanced'. 'auto' will be removed in 0.19
  " 0.19", DeprecationWarning)
 33%|
| 3/9 [03:47<07:34, 75.74s/it]C:\Program Files\Anaconda3\lib\site-packages\s
klearn\utils\class_weight.py:62: DeprecationWarning: The class_weight='auto'
heuristic is deprecated in 0.17 in favor of a new heuristic class_weight='ba
lanced'. 'auto' will be removed in 0.19
  " 0.19", DeprecationWarning)
 44%|
| 4/9 [04:47<05:58, 71.79s/it]C:\Program Files\Anaconda3\lib\site-packages\s
klearn\utils\class_weight.py:62: DeprecationWarning: The class_weight='auto'
heuristic is deprecated in 0.17 in favor of a new heuristic class_weight='ba
lanced'. 'auto' will be removed in 0.19
  " 0.19", DeprecationWarning)
 56%|
| 5/9 [0
5:38<04:30, 67.64s/it]C:\Program Files\Anaconda3\lib\site-packages\sklearn\u
tils\class_weight.py:62: DeprecationWarning: The class_weight='auto' heurist
```

```
ic is deprecated in 0.17 in favor of a new heuristic class_weight='balance
d'. 'auto' will be removed in 0.19
  " 0.19", DeprecationWarning)
 67%|████████████████████████████████████████████████████████████
████████████████                                                  | 6/9 [0
6:30<03:15, 65.05s/it]C:\Program Files\Anaconda3\lib\site-packages\sklearn\u
tils\class_weight.py:62: DeprecationWarning: The class_weight='auto' heurist
ic is deprecated in 0.17 in favor of a new heuristic class_weight='balance
d'. 'auto' will be removed in 0.19
  " 0.19", DeprecationWarning)
 78%|████████████████████████████████████████████████████████████
████████████████████████████████                                  | 7/9 [0
7:30<02:08, 64.42s/it]C:\Program Files\Anaconda3\lib\site-packages\sklearn\u
tils\class_weight.py:62: DeprecationWarning: The class_weight='auto' heurist
ic is deprecated in 0.17 in favor of a new heuristic class_weight='balance
d'. 'auto' will be removed in 0.19
  " 0.19", DeprecationWarning)
 89%|████████████████████████████████████████████████████████████
████████████████████████████████████████                          | 8/9 [1
0:10<01:16, 76.29s/it]C:\Program Files\Anaconda3\lib\site-packages\sklearn\u
tils\class_weight.py:62: DeprecationWarning: The class_weight='auto' heurist
ic is deprecated in 0.17 in favor of a new heuristic class_weight='balance
d'. 'auto' will be removed in 0.19
  " 0.19", DeprecationWarning)
100%|████████████████████████████████████████████████████████████
████████████████████████████████████████████████████| 9/9 [1
6:42<00:00, 111.38s/it]

{0.1: 0.80842747693864259, 1: 0.83824824421432864, 100: 0.92872706542357264,
10000: 0.99165730270874863, 1000: 0.97312155785970855, 0.0001: 0.75793908271
787724, 10: 0.87695636677417421, 0.01: 0.76659805801894476, 0.001: 0.7579391
0853227575}
{0.1: 0.80929404596539722, 1: 0.82844118175876702, 100: 0.81908779854538305,
10000: 0.74472090717642858, 1000: 0.78603601941819212, 0.0001: 0.76487168902
62206, 10: 0.8329648702521949, 0.01: 0.77305560808835039, 0.001: 0.764871689
02622071}
```

In [70]:

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

Out[70]:

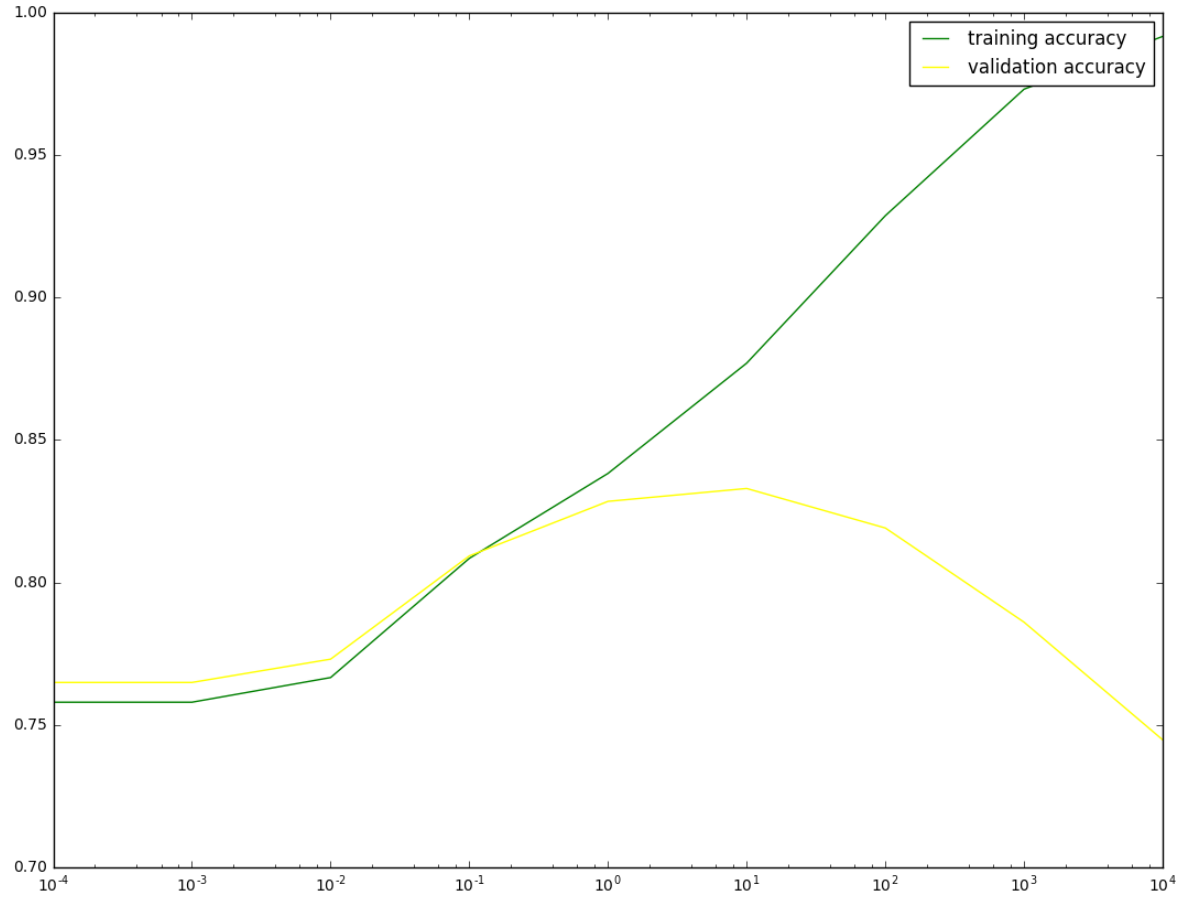
10

In [71]:

```
import pylab
plt.figure(figsize=(13, 10))
neighbors_settings = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
plt.plot(neighbors_settings, tfidf_lgr_train_score_list, label="training accuracy", color='
plt.plot(neighbors_settings, tfidf_lgr_val_score_list, label="validation accuracy", color='
# plt.plot(neighbors_settings, auc_test, label="test accuracy", color='red')

plt.legend()
plt.xscale('log')

plt.show()
```



In [72]:

```
tfidfw2v_svm_linear=SVC(C=best_c, kernel='rbf')
tfidfw2v_svm_linear.fit(tfidfw2v_dict['X_train_tfidfw2v'], Y_train)
f = CalibratedClassifierCV(base_estimator=tfidfw2v_svm_linear)
f.fit(tfidfw2v_dict['X_train_tfidfw2v'], Y_train)
tfidfw2v_test_proba = f.predict_proba(tfidfw2v_dict['X_test_tfidfw2v'])
tfidfw2v_train_proba = f.predict_proba(tfidfw2v_dict['X_train_tfidfw2v'])
tfidfw2v_test_proba
```

Out[72]:

```
array([[ 0.18847347,  0.81152653],
       [ 0.10093038,  0.89906962],
       [ 0.20384667,  0.79615333],
       ...,
       [ 0.07847939,  0.92152061],
       [ 0.05661477,  0.94338523],
       [ 0.11642975,  0.88357025]])
```

In [73]:

```
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)
```

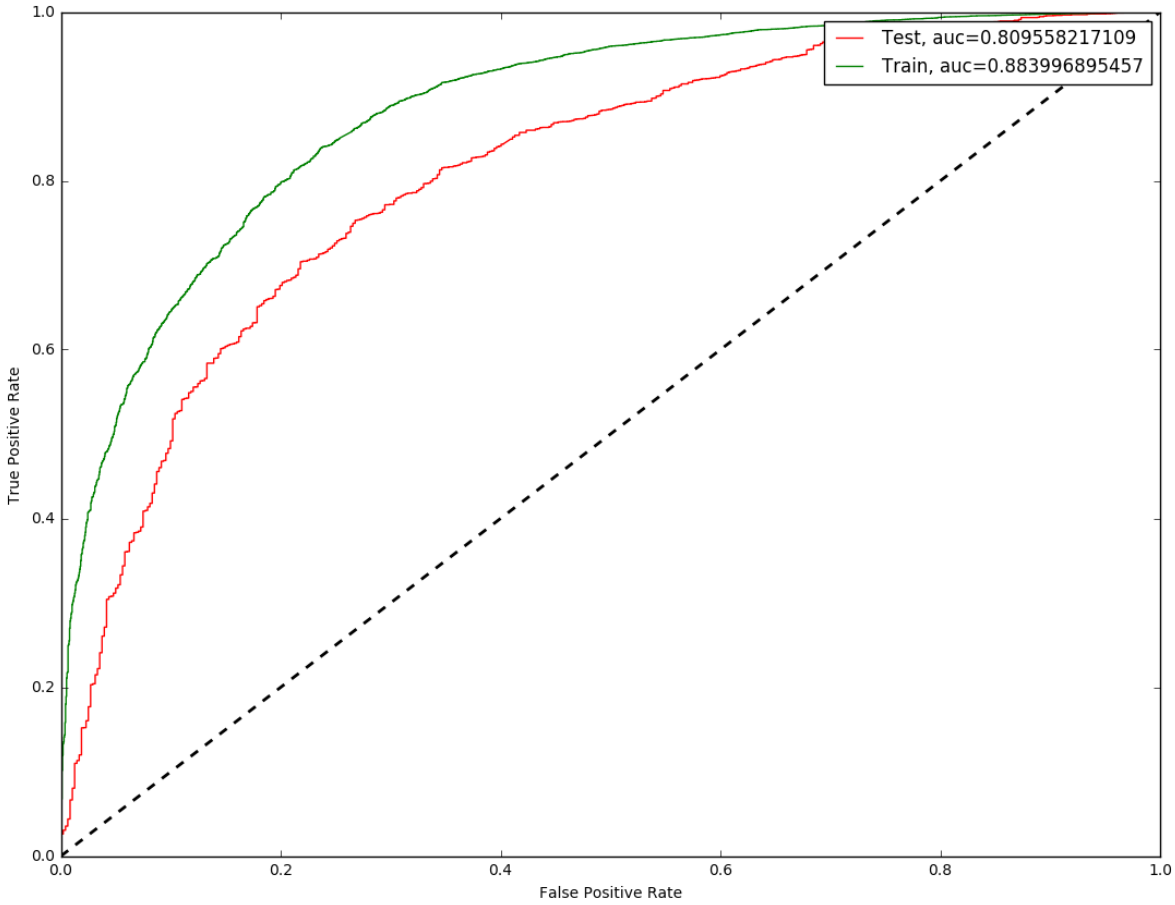
```
0.809558217109
0.883996895457
```

In [74]:

```
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 [75]:

```
tfidfw2v_test_conf = tfidfw2v_svm_linear.predict(tfidfw2v_dict['X_test_tfidfw2v'])
tfidfw2v_train_conf = tfidfw2v_svm_linear.predict(tfidfw2v_dict['X_train_tfidfw2v'])
```

In [76]:

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

```
[[   88  2085]
 [   15 17812]]
             precision    recall  f1-score   support

          0       0.77      0.02      0.04       482
          1       0.91      1.00      0.95      4518

avg / total       0.89      0.91      0.86      5000
```

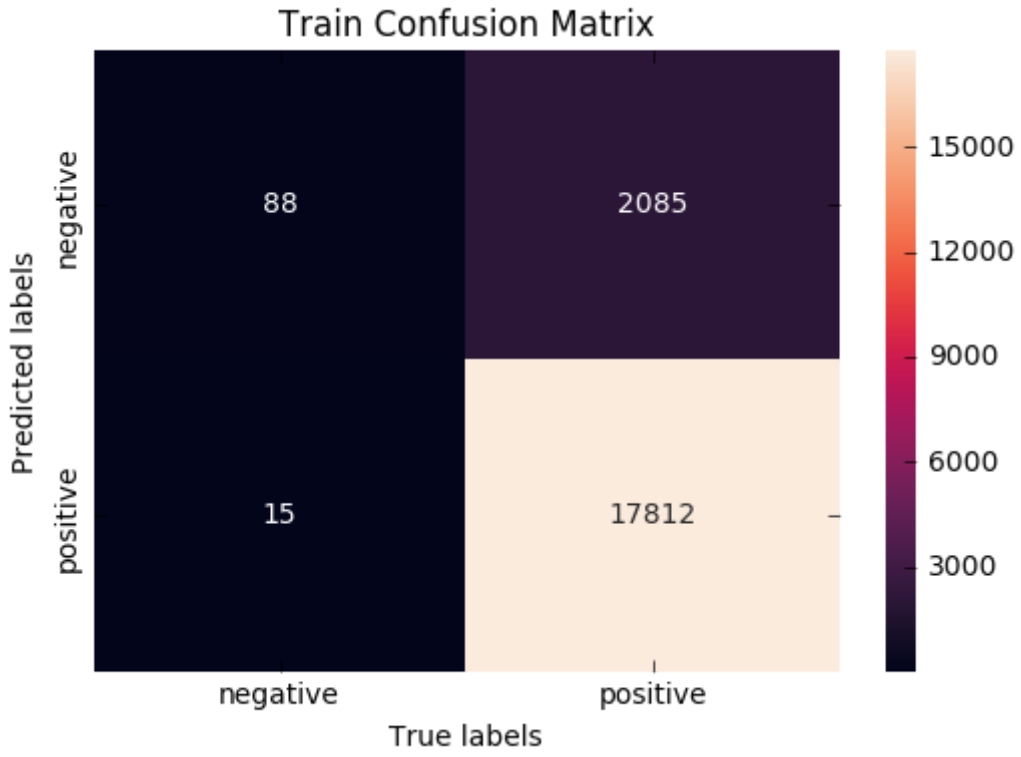
In [77]:

```
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[77]:

```
[<matplotlib.text.Text at 0x2502c1d0>, <matplotlib.text.Text at 0x19a78a90>]
```



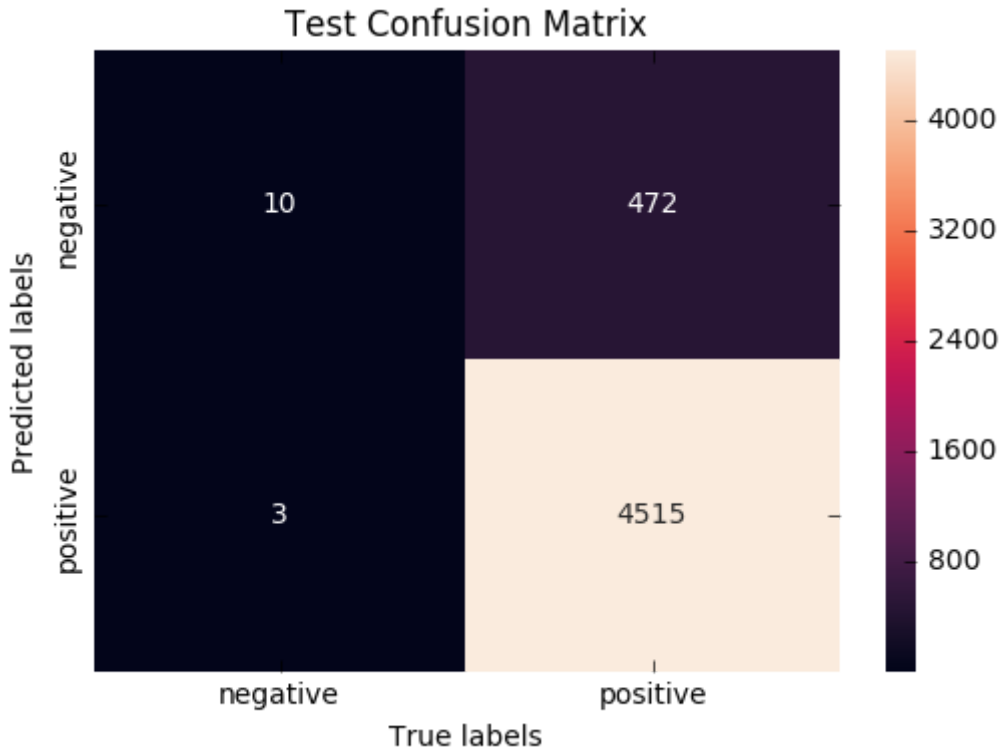
In [78]:

```
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[78]:

```
[<matplotlib.text.Text at 0x253e13c8>, <matplotlib.text.Text at 0x21367be0>]
```



In [321]:

```
from prettytable import PrettyTable

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

x.add_row(["Linear SVM",  "BoW", 0.845, 0.851])
x.add_row(["Linear SVM", "Tf-idf", 0.982, 0.971])
x.add_row(["Linear SVM", "Avg-w2v", 0.921, 0.924])
x.add_row(["Linear SVM", "tfidf_w2v", 0.834, 0.835])
x.add_row(["RBF",  "BoW", 0.992, 0.954])
x.add_row(["RBF", "Tf-idf", 0.976, 0.903])
x.add_row(["RBF", "Avg-w2v", 0.936, 0.868])
x.add_row(["RBF", "tfidf_w2v", 0.883, 0.809])
print(x)
```

```
+------------+------------+-------+-------+
| Algorithm  | Vectorizer | Train |  Test |
+------------+------------+-------+-------+
| Linear SVM |    BoW     | 0.845 | 0.851 |
| Linear SVM |   Tf-idf   | 0.982 | 0.971 |
| Linear SVM |  Avg-w2v   | 0.921 | 0.924 |
| Linear SVM | tfidf_w2v  | 0.834 | 0.835 |
|    RBF     |    BoW     | 0.992 | 0.954 |
|    RBF     |   Tf-idf   | 0.976 | 0.903 |
|    RBF     |  Avg-w2v   | 0.936 | 0.868 |
|    RBF     | tfidf_w2v  | 0.883 | 0.809 |
+------------+------------+-------+-------+
```

Steps taken to increase accuracy:
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.