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
```

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 [27]:

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

```
(100000, 13)
```

In [28]:

```
(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 [30]:

```
#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 [31]:

```
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 [32]:

```
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 [2]:

```
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 [3]:

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

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

In [4]:

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

In [5]:

```
filtered_data.shape
```

Out[5]:

```
(364171, 12)
```

In [6]:

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

In [7]:

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

Out[7]:

	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 [8]:

```
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 [9]:

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

Out[9]:

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

In [10]:

```
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 [11]:

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

Out[11]:

100000

In [12]:

```
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 [13]:

```
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 [78]:

```
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 [0]:

```
X_train_vect.shape
```

Out[39]:

(60000, 48270)

In [0]:

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

Out[40]:

364171

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 [101]:

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

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

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

In [176]:

```
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 [177]:

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

In [178]:

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

[4.4] Word2Vec

In [241]:

```
# 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 [242]:

```
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
```

```
[('fantastic', 0.9040200710296631), ('wonderful', 0.8998451232910156), ('goo
d', 0.8745156526565552), ('terrific', 0.8421463966369629), ('nice', 0.798710
2270126343), ('fabulous', 0.7933881282806396), ('perfect', 0.783984422683715
8), ('excellent', 0.7502694129943848), ('delicious', 0.7368237972259521),
('decent', 0.7319214344024658)]
==================================================
[('best', 0.8560127019882202), ('greatest', 0.8049135208129883), ('tasties
t', 0.7658340930938721), ('closest', 0.7615100145339966), ('BEST', 0.7247588
634490967), ('best-tasting', 0.7184524536132812), ('smoothest', 0.6995263695
716858), ('best.', 0.69404137134552), ('Best', 0.6918725967407227), ('fines
t', 0.678147554397583)]
```

In [243]:

```
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  32552
sample words  ['lime', 'brave', 'cheapest.', 'Bariani', 'digests', 'news,',
'shipping,', 'Nice,', 'notice', 'poppy', "Oreo's", 'sites', "ISN'T", 'crac
k', 'oatmeal.<br', 'defrost', 'discovers', 'Jerky,', '"pill', 'bears', 'scar
fed', 'make.', 'gravy', 'forced', 'robust.', 'NY,', 'Kellogs', 'try!', 'saut
eed', "Dave's", 'Le', 'chalk.', 'Biscotti.', 'unless', 'Geyser', 'PRIME', 'f
actor,', 'describe', 'applying', 'reasons.', 'the', 'Highland', 'layer', 'Us
e', 'Retrievers', 'considering.', '/>it', 'planet,', 'Aroma', 'objects']
```

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

[4.4.1.1] Avg W2v

In [244]:

```
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
20000
20000
```

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]))
```

```
20000
```

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%|████████████████████████████████████████████████████████████
████████████████████████████| 60000/60000 [1:26:58<00:00, 11.50it/s]

60000
50

100%|████████████████████████████████████████████████████████████
████████████████████████████| 20000/20000 [30:10<00:00, 11.05it/s]

20000
50

100%|████████████████████████████████████████████████████████████
████████████████████████████| 20000/20000 [26:46<00:00, 12.45it/s]

20000
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('avg_w2v.pkl', 'wb') as handle:
    pickle.dump(Avg_w2v_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

[4.4.1.2] TFIDF weighted W2v

In [245]:

```
# 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 [246]:

```
# 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 [ ]:

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

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

RandomForest on BoW

In [157]:

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

In [158]:

```
from scipy.sparse import vstack
X_train_val = vstack((BoW_dict['X_train_vect'], BoW_dict['X_val_vect']))
```

In [159]:

```
Y_train_val = pd.concat([Y_train, Y_val], axis= 0)
```

In [160]:

```
Y_train_val.shape
```

Out[160]:

```
(80000,)
```

In [164]:

```
import datetime
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
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()
fpr_list = []
tpr_list = []

param_grid = {
    'max_depth': [4, 8, 16, 32],
    'n_estimators': [1, 2, 5, 10, 50, 100, 200]
}

t1 = datetime.datetime.now()
rf = RandomForestClassifier(n_jobs=-1)
clf = GridSearchCV(estimator = rf, param_grid = param_grid, scoring = 'roc_auc')
clf.fit(X_train_val,Y_train_val)
print("time required = ", datetime.datetime.now() - t1)
```

```
time required =  0:02:33.990216
```

In [165]:

```
f = clf.cv_results_
f
```

Out[165]:

```
{'mean_fit_time': array([ 0.97740539,  0.70114668,  0.15065948,  0.1859403
5,  0.47265498,
         0.76851662,  1.34959571,  0.15388465,  0.16017962,  0.19231788,
         0.25494369,  0.73722482,  1.28683702,  2.4001766 ,  0.23009435,
         0.22432415,  0.28558477,  0.4133714 ,  1.38795837,  2.4608597 ,
         4.67881711,  0.36152228,  0.37205696,  0.46778607,  0.76128507,
         2.74038045,  5.19341008, 10.24588577]),
 'mean_score_time': array([0.02673141, 0.12127558, 0.12259523, 0.12348517,
0.12509839,
        0.12470937, 0.22550464, 0.02404054, 0.12200419, 0.12255851,
        0.12380783, 0.12453771, 0.12428466, 0.26012532, 0.02726666,
        0.12349645, 0.12457132, 0.12589081, 0.12573314, 0.22504497,
        0.32543484, 0.02674262, 0.12257338, 0.12438456, 0.12488071,
        0.12476722, 0.22517904, 0.32543405]),
 'mean_test_score': array([0.55766986, 0.52670725, 0.66591527, 0.70002797,
0.84617374,
        0.87517365, 0.89305158, 0.57119908, 0.62054525, 0.67092948,
        0.78276625, 0.87614757, 0.88975586, 0.90379871, 0.62458477,
```

In [36]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
```

In [166]:

```
x1_list = []
x2_list = []
for c1 in clf.cv_results_['params']:
    x1_list.append(c1['n_estimators'])
for c2 in clf.cv_results_['params']:
    x2_list.append(c2['max_depth'])
print(x1_list, x2_list)
```

```
[1, 2, 5, 10, 50, 100, 200, 1, 2, 5, 10, 50, 100, 200, 1, 2, 5, 10, 50, 100,
200, 1, 2, 5, 10, 50, 100, 200] [4, 4, 4, 4, 4, 4, 4, 8, 8, 8, 8, 8, 8, 8, 1
6, 16, 16, 16, 16, 16, 16, 32, 32, 32, 32, 32, 32, 32]
```

In [167]:

```
x1 = x1_list
y1 = x2_list
z1 = clf.cv_results_['mean_train_score'].tolist()
x2 = x1_list
y2 = x2_list
z2 = clf.cv_results_['mean_test_score'].tolist()
```

In [168]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='n_estimators'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

In [169]:

```
clf.best_params_
```

Out[169]:

```
{'max_depth': 32, 'n_estimators': 200}
```

In [170]:

```
best_max_depth = clf.best_params_['max_depth']
best_n_estimators = clf.best_params_['n_estimators']
```

In [171]:

```
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(max_depth = best_max_depth, n_estimators=best_n_estimators,
rf_clf.fit(BoW_dict['X_train_vect'],Y_train)
bow_test_proba = rf_clf.predict_proba(BoW_dict['X_test_vect'])
bow_train_proba = rf_clf.predict_proba(BoW_dict['X_train_vect'])
bow_test_proba
```

Out[171]:

```
array([[0.06852143, 0.93147857],
       [0.19292218, 0.80707782],
       [0.14024641, 0.85975359],
       ...,
       [0.10265641, 0.89734359],
       [0.06331641, 0.93668359],
       [0.22911392, 0.77088608]])
```

In [172]:

```
print("Top 20 Important Features")
d = sorted(list(zip(count_vect.get_feature_names(), rf_clf.feature_importances_ )), key=lam
features_list = []
for (i,j) in d:
    features_list.append(i)
```

```
Top 20 Important Features
```

In [173]:

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

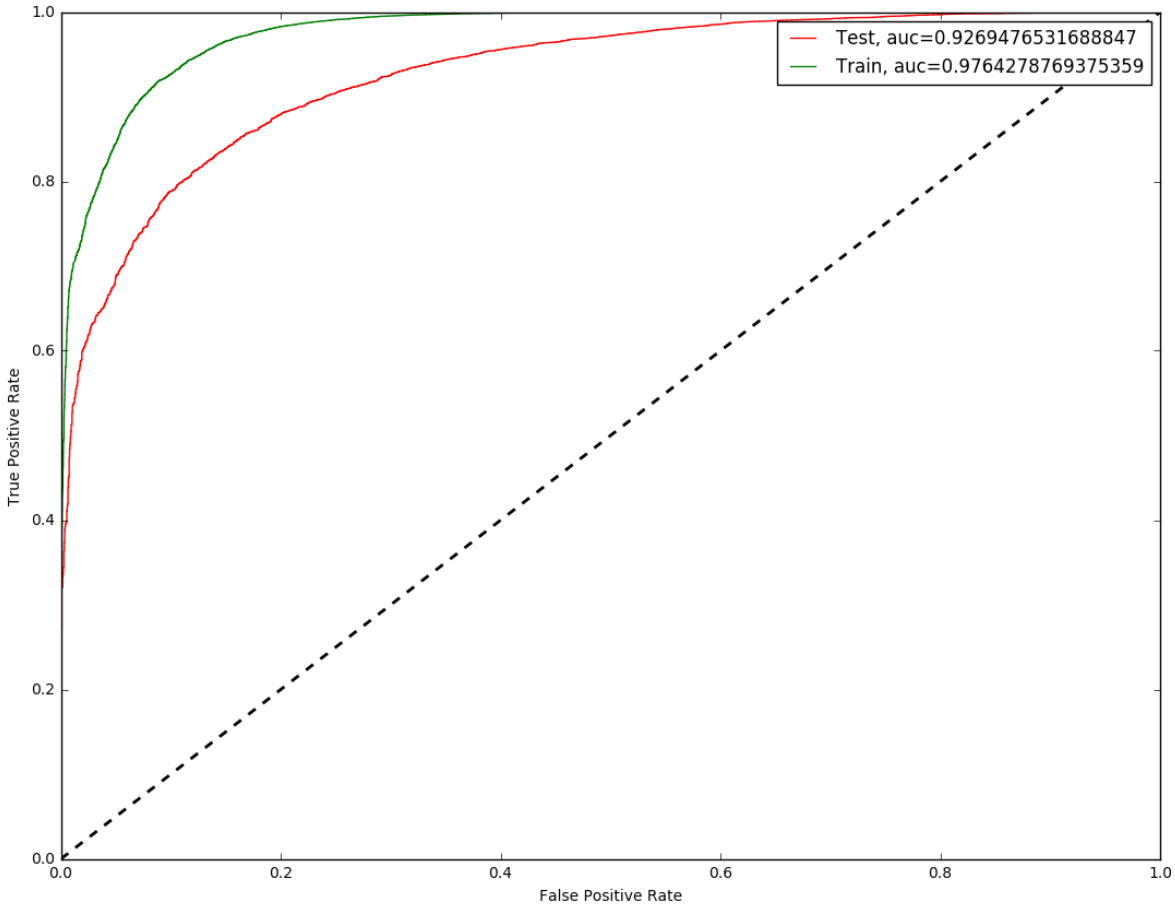
```
0.9269476531688847
0.9764278769375359
```

In [174]:

```
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 [182]:

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

In [183]:

```
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)
```

```
[[    5  2671]
 [    0 17324]]
              precision    recall  f1-score   support

           0       1.00      0.00      0.00      2676
           1       0.87      1.00      0.93     17324

   micro avg       0.87      0.87      0.87     20000
   macro avg       0.93      0.50      0.47     20000
weighted avg       0.88      0.87      0.80     20000
```

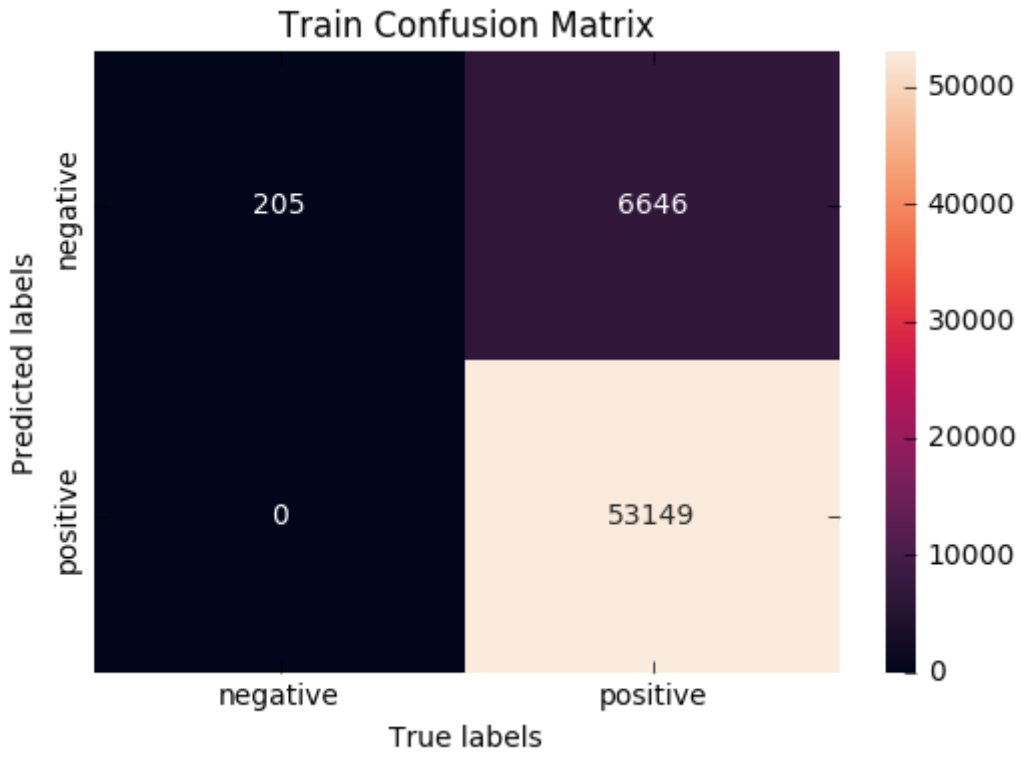
In [184]:

```
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[184]:

```
[<matplotlib.text.Text at 0x7f1e1a186d68>,
 <matplotlib.text.Text at 0x7f1e20629470>]
```



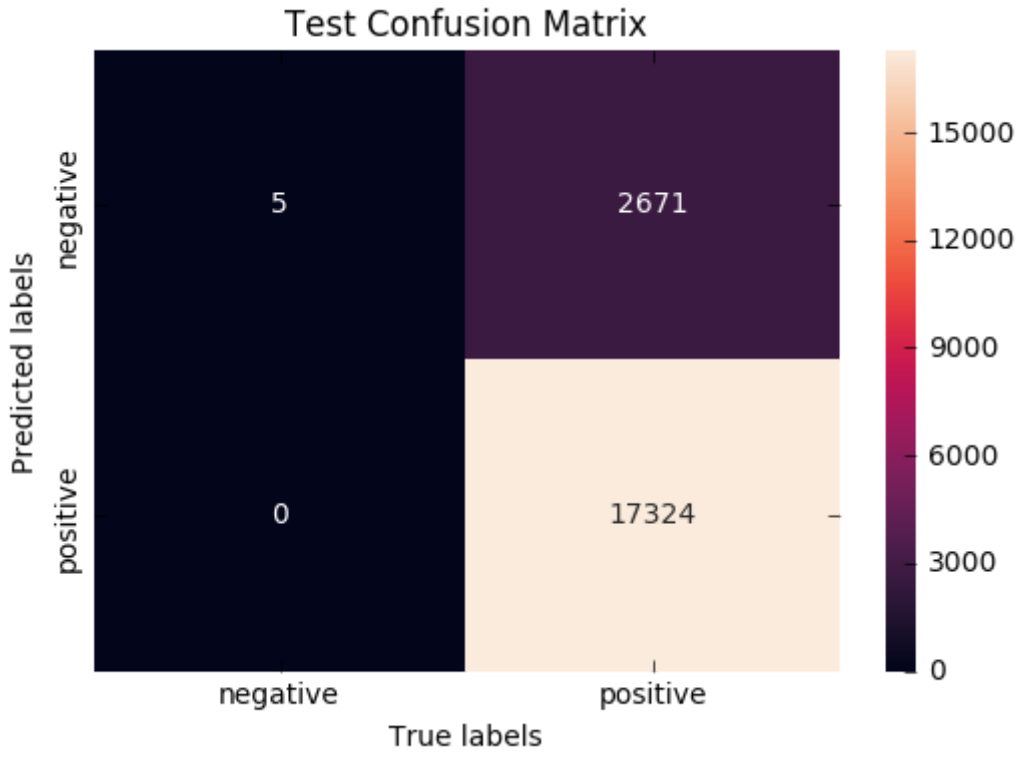
In [185]:

```
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[185]:

```
[<matplotlib.text.Text at 0x7f1da4097278>,
 <matplotlib.text.Text at 0x7f1e12f97748>]
```



In [186]:

```
complete_string = '-'.join(features_list)
```

In [187]:

```
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import pandas as pd

stopwords = set(STOPWORDS)

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(complete_string)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

Random Forest on Tf-IDF

In [82]:

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

In [83]:

```
from scipy.sparse import vstack
X_train_val_tfidf = vstack((tfidf_dict['train_tf_idf'], tfidf_dict['cv_tf_idf']))
```

In [84]:

```
print(X_train_val_tfidf.shape)
```

```
(80000, 35874)
```

In [188]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

param_grid = {
    'max_depth': [4, 8, 16, 32],
    'n_estimators': [1, 2, 5, 10, 50, 100, 200]
}

t1 = datetime.datetime.now()
rf = RandomForestClassifier(n_jobs=-1)
tfidf_clf = GridSearchCV(estimator = rf, param_grid = param_grid, scoring = 'roc_auc')
tfidf_clf.fit(X_train_val_tfidf,Y_train_val)
print("time required = ", datetime.datetime.now() - t1)
```

```
time required =  0:01:43.166788
```

In [189]:

```
tfidf_results = tfidf_clf.cv_results_
tfidf_results
```

Out[189]:

```
{'mean_fit_time': array([0.15014903, 0.14666653, 0.15765397, 0.17984613,
0.37819664,
        0.52489964, 0.87211355, 0.1490674 , 0.15321279, 0.16952984,
        0.20379901, 0.46182164, 0.7272435 , 1.21919449, 0.17298627,
        0.17706243, 0.20547533, 0.27113501, 0.72714432, 1.27915573,
        2.31394021, 0.2390151 , 0.25601085, 0.29394221, 0.42704558,
        1.42758338, 2.55353689, 4.88804245]),
 'mean_score_time': array([0.03033948, 0.12627753, 0.12750649, 0.12767768,
0.12905788,
        0.22980229, 0.32989438, 0.02937595, 0.12592673, 0.12781016,
        0.12754639, 0.12903547, 0.23011033, 0.3295571 , 0.02962724,
        0.12632132, 0.12713846, 0.12861347, 0.13052559, 0.23169589,
        0.33021156, 0.03209647, 0.12670676, 0.12808824, 0.12860147,
        0.12907942, 0.22996298, 0.3965342 ]),
 'mean_test_score': array([0.53568676, 0.5757967 , 0.66739042, 0.71552586,
0.87036892,
        0.89649206, 0.90561669, 0.55184043, 0.59121135, 0.70765464,
        0.7818555 , 0.90124691, 0.91056432, 0.92214252, 0.58578102,
```

In [190]:

```
tfidf_x_list = []
tfidf_y_list = []
for c1 in tfidf_clf.cv_results_['params']:
    tfidf_x_list.append(c1['n_estimators'])
for c2 in tfidf_clf.cv_results_['params']:
    tfidf_y_list.append(c2['max_depth'])
print(tfidf_x_list, tfidf_y_list)
```

```
[1, 2, 5, 10, 50, 100, 200, 1, 2, 5, 10, 50, 100, 200, 1, 2, 5, 10, 50, 100,
200, 1, 2, 5, 10, 50, 100, 200] [4, 4, 4, 4, 4, 4, 4, 8, 8, 8, 8, 8, 8, 8, 1
6, 16, 16, 16, 16, 16, 16, 32, 32, 32, 32, 32, 32, 32]
```

In [191]:

```
x1 = tfidf_x_list
y1 = tfidf_y_list
z1 = tfidf_clf.cv_results_['mean_train_score'].tolist()
x2 = tfidf_x_list
y2 = tfidf_y_list
z2 = tfidf_clf.cv_results_['mean_test_score'].tolist()
```

In [192]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='n_estimators'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

In [91]:

```
tfidf_clf.best_params_
```

Out[91]:

```
{'max_depth': 32, 'n_estimators': 200}
```

In [92]:

```
tfidf_best_max_depth = tfidf_clf.best_params_['max_depth']
tfidf_best_n_estimators = tfidf_clf.best_params_['n_estimators']
```

In [99]:

```
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(max_depth = tfidf_best_max_depth, n_estimators=tfidf_best_r
rf_clf.fit(tfidf_dict['train_tf_idf'],Y_train)
tfidf_test_proba = rf_clf.predict_proba(tfidf_dict['test_tf_idf'])
tfidf_train_proba = rf_clf.predict_proba(tfidf_dict['train_tf_idf'])
tfidf_test_proba
```

Out[99]:

```
array([[0.06929329, 0.93070671],
       [0.23496998, 0.76503002],
       [0.10822995, 0.89177005],
       ...,
       [0.10162364, 0.89837636],
       [0.06253476, 0.93746524],
       [0.27811993, 0.72188007]])
```

In [100]:

```
print("Top 20 Important Features")
d = sorted(list(zip(tf_idf_vect.get_feature_names(), rf_clf.feature_importances_ )), key=la
features_list_tfidf = []
for (i,j) in d:
    features_list_tfidf.append(i)
```

Top 20 Important Features

In [101]:

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

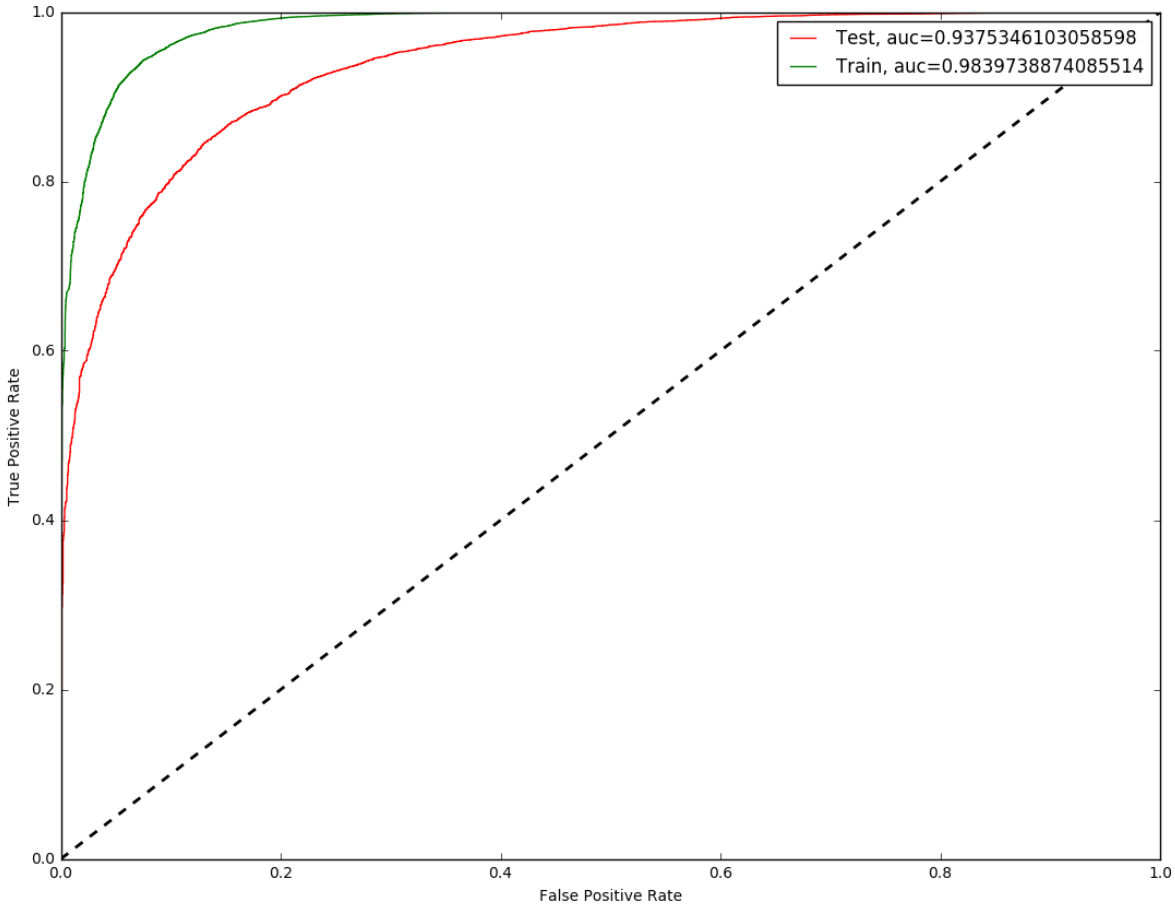
```
0.9375346103058598
0.9839738874085514
```

In [102]:

```
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 [103]:

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

In [104]:

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

```
[[   51  2625]
 [    0 17324]]
              precision    recall  f1-score   support

           0       1.00      0.02      0.04      2676
           1       0.87      1.00      0.93     17324

   micro avg       0.87      0.87      0.87     20000
   macro avg       0.93      0.51      0.48     20000
weighted avg       0.89      0.87      0.81     20000
```

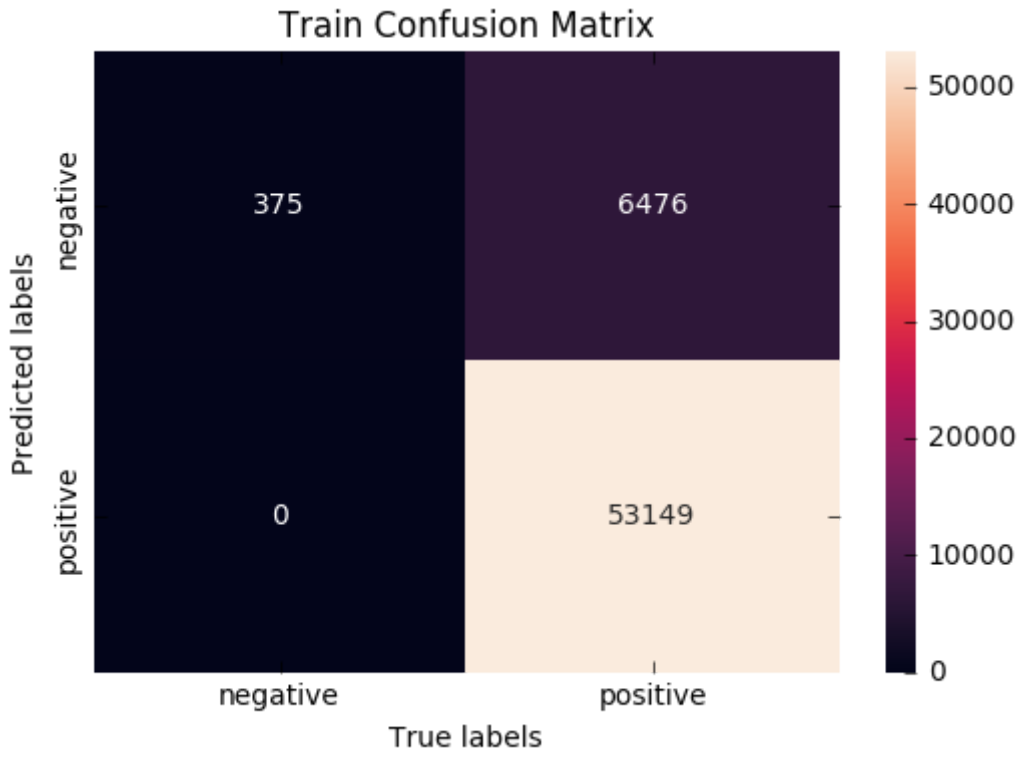
In [105]:

```
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[105]:

```
[<matplotlib.text.Text at 0x7f70e0c6f0f0>,
 <matplotlib.text.Text at 0x7f70dade2518>]
```



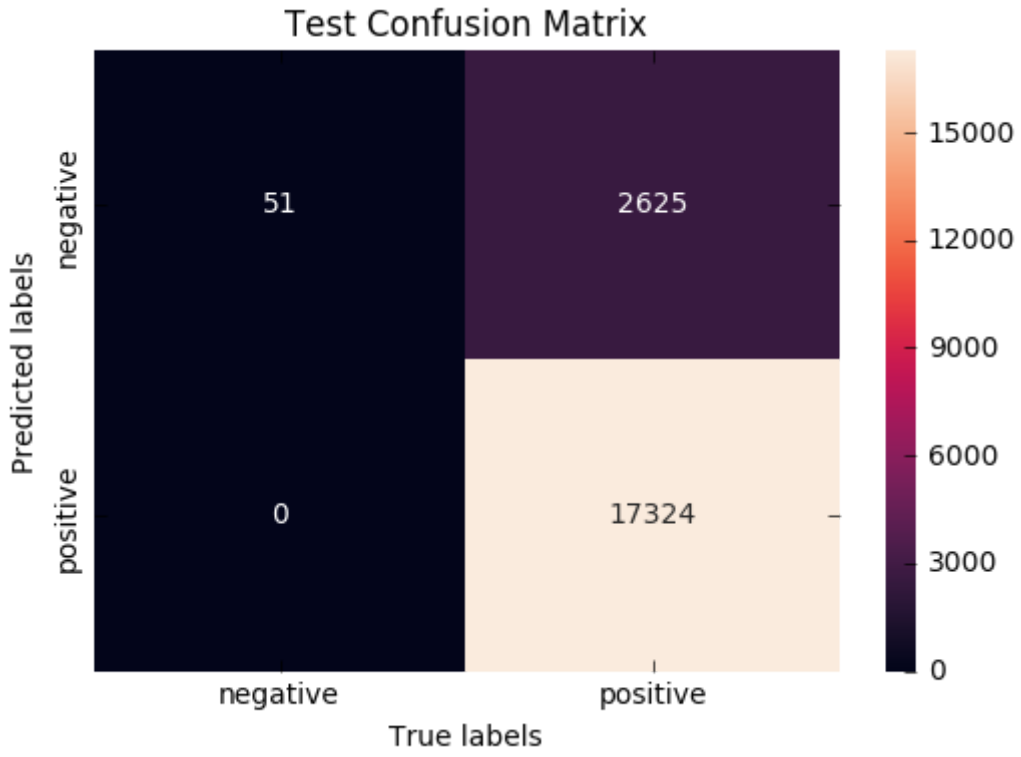
In [106]:

```
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[106]:

```
[<matplotlib.text.Text at 0x7f70dd7096a0>,
 <matplotlib.text.Text at 0x7f70e18bda58>]
```



In [107]:

```
complete_string_tfidf = '-'.join(features_list_tfidf)
```

In [108]:

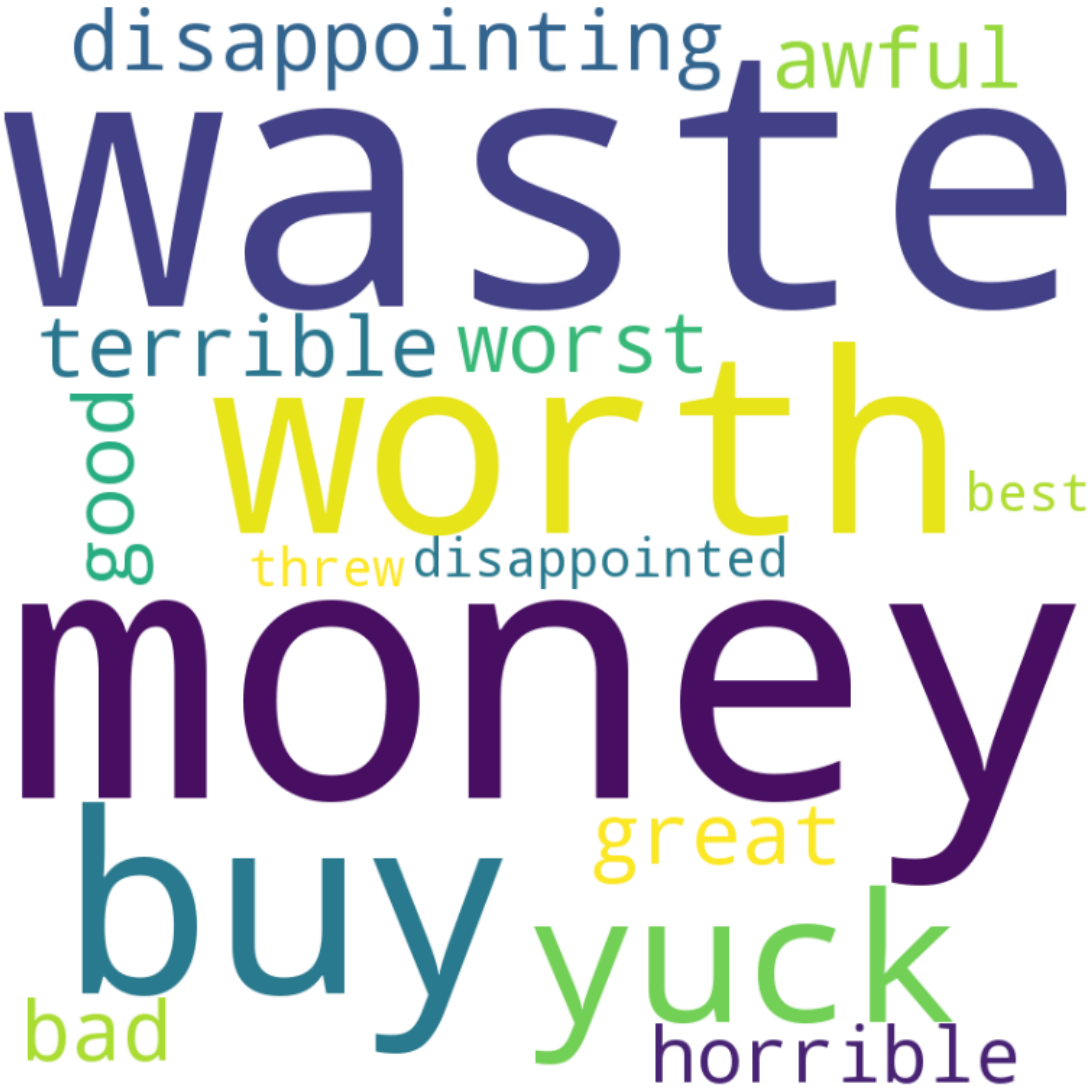
```
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import pandas as pd

stopwords = set(STOPWORDS)

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(complete_string_tfidf)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



Random Forest on Avg-w2v

In [15]:

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

In [16]:

```
from scipy.sparse import vstack
X_train_val_avg = vstack((avg_tfidf_dict['X_train_avgw2v'], avg_tfidf_dict['X_val_avgw2v']))
```

In [17]:

```
print(X_train_val_avg.shape)
```

```
(80000, 50)
```

In [22]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

param_grid = {
    'max_depth': [4, 8, 16, 32],
    'n_estimators': [1, 2, 5, 10, 50, 100, 200]
}

rf = RandomForestClassifier(n_jobs=-1)
avg_clf = GridSearchCV(estimator = rf, param_grid = param_grid, scoring = 'roc_auc')
avg_clf.fit(X_train_val_avg,Y_train_val)
```

Out[22]:

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, c
riterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=-1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'max_depth': [4, 8, 16, 32], 'n_estimators': [1, 2, 5, 1
0, 50, 100, 200]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [23]:

```
f = avg_clf.cv_results_
f
```

Out[23]:


```
{'mean_fit_time': array([ 1.16928315,  0.91250451,  0.39891299,  0.6126488
8,  2.04677916,
         3.83544485,  7.43842991,  0.70321751,  0.71991396,  0.88898826,
         1.49351199,  5.69642615, 10.50229581, 20.41955439,  1.60855373,
         1.58577553,  1.86641471,  3.47904801, 13.11870615, 24.82228676,
        49.3317349 ,  1.76349123,  1.74969506,  2.17763996,  3.93251562,
        14.73958937, 27.96743345, 55.25890509]),
 'mean_score_time': array([0.03179463, 0.12945366, 0.12915047, 0.13008507,
0.13046098,
        0.13027032, 0.23244619, 0.03100467, 0.12848783, 0.12977648,
        0.13042967, 0.13571517, 0.13130442, 0.23067991, 0.03164109,
        0.12858303, 0.12923145, 0.13035758, 0.13031061, 0.23154004,
        0.33419259, 0.03249153, 0.12689384, 0.12947321, 0.12842647,
        0.12885745, 0.22914831, 0.32987054]),
 'mean_test_score': array([0.73389127, 0.79834064, 0.84606253, 0.86275247,
0.87677814,
        0.88264378, 0.88279825, 0.78921409, 0.83692927, 0.87249977,
        0.88889682, 0.90334384, 0.90598936, 0.9064334 , 0.65288399,
```


In [24]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
```

In [25]:

```
x1_list_avg = []
x2_list_avg = []
for c1 in avg_clf.cv_results_['params']:
    x1_list_avg.append(c1['n_estimators'])
for c2 in avg_clf.cv_results_['params']:
    x2_list_avg.append(c2['max_depth'])
print(x1_list_avg, x2_list_avg)
```

```
[1, 2, 5, 10, 50, 100, 200, 1, 2, 5, 10, 50, 100, 200, 1, 2, 5, 10, 50, 100,
200, 1, 2, 5, 10, 50, 100, 200] [4, 4, 4, 4, 4, 4, 4, 8, 8, 8, 8, 8, 8, 8, 1
6, 16, 16, 16, 16, 16, 16, 32, 32, 32, 32, 32, 32, 32]
```

In [26]:

```
x1 = x1_list_avg
y1 = x2_list_avg
z1 = avg_clf.cv_results_['mean_train_score'].tolist()
x2 = x1_list_avg
y2 = x2_list_avg
z2 = avg_clf.cv_results_['mean_test_score'].tolist()
```

In [27]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='n_estimators'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

In [28]:

```
avg_clf.best_params_
```

Out[28]:

```
{'max_depth': 16, 'n_estimators': 200}
```

In [29]:

```
best_max_depth_avg = avg_clf.best_params_['max_depth']
best_n_estimators_avg = avg_clf.best_params_['n_estimators']
```

In [31]:

```
from sklearn.ensemble import RandomForestClassifier

avg_rf_clf = RandomForestClassifier(max_depth = best_max_depth_avg, n_estimators=best_n_est
avg_rf_clf.fit(avg_tfidf_dict['X_train_avgw2v'],Y_train)
avg_test_proba = avg_rf_clf.predict_proba(avg_tfidf_dict['X_test_avgw2v'])
avg_train_proba = avg_rf_clf.predict_proba(avg_tfidf_dict['X_train_avgw2v'])
avg_test_proba
```

Out[31]:

```
array([[0.06840416, 0.93159584],
       [0.61822774, 0.38177226],
       [0.02825124, 0.97174876],
       ...,
       [0.2422273 , 0.7577727 ],
       [0.00987844, 0.99012156],
       [0.49571719, 0.50428281]])
```

In [34]:

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

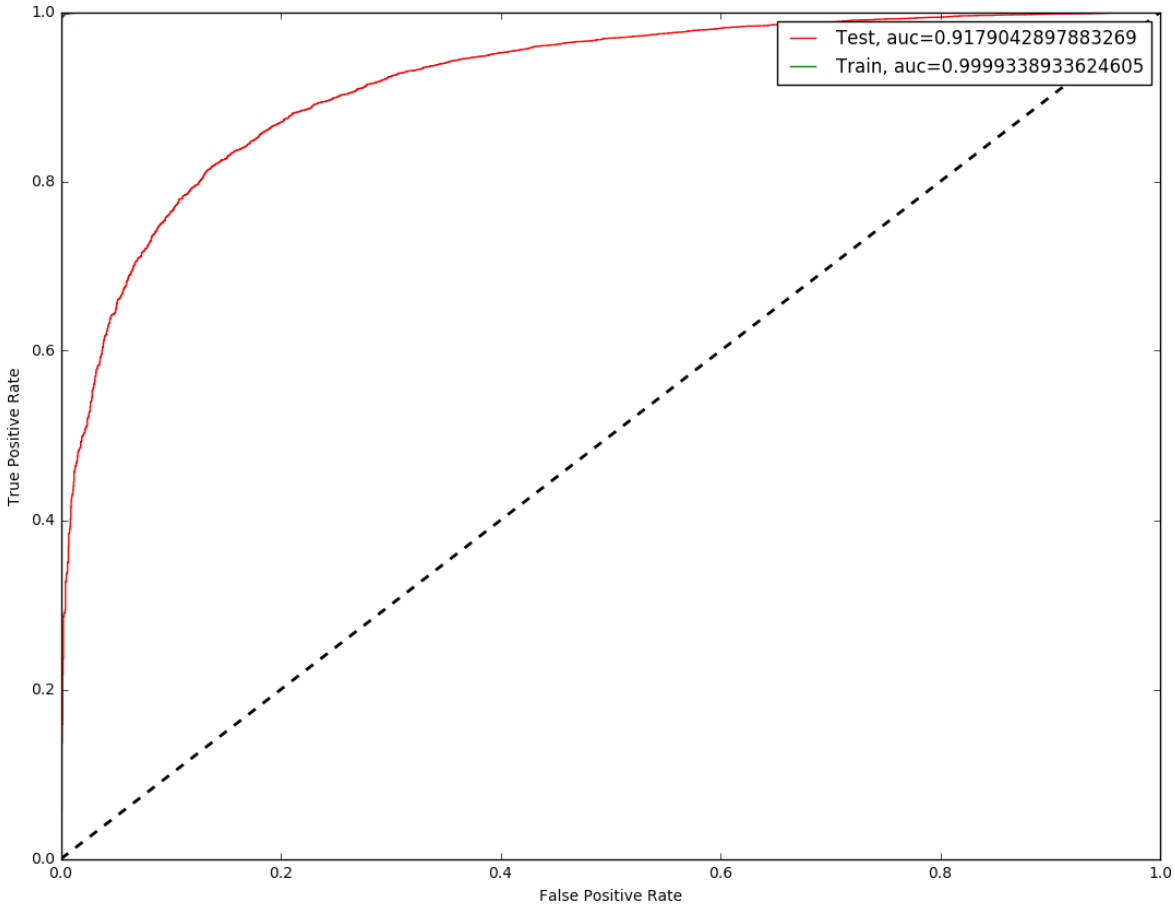
```
0.9179042897883269
0.9999338933624605
```

In [35]:

```
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 [36]:

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

In [37]:

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

```
[[  720  1956]
 [  163 17161]]
              precision    recall  f1-score   support

           0       0.82      0.27      0.40      2676
           1       0.90      0.99      0.94     17324

   micro avg       0.89      0.89      0.89     20000
   macro avg       0.86      0.63      0.67     20000
weighted avg       0.89      0.89      0.87     20000
```

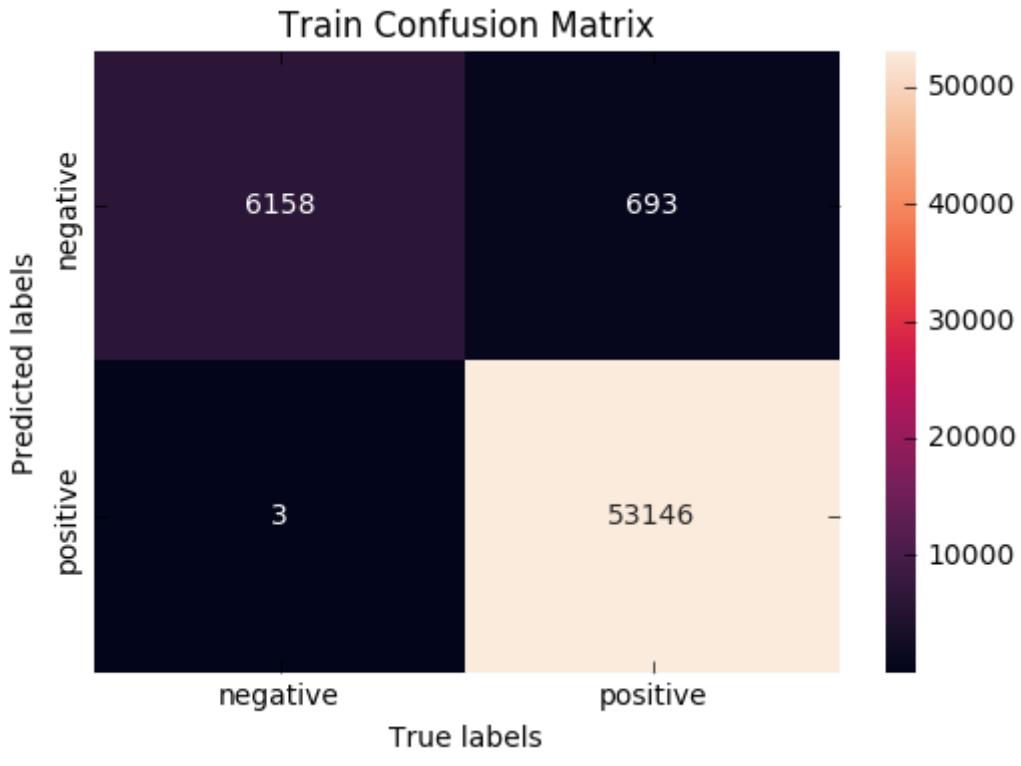
In [38]:

```
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[38]:

```
[<matplotlib.text.Text at 0x7f1e1b525550>,
 <matplotlib.text.Text at 0x7f1e208601d0>]
```



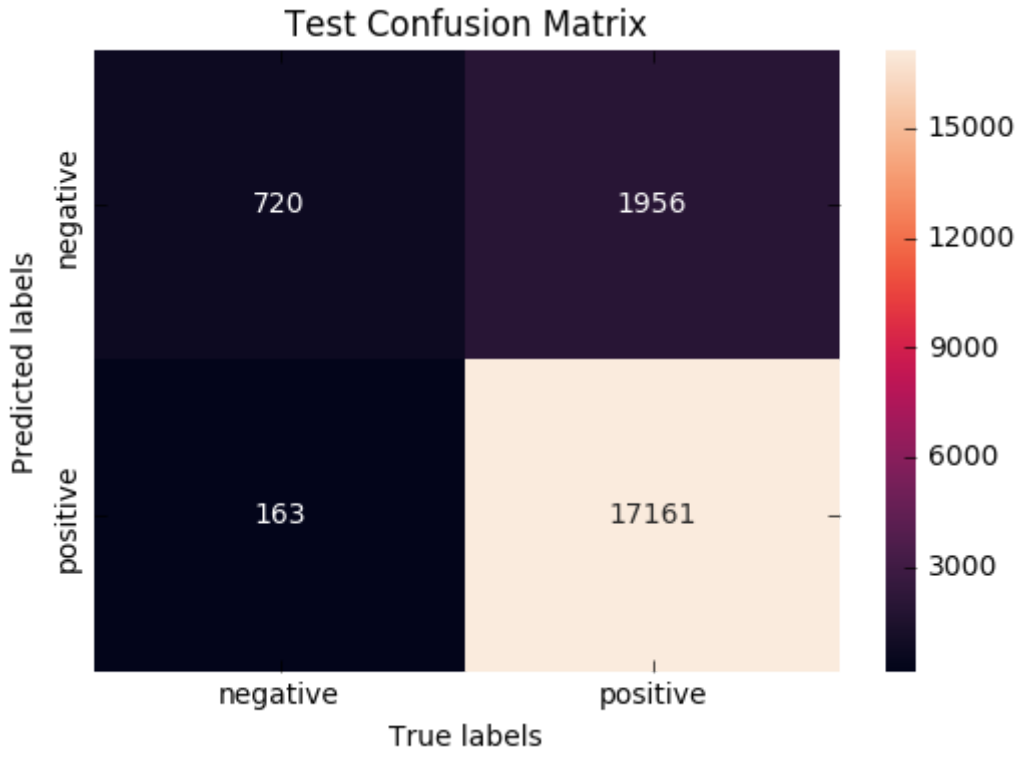
In [39]:

```
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[39]:

```
[<matplotlib.text.Text at 0x7f1e2103aeb8>,
 <matplotlib.text.Text at 0x7f1e1ef65a20>]
```



Random Forest on tfidf-w2v

In [40]:

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

In [41]:

```
from scipy.sparse import vstack
X_train_val_tfw2v = vstack((tfidfw2v_dict['X_train_tfidfw2v'], tfidfw2v_dict['X_val_tfidfw2
```

In [42]:

```
print(X_train_val_tfw2v.shape)
```

```
(80000, 50)
```

In [43]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

param_grid = {
    'max_depth': [4, 8, 16, 32],
    'n_estimators': [1, 2, 5, 10, 50, 100, 200]
}

rf = RandomForestClassifier(n_jobs=-1)
tfidf_clf = GridSearchCV(estimator = rf, param_grid = param_grid, scoring = 'roc_auc')
tfidf_clf.fit(X_train_val_tfw2v,Y_train_val)
```

Out[43]:

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, c
riterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=-1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'max_depth': [4, 8, 16, 32], 'n_estimators': [1, 2, 5, 1
0, 50, 100, 200]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [44]:

```
f = tfidf_clf.cv_results_
f
```

Out[44]:

```
{'mean_fit_time': array([ 1.1616679 ,  0.93776552,  0.4096148 ,  0.6329835
3,  2.04843752,
         3.97805985,  7.77070085,  0.69174655,  0.73983494,  0.89388847,
         1.54690981,  5.76354917, 10.68480412, 20.69904796,  1.65234963,
         1.64941963,  1.97124863,  3.63343072, 13.81511474, 26.19533491,
        51.89766041,  1.85127743,  1.86368354,  2.21636844,  4.16626652,
        15.79864685, 29.90068587, 58.90329838]),
 'mean_score_time': array([0.02964862, 0.12757119, 0.12915277, 0.1286575 ,
0.12746525,
        0.12792985, 0.22949855, 0.02849118, 0.12783281, 0.1274581 ,
        0.1279331 , 0.12905486, 0.16175167, 0.22854034, 0.02925014,
        0.12579576, 0.12721149, 0.13062342, 0.12871631, 0.23029256,
        0.32983907, 0.0301497 , 0.12442382, 0.12534722, 0.1260496 ,
        0.12615705, 0.22787007, 0.32823332]),
 'mean_test_score': array([0.72401387, 0.76415829, 0.80541541, 0.82669922,
0.83998243,
        0.84466451, 0.84464989, 0.76602945, 0.80028624, 0.83730107,
        0.85323878. 0.872535  . 0.87437433. 0.87609984. 0.62080704.
```

In [45]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
```

In [46]:

```
x1_list_tfw2v = []
x2_list_tfw2v = []
for c1 in tfidf_clf.cv_results_['params']:
    x1_list_tfw2v.append(c1['n_estimators'])
for c2 in tfidf_clf.cv_results_['params']:
    x2_list_tfw2v.append(c2['max_depth'])
print(x1_list_avg, x2_list_avg)
```

```
[1, 2, 5, 10, 50, 100, 200, 1, 2, 5, 10, 50, 100, 200, 1, 2, 5, 10, 50, 100,
200, 1, 2, 5, 10, 50, 100, 200] [4, 4, 4, 4, 4, 4, 4, 8, 8, 8, 8, 8, 8, 8, 1
6, 16, 16, 16, 16, 16, 16, 32, 32, 32, 32, 32, 32, 32]
```

In [47]:

```
x1 = x1_list_tfw2v
y1 = x2_list_tfw2v
z1 = tfidf_clf.cv_results_['mean_train_score'].tolist()
x2 = x1_list_tfw2v
y2 = x2_list_tfw2v
z2 = tfidf_clf.cv_results_['mean_test_score'].tolist()
```

In [48]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='n_estimators'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

In [50]:

```
tfidf_clf.best_params_
```

Out[50]:

```
{'max_depth': 32, 'n_estimators': 200}
```

In [51]:

```
best_max_depth_tfw2v = tfidf_clf.best_params_['max_depth']
best_n_estimators_tfw2v = tfidf_clf.best_params_['n_estimators']
```

In [52]:

```
from sklearn.ensemble import RandomForestClassifier

tfw2v_rf_clf = RandomForestClassifier(max_depth = best_max_depth_tfw2v, n_estimators=best_r
tfw2v_rf_clf.fit(tfidfw2v_dict['X_train_tfidfw2v'],Y_train)
tfw2v_test_proba = tfw2v_rf_clf.predict_proba(tfidfw2v_dict['X_test_tfidfw2v'])
tfw2v_train_proba = tfw2v_rf_clf.predict_proba(tfidfw2v_dict['X_train_tfidfw2v'])
tfw2v_test_proba
```

Out[52]:

```
array([[0.13 , 0.87 ],
       [0.515, 0.485],
       [0.01 , 0.99 ],
       ...,
       [0.195, 0.805],
       [0.015, 0.985],
       [0.5  , 0.5  ]])
```

In [53]:

```
tfw2v_fpr_train, tfw2v_tpr_train, _ = roc_curve(Y_train, tfw2v_train_proba[:, 1])
tfw2v_fpr_test, tfw2v_tpr_test, _ = roc_curve(Y_test, tfw2v_test_proba[:, 1])
tfw2v_test_auc = auc(tfw2v_fpr_test, tfw2v_tpr_test)
tfw2v_train_auc = auc(tfw2v_fpr_train, tfw2v_tpr_train)
print(tfw2v_test_auc)
print(tfw2v_train_auc)
```

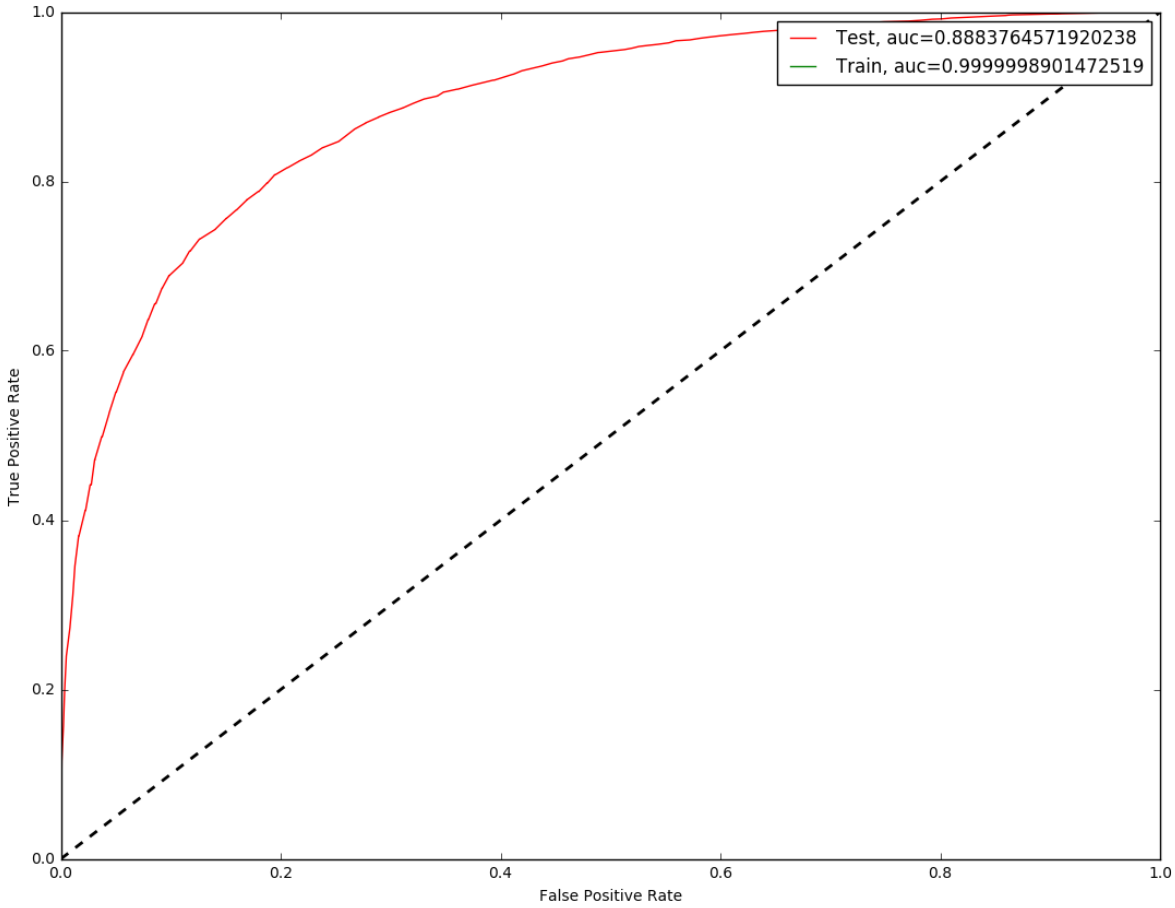
```
0.8883764571920238
0.9999998901472519
```

In [54]:

```
import pylab
plt.figure(figsize=(13, 10))
plt.plot([0,1], [0,1], color='black', lw=2, linestyle='--')
plt.plot(tfw2v_fpr_test, tfw2v_tpr_test, label="Test, auc="+str(tfw2v_test_auc), color = 'r
plt.plot(tfw2v_fpr_train, tfw2v_tpr_train, label="Train, auc="+str(tfw2v_train_auc), color

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

plt.show()
```



In [55]:

```
tfw2v_test_conf = tfw2v_rf_clf.predict(tfidfw2v_dict['X_test_tfidfw2v'])
tfw2v_train_conf = tfw2v_rf_clf.predict(tfidfw2v_dict['X_train_tfidfw2v'])
```

In [57]:

```
from sklearn.metrics import classification_report, confusion_matrix
tfw2v_train_conf_matrix = confusion_matrix(Y_train, tfw2v_train_conf)
tfw2v_test_conf_matrix = confusion_matrix(Y_test, tfw2v_test_conf)
class_report = classification_report(Y_test, tfw2v_test_conf)
print(tfw2v_test_conf_matrix)
print(class_report)
```

```
[[  597  2079]
 [  177 17147]]
              precision    recall  f1-score   support

           0       0.77      0.22      0.35      2676
           1       0.89      0.99      0.94     17324

   micro avg       0.89      0.89      0.89     20000
   macro avg       0.83      0.61      0.64     20000
weighted avg       0.88      0.89      0.86     20000
```

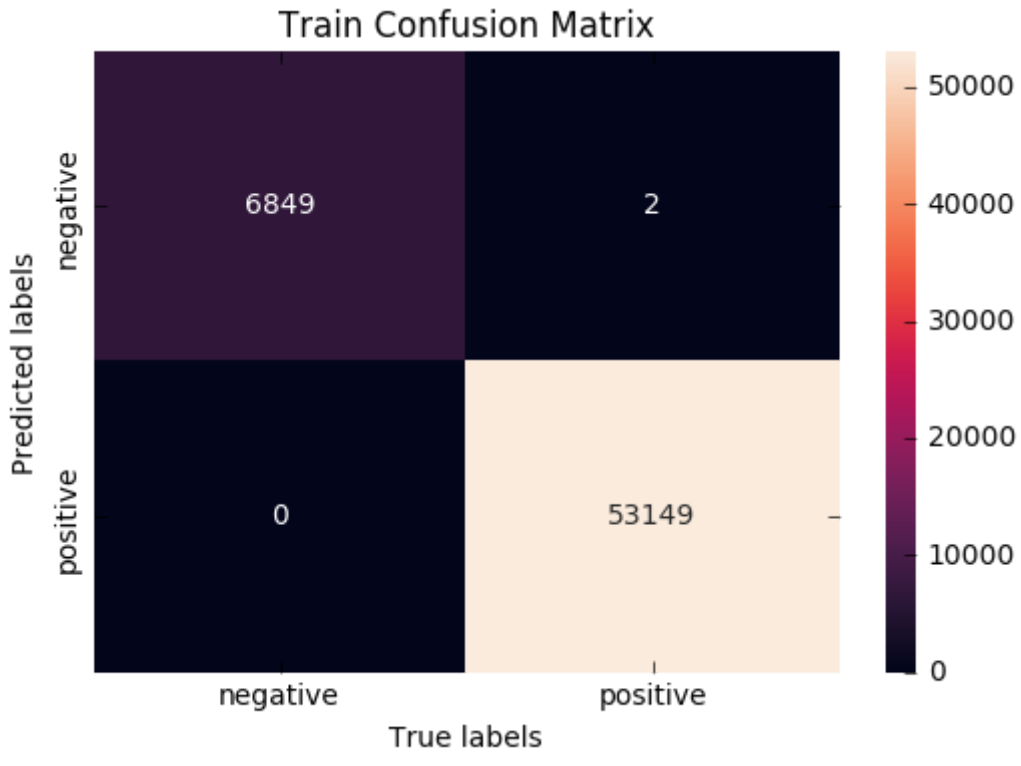
In [58]:

```
ax= plt.subplot()
sns.heatmap(tfw2v_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[58]:

```
[<matplotlib.text.Text at 0x7f1e1dcc0198>,
 <matplotlib.text.Text at 0x7f1e1e1c49b0>]
```



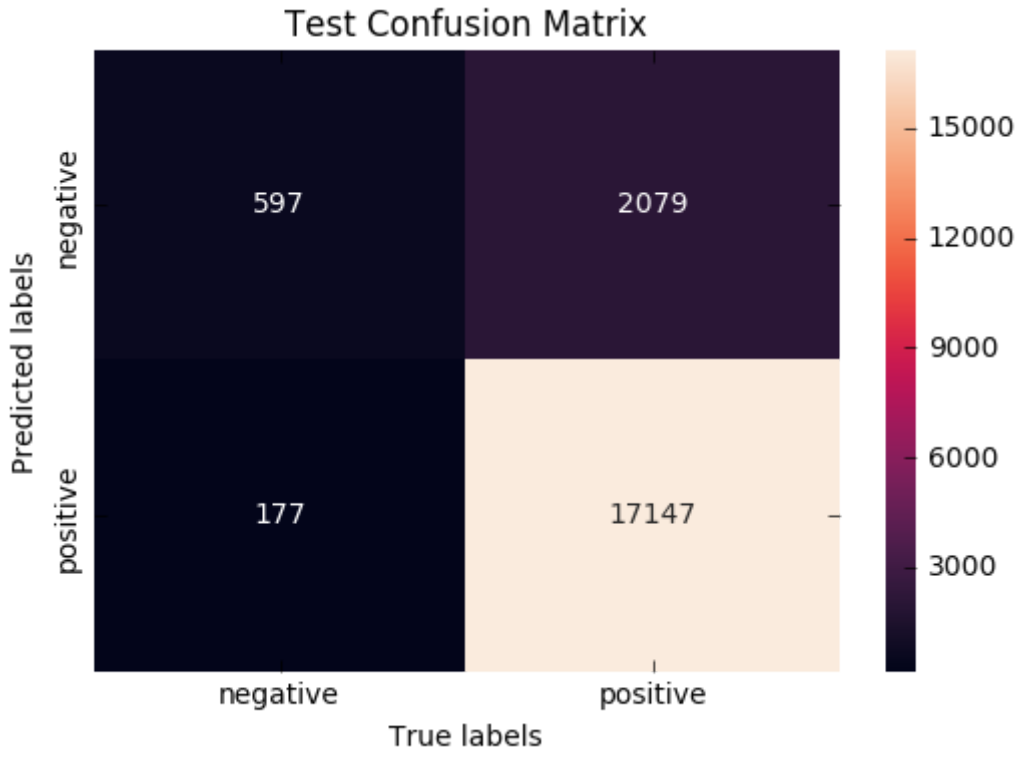
In [59]:

```
ax= plt.subplot()
sns.heatmap(tfw2v_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[59]:

```
[<matplotlib.text.Text at 0x7f1e1b798780>,
 <matplotlib.text.Text at 0x7f1e1b7af208>]
```



GBDT using XGBOOST on BoW

In [60]:

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

In [61]:

```
from scipy.sparse import vstack
X_train_val_bow = vstack((BoW_dict['X_train_vect'], BoW_dict['X_val_vect']))
```

In [64]:

```
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV

param_grid = {
    'max_depth': [4, 8, 16, 32],
    'n_estimators': [1, 2, 5, 10, 50, 100, 200]
}

bow_xgbclf = XGBClassifier(n_jobs=-1)
bow_gdclf = GridSearchCV(estimator = bow_xgbclf, param_grid = param_grid, scoring = 'roc_au
bow_gdclf.fit(X_train_val_bow,Y_train_val)
```

Out[64]:

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_b
ylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=-1, nthread=None, objective='binary:logistic',
       random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
       seed=None, silent=True, subsample=1),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'max_depth': [4, 8, 16, 32], 'n_estimators': [1, 2, 5, 1
0, 50, 100, 200]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [65]:

```
f = bow_gdclf.cv_results_
f
```

Out[65]:

```
{'mean_fit_time': array([  0.8598423 ,   0.91401863,   1.05132945,   1.478
72488,
         3.79973276,   6.77970767,  12.83034627,   0.89145271,
         1.02610048,   1.40350342,   2.02674556,   7.07837685,
        13.15599664,  25.00009664,   1.06002617,   1.33129867,
         2.16667914,   3.58025996,  14.41660198,  26.9937102 ,
        51.05000202,   1.38750434,   1.97296755,   3.7676487 ,
         6.85786978,  29.45196001,  54.67489036, 103.11327291]),
 'mean_score_time': array([0.37653995, 0.37894758, 0.35997605, 0.36318064,
0.36706916,
        0.37398052, 0.4101874 , 0.35804804, 0.36149089, 0.36099752,
        0.36288786, 0.3763926 , 0.39357376, 0.42622328, 0.36535732,
        0.36450577, 0.36424478, 0.36830743, 0.40275931, 0.43375341,
        0.51425926, 0.36532903, 0.36781128, 0.37069138, 0.37765511,
        0.44883696, 0.50853705, 0.72436023]),
 'mean_test_score': array([0.69320804, 0.73359178, 0.80183729, 0.81413214,
0.89186677,
        0.9172359 . 0.93509277. 0.77289603. 0.77815942. 0.829859  .
```

In [67]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
```

In [68]:

```
x1_list_xg_bow = []
x2_list_xg_bow = []
for c1 in bow_gdclf.cv_results_['params']:
    x1_list_xg_bow.append(c1['n_estimators'])
for c2 in bow_gdclf.cv_results_['params']:
    x2_list_xg_bow.append(c2['max_depth'])
print(x1_list_xg_bow, x2_list_xg_bow)
```

```
[1, 2, 5, 10, 50, 100, 200, 1, 2, 5, 10, 50, 100, 200, 1, 2, 5, 10, 50, 100,
200, 1, 2, 5, 10, 50, 100, 200] [4, 4, 4, 4, 4, 4, 4, 8, 8, 8, 8, 8, 8, 8, 1
6, 16, 16, 16, 16, 16, 16, 32, 32, 32, 32, 32, 32, 32]
```

In [72]:

```
x1 = x1_list_xg_bow
y1 = x2_list_xg_bow
z1 = bow_gdclf.cv_results_['mean_train_score'].tolist()
x2 = x1_list_xg_bow
y2 = x2_list_xg_bow
z2 = bow_gdclf.cv_results_['mean_test_score'].tolist()
```

In [73]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='n_estimators'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

In [74]:

```
bow_gdclf.best_params_
```

Out[74]:

```
{'max_depth': 32, 'n_estimators': 200}
```

In [75]:

```
best_max_depth_xg_bow = bow_gdclf.best_params_['max_depth']
best_n_estimators_xg_bow = bow_gdclf.best_params_['n_estimators']
```

In [76]:

```
from sklearn.ensemble import RandomForestClassifier

xgbow_clf = XGBClassifier(max_depth = best_max_depth_xg_bow, n_estimators=best_n_estimators
xgbow_clf.fit(BoW_dict['X_train_vect'],Y_train)
xgbow_test_proba = xgbow_clf.predict_proba(BoW_dict['X_test_vect'])
xgbow_train_proba = xgbow_clf.predict_proba(BoW_dict['X_train_vect'])
xgbow_test_proba
```

Out[76]:

```
array([[1.8140674e-03, 9.9818593e-01],
       [8.5452878e-01, 1.4547125e-01],
       [2.7268529e-03, 9.9727315e-01],
       ...,
       [1.0111630e-02, 9.8988837e-01],
       [6.0582161e-04, 9.9939418e-01],
       [5.6807244e-01, 4.3192753e-01]], dtype=float32)
```

In [79]:

```
print("Top 20 Important Features")
d = sorted(list(zip(count_vect.get_feature_names(), xgbow_clf.feature_importances_ )), key=
xgbow_features_list = []
for (i,j) in d:
    xgbow_features_list.append(i)
```

```
Top 20 Important Features
```

In [80]:

```
xgbow_fpr_train, xgbow_tpr_train, _ = roc_curve(Y_train, xgbow_train_proba[:, 1])
xgbow_fpr_test, xgbow_tpr_test, _ = roc_curve(Y_test, xgbow_test_proba[:, 1])
xgbow_test_auc = auc(xgbow_fpr_test, xgbow_tpr_test)
xgbow_train_auc = auc(xgbow_fpr_train, xgbow_tpr_train)
print(xgbow_test_auc)
print(xgbow_train_auc)
```

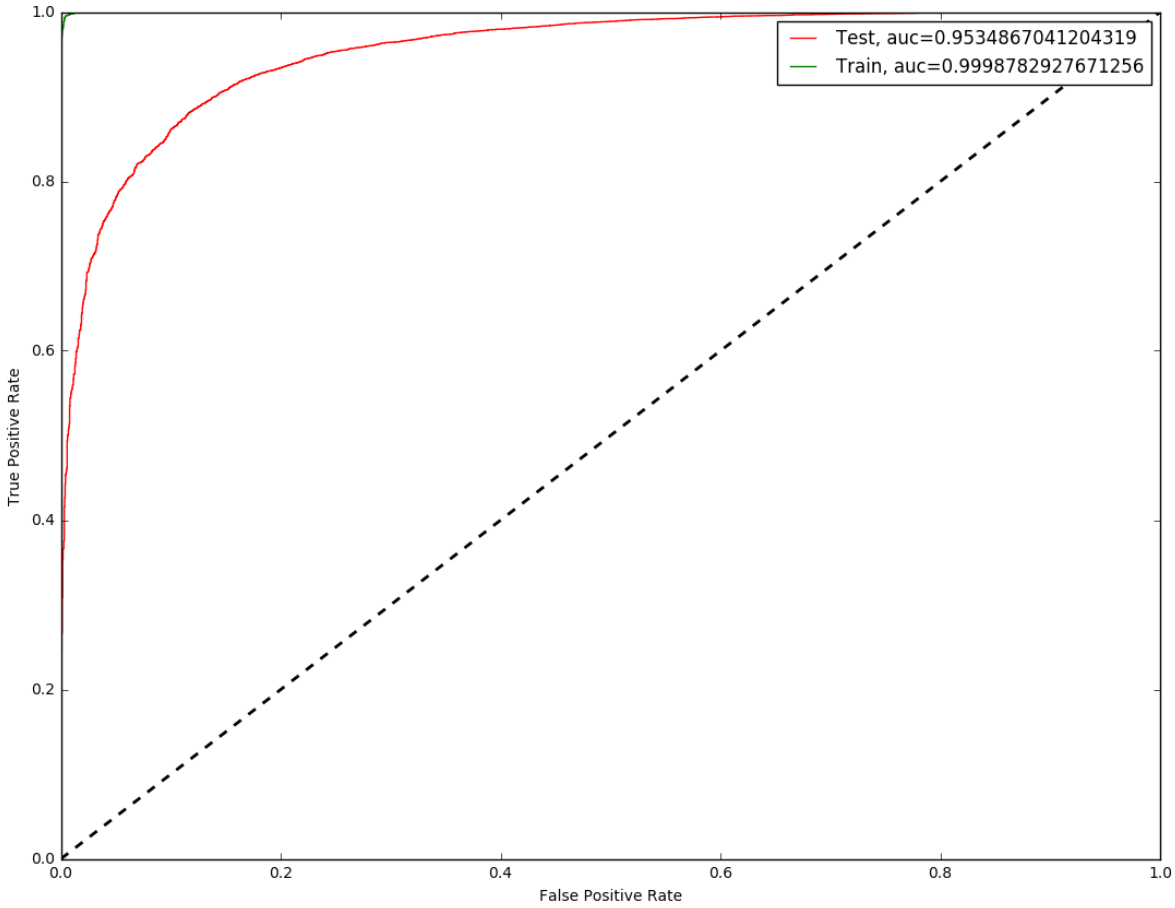
```
0.9534867041204319
0.9998782927671256
```

In [81]:

```
import pylab
plt.figure(figsize=(13, 10))
plt.plot([0,1], [0,1], color='black', lw=2, linestyle='--')
plt.plot(xgbow_fpr_test, xgbow_tpr_test, label="Test, auc="+str(xgbow_test_auc), color = 'r
plt.plot(xgbow_fpr_train, xgbow_tpr_train, label="Train, auc="+str(xgbow_train_auc), color

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

plt.show()
```



In [82]:

```
xgbow_test_conf = xgbow_clf.predict(BoW_dict['X_test_vect'])
xgbow_train_conf = xgbow_clf.predict(BoW_dict['X_train_vect'])
```

In [83]:

```
from sklearn.metrics import classification_report, confusion_matrix
xgbow_train_conf_matrix = confusion_matrix(Y_train, xgbow_train_conf)
xgbow_test_conf_matrix = confusion_matrix(Y_test, xgbow_test_conf)
class_report = classification_report(Y_test, xgbow_test_conf)
print(xgbow_test_conf_matrix)
print(class_report)
```

```
[[ 1459  1217]
 [  257 17067]]
              precision    recall  f1-score   support

           0       0.85      0.55      0.66      2676
           1       0.93      0.99      0.96     17324

   micro avg       0.93      0.93      0.93     20000
   macro avg       0.89      0.77      0.81     20000
weighted avg       0.92      0.93      0.92     20000
```

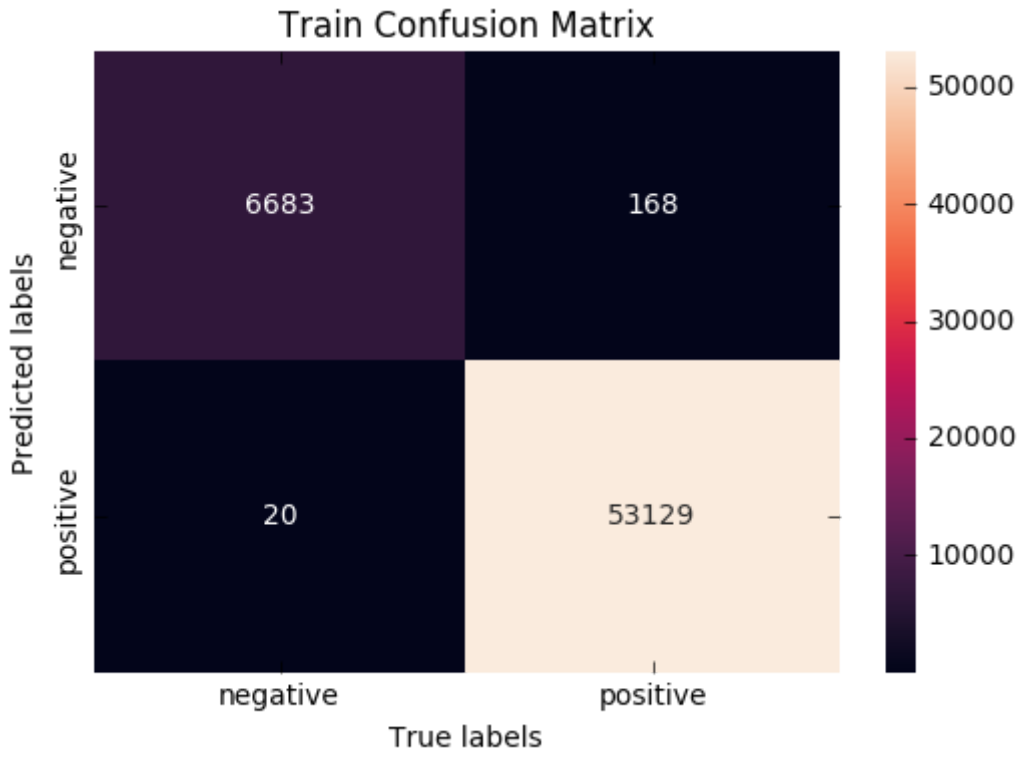
In [84]:

```
ax= plt.subplot()
sns.heatmap(xgbow_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[84]:

```
[<matplotlib.text.Text at 0x7f1e130e14a8>,
 <matplotlib.text.Text at 0x7f1dcc31d0b8>]
```



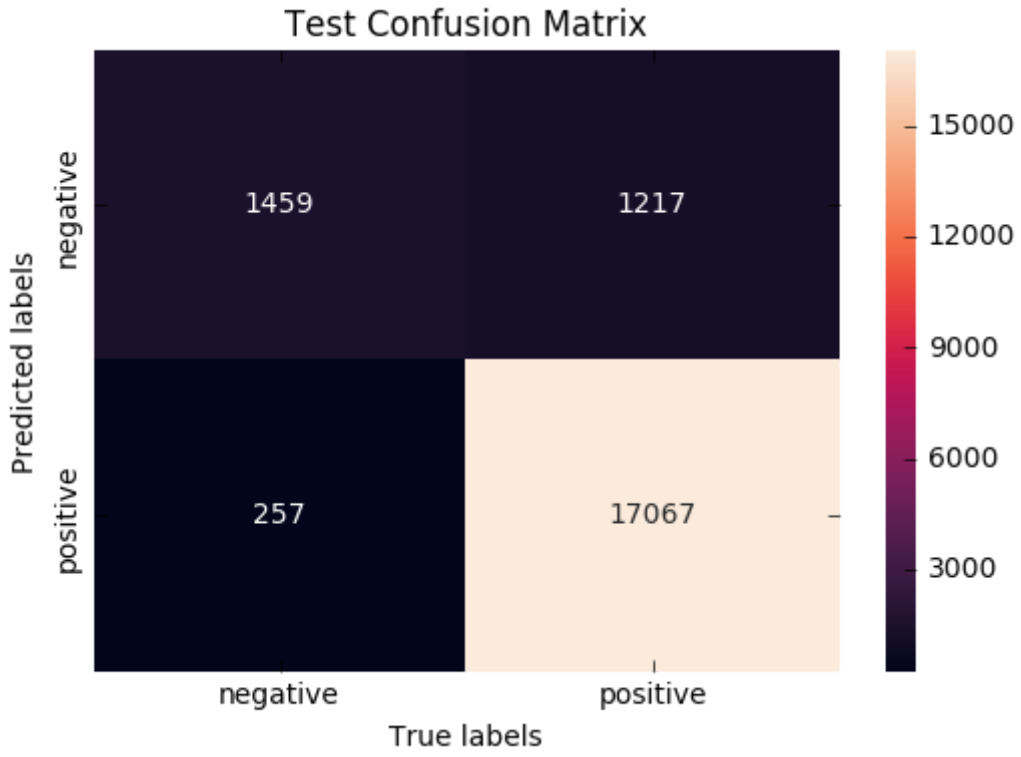
In [85]:

```
ax= plt.subplot()
sns.heatmap(xgbow_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[85]:

```
[<matplotlib.text.Text at 0x7f1e11ef7208>,
 <matplotlib.text.Text at 0x7f1e11ef1128>]
```



In [86]:

```
xgbow_complete_string = '-'.join(xgbow_features_list)
```

In [87]:

```
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import pandas as pd

stopwords = set(STOPWORDS)

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(xgbow_complete_string)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

GBDT using XGBOOST on tfidf

In [88]:

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

In [89]:

```
from scipy.sparse import vstack
X_train_val_tfidf = vstack((tfidf_dict['train_tf_idf'], tfidf_dict['cv_tf_idf']))
```

In [90]:

```
print(X_train_val_tfidf.shape)
```

```
(80000, 35874)
```

In [91]:

```
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV

param_grid = {
    'max_depth': [4, 8, 16, 32],
    'n_estimators': [1, 2, 5, 10, 50, 100, 200]
}

tfidf_xgbclf = XGBClassifier(n_jobs=-1)
tfidf_gdclf = GridSearchCV(estimator = tfidf_xgbclf, param_grid = param_grid, scoring = 'ro
tfidf_gdclf.fit(X_train_val_tfidf,Y_train_val)
```

Out[91]:

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_b
ylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=-1, nthread=None, objective='binary:logistic',
       random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
       seed=None, silent=True, subsample=1),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'max_depth': [4, 8, 16, 32], 'n_estimators': [1, 2, 5, 1
0, 50, 100, 200]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [92]:

```
xgtfidf_results = tfidf_gdclf.cv_results_
xgtfidf_results
```

Out[92]:

```
{'mean_fit_time': array([  1.15454038,   1.22315248,   1.47843949,   2.071
23677,
         6.33149187,  11.32001885,  21.26337099,   1.21369672,
         1.44228379,   2.10142899,   3.16489387,  11.5389142 ,
        21.59475581,  40.93587669,   1.47608336,   1.93204482,
         3.29741025,   5.55901694,  22.5280002 ,  42.10141325,
        78.75508555,   1.91662184,   2.79964137,   5.53901315,
         9.96795376,  42.10870012,  78.04482206, 146.04183801]),
 'mean_score_time': array([0.46674903, 0.48436189, 0.46560113, 0.46560438,
0.48184252,
        0.47819734, 0.48896964, 0.46492147, 0.4698325 , 0.46862086,
        0.47349477, 0.48575568, 0.50085982, 0.54051948, 0.47274613,
        0.47687046, 0.47940024, 0.48511799, 0.51138043, 0.51324153,
        0.58056966, 0.46533505, 0.4630332 , 0.4801836 , 0.47720019,
        0.53151631, 0.59575367, 0.75504677]),
 'mean_test_score': array([0.6770452 , 0.72532972, 0.79827802, 0.82315935,
0.89919463,
        0.92530959, 0.94342858, 0.77784825, 0.78803394, 0.83401714,
```

In [94]:

```
xgtfidf_x_list = []
xgtfidf_y_list = []
for c1 in tfidf_gdclf.cv_results_['params']:
    xgtfidf_x_list.append(c1['n_estimators'])
for c2 in tfidf_gdclf.cv_results_['params']:
    xgtfidf_y_list.append(c2['max_depth'])
print(xgtfidf_x_list, xgtfidf_y_list)
```

```
[1, 2, 5, 10, 50, 100, 200, 1, 2, 5, 10, 50, 100, 200, 1, 2, 5, 10, 50, 100,
200, 1, 2, 5, 10, 50, 100, 200] [4, 4, 4, 4, 4, 4, 4, 8, 8, 8, 8, 8, 8, 8, 1
6, 16, 16, 16, 16, 16, 16, 32, 32, 32, 32, 32, 32, 32]
```

In [95]:

```
x1 = xgtfidf_x_list
y1 = xgtfidf_y_list
z1 = tfidf_gdclf.cv_results_['mean_train_score'].tolist()
x2 = xgtfidf_x_list
y2 = xgtfidf_y_list
z2 = tfidf_gdclf.cv_results_['mean_test_score'].tolist()
```

In [96]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='n_estimators'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

In [97]:

```
tfidf_gdclf.best_params_
```

Out[97]:

```
{'max_depth': 32, 'n_estimators': 200}
```

In [98]:

```
xgtfidf_best_max_depth = tfidf_gdclf.best_params_['max_depth']
xgtfidf_best_n_estimators = tfidf_gdclf.best_params_['n_estimators']
```

In [99]:

```
from sklearn.ensemble import RandomForestClassifier

xg_rf_clf = RandomForestClassifier(max_depth = xgtfidf_best_max_depth, n_estimators=xgtfidf
xg_rf_clf.fit(tfidf_dict['train_tf_idf'],Y_train)
xgtfidf_test_proba = xg_rf_clf.predict_proba(tfidf_dict['test_tf_idf'])
xgtfidf_train_proba = xg_rf_clf.predict_proba(tfidf_dict['train_tf_idf'])
xgtfidf_test_proba
```

Out[99]:

```
array([[0.06563725, 0.93436275],
       [0.2154881 , 0.7845119 ],
       [0.1261096 , 0.8738904 ],
       ...,
       [0.09795343, 0.90204657],
       [0.06021106, 0.93978894],
       [0.27127318, 0.72872682]])
```

In [102]:

```
print("Top 20 Important Features")
d = sorted(list(zip(tf_idf_vect.get_feature_names(), xg_rf_clf.feature_importances_ )), key
xgfeatures_list_tfidf = []
for (i,j) in d:
    xgfeatures_list_tfidf.append(i)
```

```
Top 20 Important Features
```

In [103]:

```
xgtfidf_fpr_train, xgtfidf_tpr_train, _ = roc_curve(Y_train, xgtfidf_train_proba[:, 1])
xgtfidf_fpr_test, xgtfidf_tpr_test, _ = roc_curve(Y_test, xgtfidf_test_proba[:, 1])
xgtfidf_test_auc = auc(xgtfidf_fpr_test, xgtfidf_tpr_test)
xgtfidf_train_auc = auc(xgtfidf_fpr_train, xgtfidf_tpr_train)
print(xgtfidf_test_auc)
print(xgtfidf_train_auc)
```

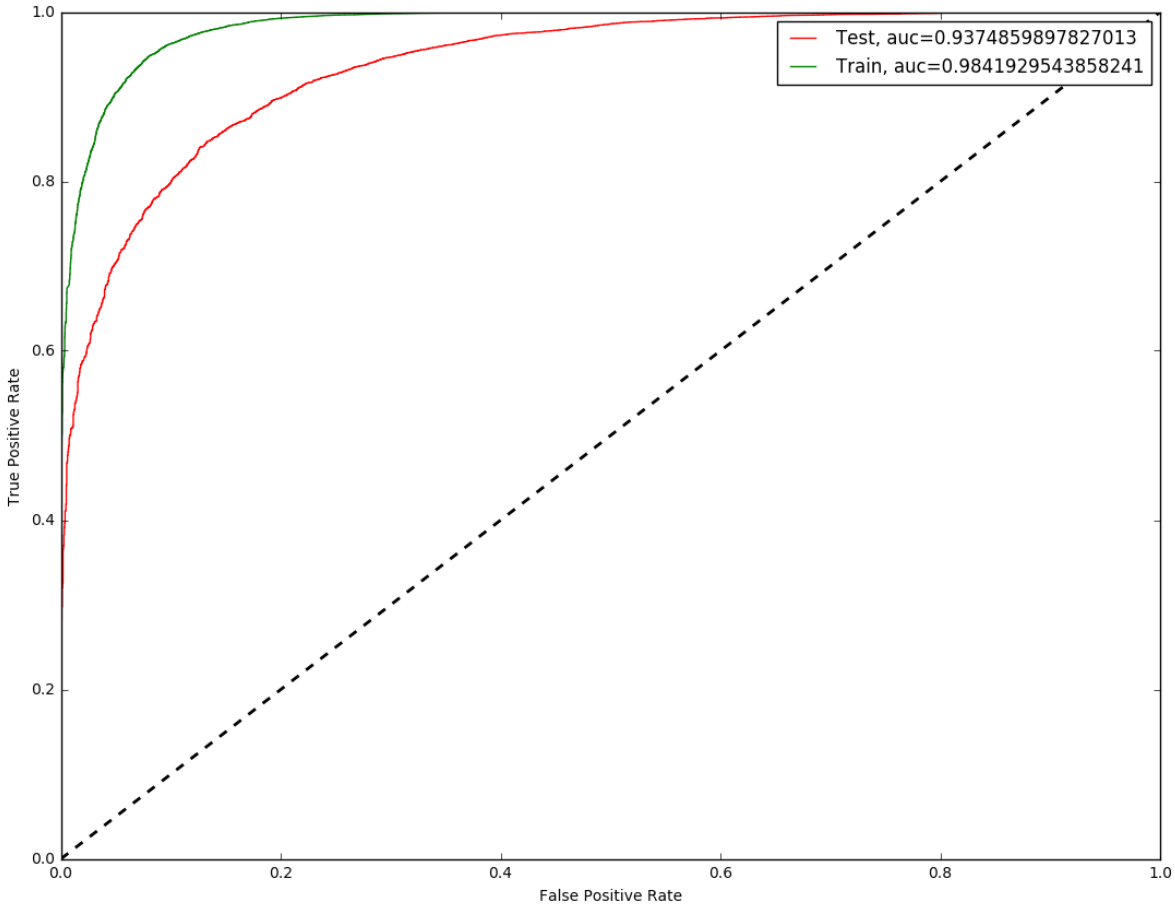
```
0.9374859897827013
0.9841929543858241
```

In [104]:

```
import pylab
plt.figure(figsize=(13, 10))
plt.plot([0,1], [0,1], color='black', lw=2, linestyle='--')
plt.plot(xgtfidf_fpr_test, xgtfidf_tpr_test, label="Test, auc="+str(xgtfidf_test_auc), colo
plt.plot(xgtfidf_fpr_train, xgtfidf_tpr_train, label="Train, auc="+str(xgtfidf_train_auc),

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

plt.show()
```



In [105]:

```
xgtfidf_test_conf = xg_rf_clf.predict(tfidf_dict['test_tf_idf'])
xgtfidf_train_conf = xg_rf_clf.predict(tfidf_dict['train_tf_idf'])
```

In [106]:

```
from sklearn.metrics import classification_report, confusion_matrix
xgtfidf_train_conf_matrix = confusion_matrix(Y_train, xgtfidf_train_conf)
xgtfidf_test_conf_matrix = confusion_matrix(Y_test, xgtfidf_test_conf)
class_report = classification_report(Y_test, xgtfidf_test_conf)
print(xgtfidf_test_conf_matrix)
print(class_report)
```

```
[[   37  2639]
 [    0 17324]]
              precision    recall  f1-score   support

           0       1.00      0.01      0.03      2676
           1       0.87      1.00      0.93     17324

   micro avg       0.87      0.87      0.87     20000
   macro avg       0.93      0.51      0.48     20000
weighted avg       0.89      0.87      0.81     20000
```

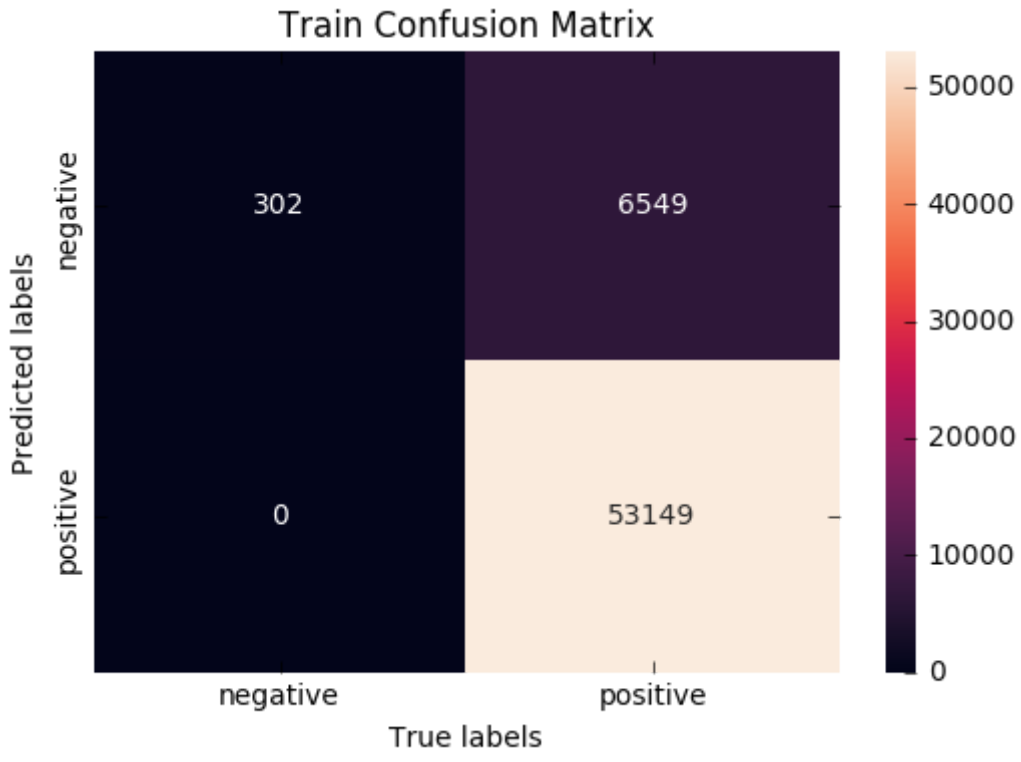
In [107]:

```
ax= plt.subplot()
sns.heatmap(xgtfidf_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[107]:

```
[<matplotlib.text.Text at 0x7f1e12b9a4e0>,
 <matplotlib.text.Text at 0x7f1e1261aeb8>]
```



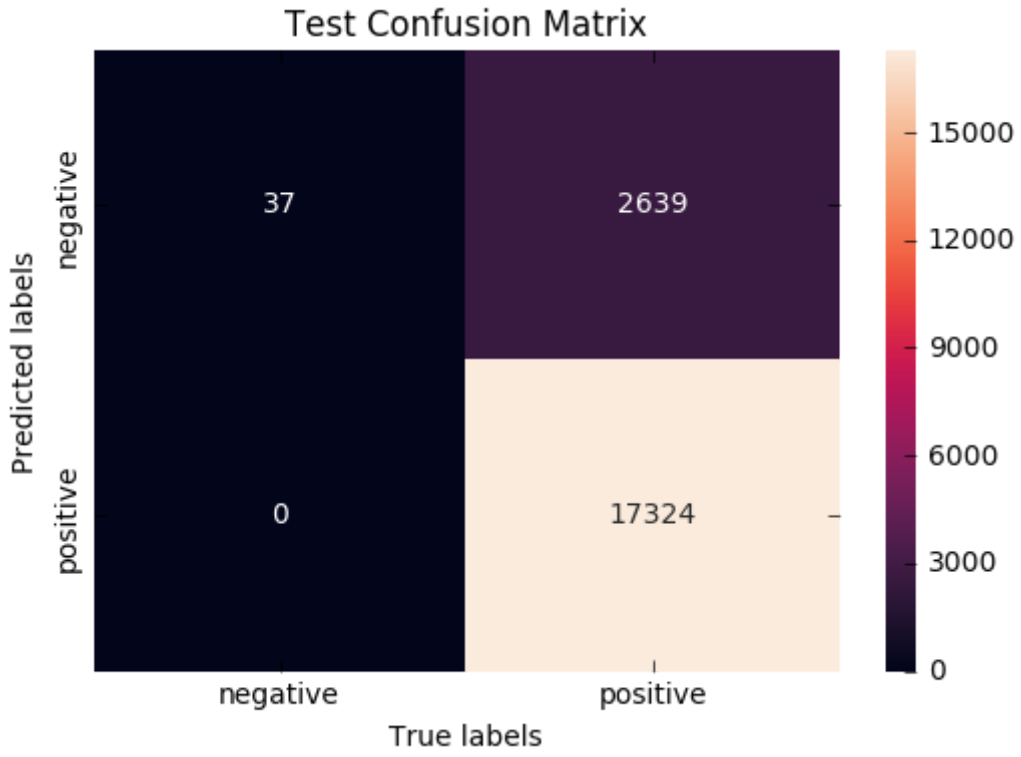
In [108]:

```
ax= plt.subplot()
sns.heatmap(xgtfidf_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[108]:

```
[<matplotlib.text.Text at 0x7f1e11bb6128>,
 <matplotlib.text.Text at 0x7f1e13d12630>]
```



In [109]:

```
xgcomplete_string_tfidf = '-'.join(xgfeatures_list_tfidf)
```

In [110]:

```
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import pandas as pd

stopwords = set(STOPWORDS)

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(xgcomplete_string_tfidf)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

GBDT using XGBOOST on avg-w2v

In [111]:

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

In [16]:

```
from scipy.sparse import vstack
X_train_val_avg = vstack((avg_tfidf_dict['X_train_avgw2v'], avg_tfidf_dict['X_val_avgw2v'])
```

In [17]:

```
print(X_train_val_avg.shape)
```

```
(80000, 50)
```

In [116]:

```
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
import datetime

param_grid = {
    'max_depth': [4, 8, 16, 32],
    'n_estimators': [1, 2, 5, 10, 50, 100, 200]
}

t1 = datetime.datetime.now()
avg_xgbclf = XGBClassifier(n_jobs=-1)
avg_gdclf = GridSearchCV(estimator = avg_xgbclf, param_grid = param_grid, scoring = 'roc_au
avg_gdclf.fit(X_train_val_avg,Y_train_val)
print("time required", datetime.datetime.now() - t1)
```

```
time required 0:28:52.268966
```

In [117]:

```
f = avg_gdclf.cv_results_
f
```

Out[117]:

```
{'mean_fit_time': array([ 1.17645017,  1.26936706,  1.58899164,  2.1131396
3,  6.51726969,
        11.82419213, 23.28593818,  1.29744546,  1.41104364,  2.09005872,
         3.28395851, 12.48343643, 24.00657225, 46.75317788,  1.51324352,
         1.93123881,  3.26372329,  5.50713682, 23.16014981, 43.86378821,
        82.28885452,  1.57790407,  2.0269649 ,  3.50376527,  6.06640379,
        25.71332431, 48.88165998, 90.75627939]),
 'mean_score_time': array([0.48871764, 0.49799951, 0.49118463, 0.49914336,
0.52270222,
        0.51801165, 0.55580235, 0.49695873, 0.49018224, 0.49298422,
        0.50069674, 0.5169038 , 0.53712893, 0.58581821, 0.49196998,
        0.49053931, 0.49993483, 0.50502531, 0.53616818, 0.57855376,
        0.67732938, 0.49318091, 0.49432389, 0.49919113, 0.50681106,
        0.54670572, 0.59234667, 0.69435827]),
 'mean_test_score': array([0.79973793, 0.82869391, 0.84579926, 0.85838639,
0.90699163,
        0.91996567, 0.92661367, 0.8307186 , 0.85331974, 0.87621143,
        0.88888341, 0.91906127, 0.9255619 , 0.9278548 , 0.81792651,
```

In [118]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
```

In [119]:

```
x1_list_xgavg = []
x2_list_xgavg = []
for c1 in avg_gdclf.cv_results_['params']:
    x1_list_xgavg.append(c1['n_estimators'])
for c2 in avg_gdclf.cv_results_['params']:
    x2_list_xgavg.append(c2['max_depth'])
print(x1_list_xgavg, x2_list_xgavg)
```

```
[1, 2, 5, 10, 50, 100, 200, 1, 2, 5, 10, 50, 100, 200, 1, 2, 5, 10, 50, 100,
200, 1, 2, 5, 10, 50, 100, 200] [4, 4, 4, 4, 4, 4, 4, 8, 8, 8, 8, 8, 8, 8, 1
6, 16, 16, 16, 16, 16, 16, 32, 32, 32, 32, 32, 32, 32]
```

In [120]:

```
x1 = x1_list_xgavg
y1 = x2_list_xgavg
z1 = avg_gdclf.cv_results_['mean_train_score'].tolist()
x2 = x1_list_xgavg
y2 = x2_list_xgavg
z2 = avg_gdclf.cv_results_['mean_test_score'].tolist()
```

In [121]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='n_estimators'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

In [122]:

```
avg_gdclf.best_params_
```

Out[122]:

```
{'max_depth': 32, 'n_estimators': 200}
```

In [123]:

```
xgbest_max_depth_avg = avg_gdclf.best_params_['max_depth']
xgbest_n_estimators_avg = avg_gdclf.best_params_['n_estimators']
```

In [124]:

```
from sklearn.ensemble import RandomForestClassifier

xgavg_rf_clf = RandomForestClassifier(max_depth = xgbest_max_depth_avg, n_estimators=xgbest
xgavg_rf_clf.fit(avg_tfidf_dict['X_train_avgw2v'],Y_train)
xgavg_test_proba = xgavg_rf_clf.predict_proba(avg_tfidf_dict['X_test_avgw2v'])
xgavg_train_proba = xgavg_rf_clf.predict_proba(avg_tfidf_dict['X_train_avgw2v'])
xgavg_test_proba
```

Out[124]:

```
array([[0.095, 0.905],
       [0.61 , 0.39 ],
       [0.025, 0.975],
       ...,
       [0.25 , 0.75 ],
       [0.   , 1.   ],
       [0.515, 0.485]])
```

In [125]:

```
xgavg_fpr_train, xgavg_tpr_train, _ = roc_curve(Y_train, xgavg_train_proba[:, 1])
xgavg_fpr_test, xgavg_tpr_test, _ = roc_curve(Y_test, xgavg_test_proba[:, 1])
xgavg_test_auc = auc(xgavg_fpr_test, xgavg_tpr_test)
xgavg_train_auc = auc(xgavg_fpr_train, xgavg_tpr_train)
print(xgavg_test_auc)
print(xgavg_train_auc)
```

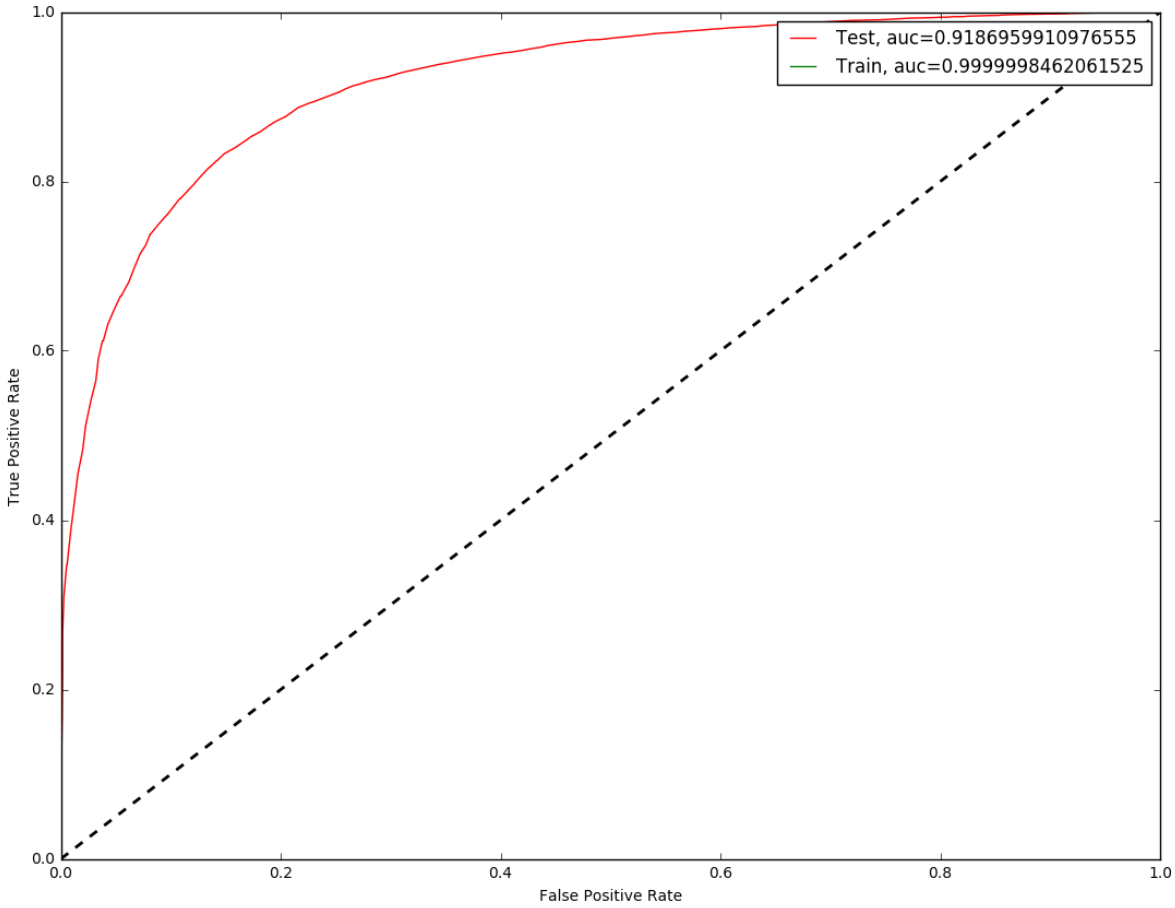
```
0.9186959910976555
0.9999998462061525
```

In [126]:

```
import pylab
plt.figure(figsize=(13, 10))
plt.plot([0,1], [0,1], color='black', lw=2, linestyle='--')
plt.plot(xgavg_fpr_test, xgavg_tpr_test, label="Test, auc="+str(xgavg_test_auc), color = 'r
plt.plot(xgavg_fpr_train, xgavg_tpr_train, label="Train, auc="+str(xgavg_train_auc), color

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

plt.show()
```



In [127]:

```
xgavg_test_conf = xgavg_rf_clf.predict(avg_tfidf_dict['X_test_avgw2v'])
xgavg_train_conf = xgavg_rf_clf.predict(avg_tfidf_dict['X_train_avgw2v'])
```

In [128]:

```
from sklearn.metrics import classification_report, confusion_matrix
xgavg_train_conf_matrix = confusion_matrix(Y_train, xgavg_train_conf)
xgavg_test_conf_matrix = confusion_matrix(Y_test, xgavg_test_conf)
class_report = classification_report(Y_test, xgavg_test_conf)
print(xgavg_test_conf_matrix)
print(class_report)
```

```
[[  771  1905]
 [  182 17142]]
              precision    recall  f1-score   support

           0       0.81      0.29      0.42      2676
           1       0.90      0.99      0.94     17324

   micro avg       0.90      0.90      0.90     20000
   macro avg       0.85      0.64      0.68     20000
weighted avg       0.89      0.90      0.87     20000
```

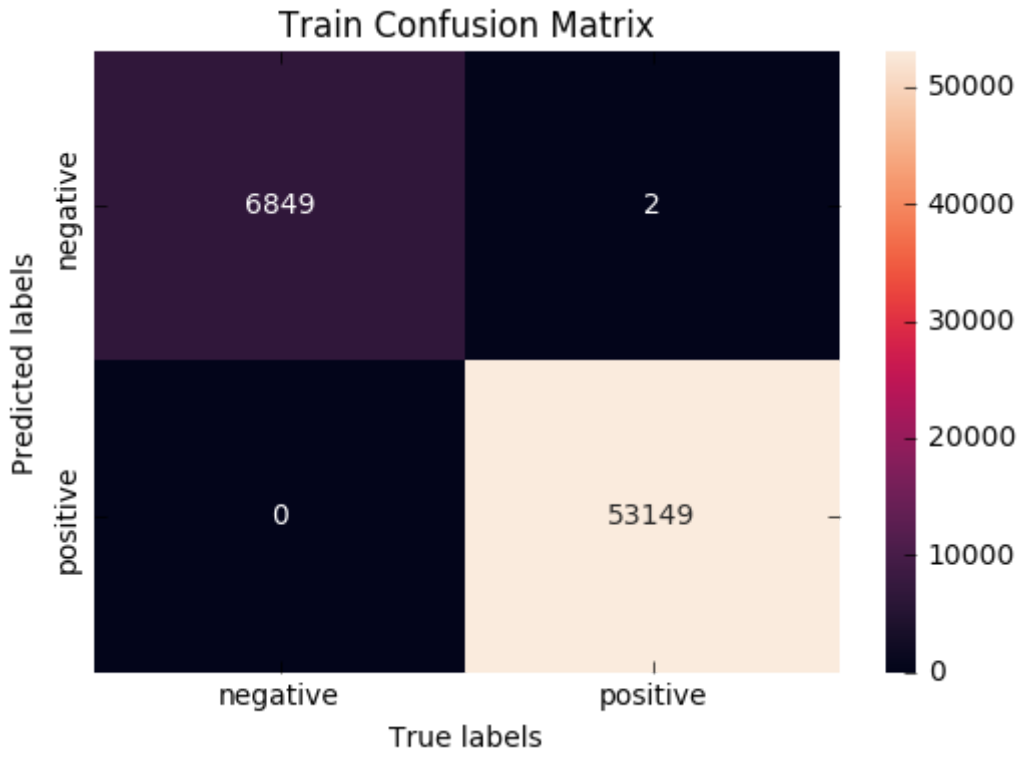
In [129]:

```
ax= plt.subplot()
sns.heatmap(xgavg_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[129]:

```
[<matplotlib.text.Text at 0x7f1e215febe0>,
 <matplotlib.text.Text at 0x7f1e20bcb1d0>]
```



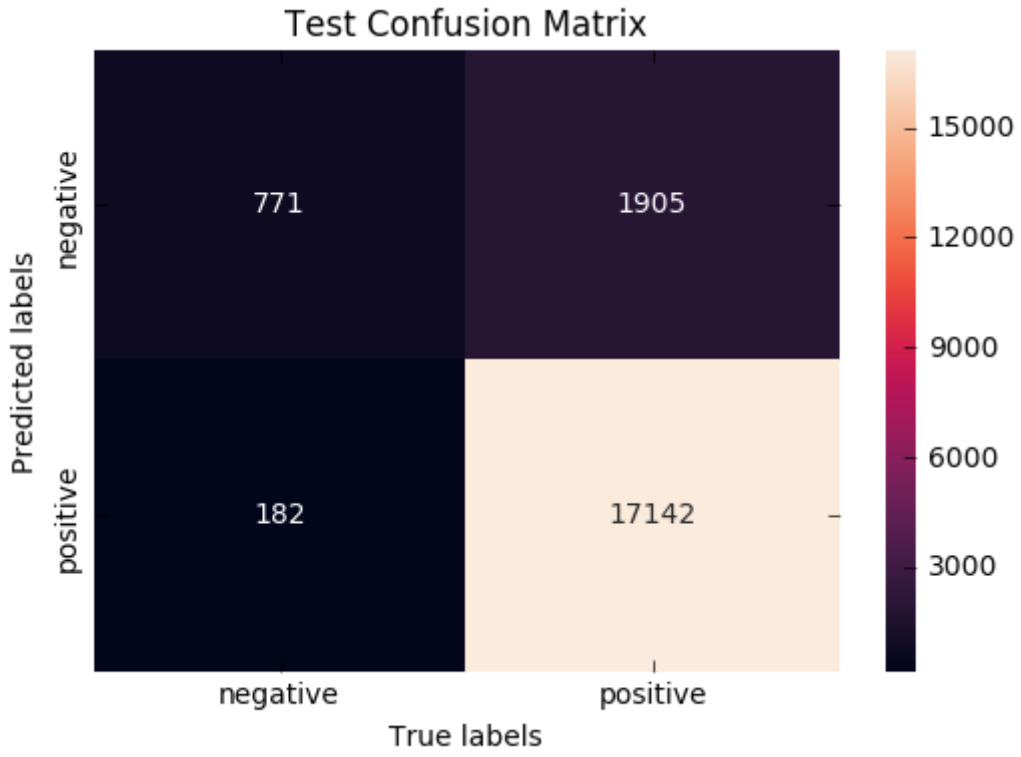
In [130]:

```
ax= plt.subplot()
sns.heatmap(xgavg_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[130]:

```
[<matplotlib.text.Text at 0x7f1e25e9e7f0>,
 <matplotlib.text.Text at 0x7f1e25a26240>]
```



GBDT using XGBOOST on tfidf-w2v

In [131]:

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

In [41]:

```
from scipy.sparse import vstack
X_train_val_tfw2v = vstack((tfidfw2v_dict['X_train_tfidfw2v'], tfidfw2v_dict['X_val_tfidfw2
```

In [42]:

```
print(X_train_val_tfw2v.shape)
```

```
(80000, 50)
```

In [138]:

```
import datetime
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV

param_grid = {
    'max_depth': [4, 8, 16, 32],
    'n_estimators': [1, 2, 5, 10, 50, 100, 200]
}

t1 = datetime.datetime.now()
tfw2v_xgbclf = XGBClassifier(n_jobs=-1)
xgtfw2v_gdclf = GridSearchCV(estimator = tfw2v_xgbclf, param_grid = param_grid, scoring = '
xgtfw2v_gdclf.fit(X_train_val_tfw2v,Y_train_val)
print("time required = ", datetime.datetime.now() - t1)
```

```
time required =  0:28:47.534685
```

In [139]:

```
f = xgtfw2v_gdclf.cv_results_
f
```

Out[139]:

```
{'mean_fit_time': array([ 1.14373295,  1.24053709,  1.53045456,  2.0872089
9,  6.43081133,
        11.85296512, 22.55655877,  1.27224167,  1.47488173,  2.15871414,
         3.25377162, 12.60930554, 24.91552552, 47.24532628,  1.51574643,
         1.93469771,  3.28322975,  5.58516399, 23.58069317, 44.62046178,
        83.11448995,  1.59391952,  2.00370264,  3.69428555,  6.31525238,
        25.94447049, 49.62468942, 92.40105915]),
 'mean_score_time': array([0.48931734, 0.49643215, 0.49868989, 0.49859699,
0.5075353 ,
        0.51749523, 0.54154611, 0.49260767, 0.49565164, 0.49988349,
        0.50114067, 0.53548137, 0.54175043, 0.58863473, 0.49476624,
        0.50120687, 0.50177248, 0.50538802, 0.5408206 , 0.59061392,
        0.69416189, 0.49712666, 0.49728394, 0.5121425 , 0.50961526,
        0.54552182, 0.59943446, 0.72088369]),
 'mean_test_score': array([0.77078016, 0.78876567, 0.80839712, 0.82381749,
0.87838016,
        0.89442241, 0.90354854, 0.7999325 , 0.82009167, 0.84325898,
        0.85850732, 0.89683364, 0.90373111, 0.90610323, 0.7839291 ,
```

In [140]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
```

In [141]:

```
x1_list_xgtfw2v = []
x2_list_xgtfw2v = []
for c1 in xgtfw2v_gdclf.cv_results_['params']:
    x1_list_xgtfw2v.append(c1['n_estimators'])
for c2 in xgtfw2v_gdclf.cv_results_['params']:
    x2_list_xgtfw2v.append(c2['max_depth'])
print(x1_list_xgtfw2v, x2_list_xgtfw2v)
```

```
[1, 2, 5, 10, 50, 100, 200, 1, 2, 5, 10, 50, 100, 200, 1, 2, 5, 10, 50, 100,
200, 1, 2, 5, 10, 50, 100, 200] [4, 4, 4, 4, 4, 4, 4, 8, 8, 8, 8, 8, 8, 8, 1
6, 16, 16, 16, 16, 16, 16, 32, 32, 32, 32, 32, 32, 32]
```

In [142]:

```
x1 = x1_list_xgtfw2v
y1 = x2_list_xgtfw2v
z1 = xgtfw2v_gdclf.cv_results_['mean_train_score'].tolist()
x2 = x1_list_xgtfw2v
y2 = x2_list_xgtfw2v
z2 = xgtfw2v_gdclf.cv_results_['mean_test_score'].tolist()
```

In [143]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='n_estimators'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

In [144]:

```
xgtfw2v_gdclf.best_params_
```

Out[144]:

```
{'max_depth': 16, 'n_estimators': 200}
```

In [145]:

```
best_max_depth_xgtfw2v = xgtfw2v_gdclf.best_params_['max_depth']
best_n_estimators_xgtfw2v = xgtfw2v_gdclf.best_params_['n_estimators']
```

In [146]:

```
from sklearn.ensemble import RandomForestClassifier

xgtfw2v_rf_clf = RandomForestClassifier(max_depth = best_max_depth_xgtfw2v, n_estimators=be
xgtfw2v_rf_clf.fit(tfidfw2v_dict['X_train_tfidfw2v'],Y_train)
xgtfw2v_test_proba = xgtfw2v_rf_clf.predict_proba(tfidfw2v_dict['X_test_tfidfw2v'])
xgtfw2v_train_proba = xgtfw2v_rf_clf.predict_proba(tfidfw2v_dict['X_train_tfidfw2v'])
xgtfw2v_test_proba
```

Out[146]:

```
array([[0.09377032, 0.90622968],
       [0.49312417, 0.50687583],
       [0.0119299 , 0.9880701 ],
       ...,
       [0.17322927, 0.82677073],
       [0.01439599, 0.98560401],
       [0.47568655, 0.52431345]])
```

In [147]:

```
xgtfw2v_fpr_train, xgtfw2v_tpr_train, _ = roc_curve(Y_train, xgtfw2v_train_proba[:, 1])
xgtfw2v_fpr_test, xgtfw2v_tpr_test, _ = roc_curve(Y_test, xgtfw2v_test_proba[:, 1])
xgtfw2v_test_auc = auc(xgtfw2v_fpr_test, xgtfw2v_tpr_test)
xgtfw2v_train_auc = auc(xgtfw2v_fpr_train, xgtfw2v_tpr_train)
print(xgtfw2v_test_auc)
print(xgtfw2v_train_auc)
```

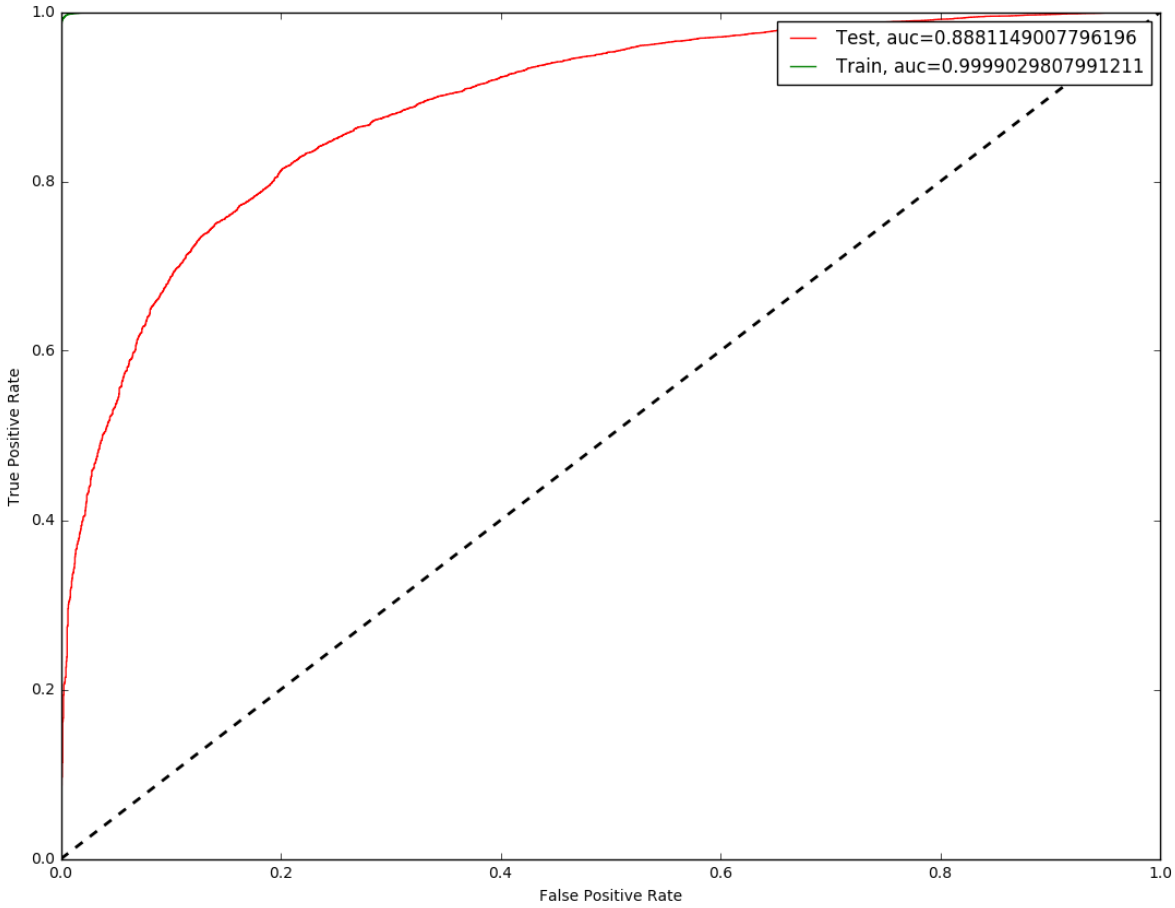
```
0.8881149007796196
0.9999029807991211
```

In [148]:

```
import pylab
plt.figure(figsize=(13, 10))
plt.plot([0,1], [0,1], color='black', lw=2, linestyle='--')
plt.plot(xgtfw2v_fpr_test, xgtfw2v_tpr_test, label="Test, auc="+str(xgtfw2v_test_auc), colo
plt.plot(xgtfw2v_fpr_train, xgtfw2v_tpr_train, label="Train, auc="+str(xgtfw2v_train_auc),

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

plt.show()
```



In [151]:

```
xgtfw2v_test_conf = xgtfw2v_rf_clf.predict(tfidfw2v_dict['X_test_tfidfw2v'])
xgtfw2v_train_conf = xgtfw2v_rf_clf.predict(tfidfw2v_dict['X_train_tfidfw2v'])
```

In [152]:

```
from sklearn.metrics import classification_report, confusion_matrix
xgtfw2v_train_conf_matrix = confusion_matrix(Y_train, xgtfw2v_train_conf)
xgtfw2v_test_conf_matrix = confusion_matrix(Y_test, xgtfw2v_test_conf)
class_report = classification_report(Y_test, xgtfw2v_test_conf)
print(xgtfw2v_test_conf_matrix)
print(class_report)
```

```
[[  553  2123]
 [  160 17164]]
              precision    recall  f1-score   support

           0       0.78      0.21      0.33      2676
           1       0.89      0.99      0.94     17324

   micro avg       0.89      0.89      0.89     20000
   macro avg       0.83      0.60      0.63     20000
weighted avg       0.87      0.89      0.86     20000
```

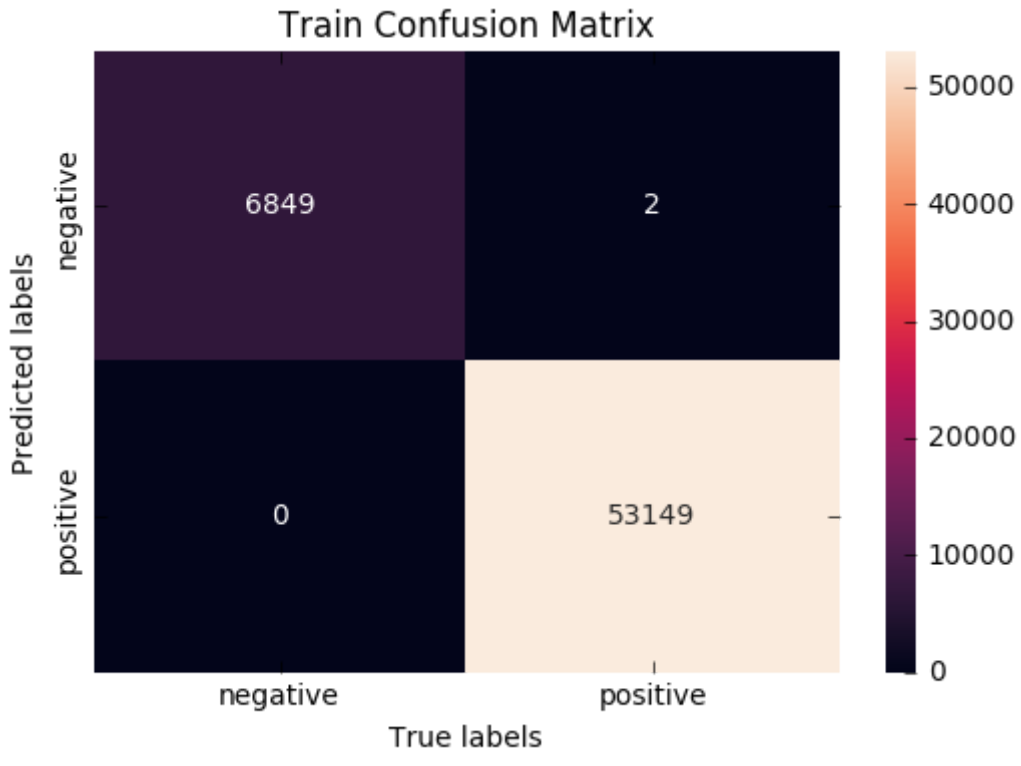
In [153]:

```
ax= plt.subplot()
sns.heatmap(tfw2v_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[153]:

```
[<matplotlib.text.Text at 0x7f1e20979be0>,
 <matplotlib.text.Text at 0x7f1e23a2d710>]
```



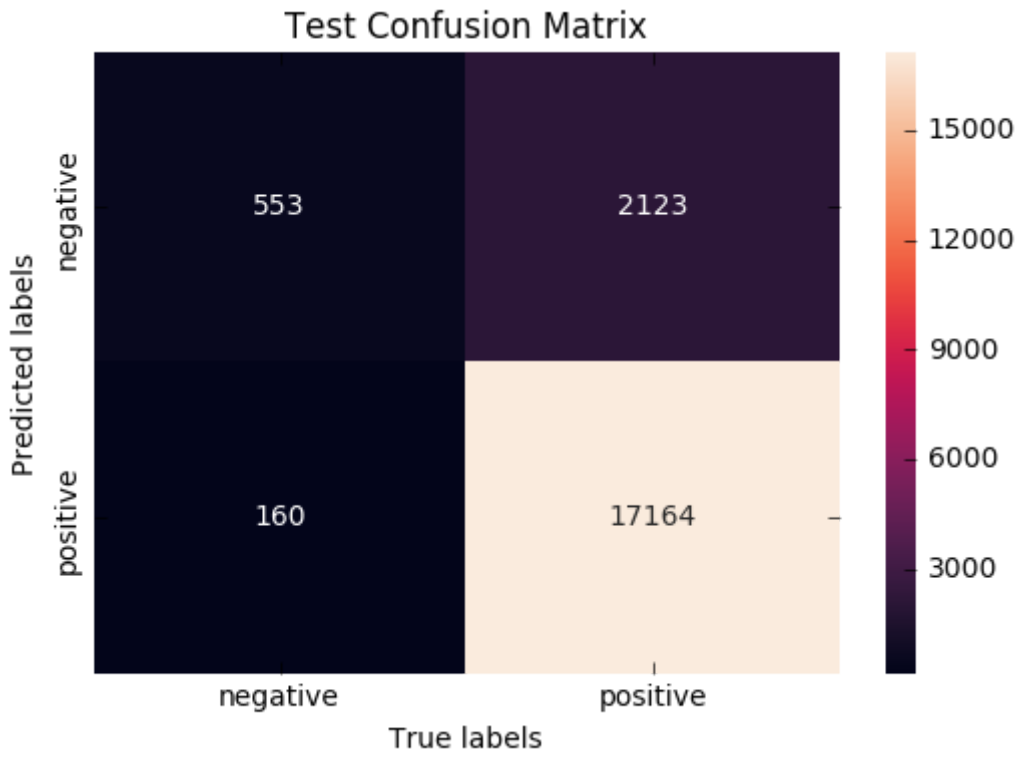
In [154]:

```
ax= plt.subplot()
sns.heatmap(xgtfw2v_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[154]:

```
[<matplotlib.text.Text at 0x7f1e23b5aef0>,
 <matplotlib.text.Text at 0x7f1e24e6e4e0>]
```



In [156]:

```
from prettytable import PrettyTable

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

x.add_row(["Random Forest", 32, 200, "BoW",  0.977, 0.925])
x.add_row(["Random Forest", 32, 200, "Tf-idf", 0.983, 0.937])
x.add_row(["Random Forest", 16, 200, "Avg-w2v", 0.999, 0.917])
x.add_row(["Random Forest", 32, 200, "tfidf_w2v", 0.999, 0.888])
x.add_row(["XGBosst", 32, 200, "BoW",  0.999, 0.953])
x.add_row(["XGBosst", 32, 200, "Tf-idf", 0.984, 0.937])
x.add_row(["XGBosst", 16, 200, "Avg-w2v", 0.999, 0.918])
x.add_row(["XGBosst", 32, 200, "tfidf_w2v", 0.999, 0.888])

print(x)
```

```
+---------------+-----------+--------------+------------+-------+-------+
|   Algorithm   | Max_depth | n_estimators | Vectorizer | Train |  Test |
+---------------+-----------+--------------+------------+-------+-------+
| Random Forest |     32    |     200      |    BoW     | 0.977 | 0.925 |
| Random Forest |     32    |     200      |   Tf-idf   | 0.983 | 0.937 |
| Random Forest |     16    |     200      |  Avg-w2v   | 0.999 | 0.917 |
| Random Forest |     32    |     200      | tfidf_w2v  | 0.999 | 0.888 |
|    XGBosst    |     32    |     200      |    BoW     | 0.999 | 0.953 |
|    XGBosst    |     32    |     200      |   Tf-idf   | 0.984 | 0.937 |
|    XGBosst    |     16    |     200      |  Avg-w2v   | 0.999 | 0.918 |
|    XGBosst    |     32    |     200      | tfidf_w2v  | 0.999 | 0.888 |
+---------------+-----------+--------------+------------+-------+-------+
```

In [ ]: