In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


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

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

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

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

from tqdm import tqdm
import os
```

In [3]:
```
#mounting the dataset from drive
# from google.colab import drive
# drive.mount('/content/gdrive')

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

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data p
# 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 LI
# for tsne assignment you can take 5k data points

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

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

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

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

Out[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulness
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulness
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1

In [4]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [5]:
```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUN
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ET0	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ET0	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBE1U	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [6]:
```
# Removing duplicate reviews
final=filtered_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}
print(final.shape)
```

```
(364173, 10)
```

In [7]:
```
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[7]: 69.25890143662969

In [8]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [9]:
```
#Before starting the next phase of preprocessing lets see the number of entries l
print(final.shape)

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

```
(364171, 10)
```

Out[9]:
```
1    307061
0     57110
Name: Score, dtype: int64
```

In [10]:
```
final["cleanReview"] = final["Summary"].map(str) + ". " + final["Text"]
final['cleanReview'].head()
```

Out[10]:
```
0    Good Quality Dog Food. I have bought several o...
1    Not as Advertised. Product arrived labeled as ...
2    "Delight" says it all. This is a confection th...
3    Cough Medicine. If you are looking for the sec...
4    Great taffy. Great taffy at a great price.  Th...
Name: cleanReview, dtype: object
```

In [11]:
```
final['lengthOfReview'] = final['cleanReview'].str.split().str.len()
final['lengthOfReview'].head()
```

Out[11]:
```
0    52
1    34
2    98
3    43
4    29
Name: lengthOfReview, dtype: int64
```

In [10]:
```
#remove urls from text python
from tqdm import tqdm
lst = []
removed_urls_list = []
for text in tqdm(final['Text']):
  removed_urls_text = re.sub(r"http\S+", "", text)
  lst.append(removed_urls_text)
```

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

In [11]:
```
#remove urls from text python
removed_urls_list = []
for text in tqdm(lst):
  removed_urls_text = re.sub(r"http\S+", "", text)
  removed_urls_list.append(removed_urls_text)
```

```
100%|██████████| 364171/364171 [00:00<00:00, 452270.97it/s]
```

In [12]:
```
from bs4 import BeautifulSoup
text_lst = []
for text in tqdm(removed_urls_list):
  soup = BeautifulSoup(text, 'lxml')
  text = soup.get_text()
  text_lst.append(text)
# print(text)
# print("="*50)
```

```
100%|██████████| 364171/364171 [01:49<00:00, 3330.00it/s]
```

In [13]:
```
print(len(final['Text']))
```

```
364171
```

In [14]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [15]:
```
decat_lst = []
for decat_text in tqdm(text_lst):
  text = decontracted(decat_text)
  decat_lst.append(text)
```

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

In [16]:
```
strip_list = []
for to_strip in tqdm(decat_lst):
  text = re.sub("\S*\d\S*", "", to_strip).strip()
  strip_list.append(text)
```

```
100%|██████████| 364171/364171 [00:22<00:00, 16465.51it/s]
```

In [17]:
```
spatial_list = []
for to_spatial in tqdm(strip_list):
  text = re.sub('[^A-Za-z0-9]+', ' ', to_spatial)
  spatial_list.append(text)
```

```
100%|██████████| 364171/364171 [00:12<00:00, 29401.19it/s]
```

In [18]:
```
stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'our
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', '
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itsel
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because'
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'th
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all'
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than',
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "di
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn'
            'won', "won't", 'wouldn', "wouldn't"])
```

In [19]:
```
# Combining all the above stundents
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(spatial_list):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in s
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 364171/364171 [02:44<00:00, 2216.92it/s]
```

In [20]:
```
print(len(preprocessed_reviews))
preprocessed_reviews[-1]
```

```
364171
```

Out[20]:
```
'satisfied product advertised use cereal raw vinegar general sweetner'
```

In [21]:
```
final['Preprocessed_text'] = preprocessed_reviews
```

In [22]:
```
print(len(final))
final.tail(5)
```

364171

Out[22]:

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

In [93]:
```
dir_path = os.getcwd()
conn = sqlite3.connect(os.path.join(dir_path, 'final.sqlite'))
# final.to_sql('Reviews', conn, if_exists='replace', index=False)
```

In [94]:
```
review_3 = pd.read_sql_query(""" SELECT count(*) FROM Reviews""", conn)
print(review_3)
```

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

In [95]:
```
filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews""", conn)
```

In [96]:
```
filtered_data.shape
```

Out[96]: (364171, 12)

In [97]:
```
filtered_data["Time"] = pd.to_datetime(filtered_data["Time"], unit = "s")
filtered_data = filtered_data.sort_values(by = "Time")
```

In [98]:
```
filtered_data.head(5)
```

Out[98]:

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

In [99]:
```
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 [100]:
```
filtered_data['Score'].value_counts()
```

Out[100]:
```
1    87729
0    12271
Name: Score, dtype: int64
```

In [101]:
```
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 [102]:
```
len(filtered_data['lengthOfReview'])
```

Out[102]:
```
100000
```

In [103]:
```
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 [104]:
```
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 [247]:
```
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_vec
print(X_train_vect.shape)
# print(feature_names)
```

```
(60000, 47535)
```

In [25]:
```
X_train_vect.shape
```

Out[25]:
```
(60000, 47535)
```

In [26]:
```
len(final['lengthOfReview'])
```

Out[26]:
```
364171
```

In [27]:
```
from scipy.sparse import hstack
# len_review = final['lengthOfReview'].to_sparse()
concat_data = hstack((X_train_vect,np.array(final['lengthOfReview'][0:60000])[:,N
concat_data_val = hstack((X_val_vect,np.array(final['lengthOfReview'][60000:80000
concat_data_test = hstack((X_test_vect,np.array(final['lengthOfReview'][80000:100
```

In [28]:
```
print(concat_data.shape)
print(concat_data_val.shape)
print(concat_data_test.shape)
```

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

In [29]:
```
print(len(feature_names))
```

```
47535
```

In [30]:
```
BoW_dict = {'X_train_vect':concat_data, 'X_test_vect': concat_data_test, 'X_val_v
print(BoW_dict['X_train_vect'].shape)
```

```
(60000, 47536)
```

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

[4.3] TF-IDF

In [31]:
```
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
```

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

In [32]:
```
tfidf_concat_data_train = hstack((train_tf_idf,np.array(final['lengthOfReview'][0
tfidf_concat_data_val = hstack((cv_tf_idf,np.array(final['lengthOfReview'][60000:
tfidf_concat_data_test = hstack((test_tf_idf,np.array(final['lengthOfReview'][800
```

In [33]:
```
tf_idf_dict = {'train_tf_idf': tfidf_concat_data_train, 'cv_tf_idf': tfidf_concat
```

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

[4.4] Word2Vec

In [34]:
```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sen=[]
for sentance in X_train:
    list_of_sen.append(sentance.split())
```

In [35]:
```
is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

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

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative3(
        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 = Tr
```

```
[('terrific', 0.8565828204154968), ('excellent', 0.8381140828132629), ('fantast
ic', 0.8366681337356567), ('awesome', 0.7857832908630371), ('wonderful', 0.7829
444408416748), ('good', 0.742619514465332), ('perfect', 0.7174795866012573),
('nice', 0.6593438386917114), ('fabulous', 0.6570981740951538), ('incredible',
0.6524804830551147)]
==================================================
[('greatest', 0.7822151780128479), ('best', 0.7523022294044495), ('tastiest',
0.6484744548797607), ('coolest', 0.6170215606689453), ('terrible', 0.6128978729
248047), ('awful', 0.6031897664070129), ('nicest', 0.5984950661659241), ('nasti
est', 0.5957451462745667), ('closest', 0.5847468376159668), ('softest', 0.57748
57401847839)]
```

In [36]:
```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  15289
sample words  ['flat', 'mater', 'elements', 'crock', 'tripe', 'reversed', 'lact
aid', 'capsule', 'easiest', 'clarify', 'pees', 'swore', 'similiar', 'powdery',
'cement', 'deb', 'burned', 'seasonally', 'stove', 'reinforcement', 'confusion',
'sky', 'mama', 'evil', 'contrast', 'start', 'booklet', 'moves', 'chestnuts', 'v
irtuous', 'monitors', 'twain', 'liquified', 'recommendations', 'quinoa', 'micr
o', 'corned', 'celebrated', 'pitcher', 'clip', 'movie', 'hfcs', 'single', 'left
over', 'inhaled', 'impulse', 'leak', 'gag', 'farming', 'brazilian']
```

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

[4.4.1.1] Avg W2v

In [37]:
```
print(X_train[117924])
print(len(X_val))
print(len(X_test))
```

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

In [38]:
```
# average Word2Vec
# compute average word2vec for each review.
def avg_w2vec(sentences_received):
    sent_vectors = []; # the avg-w2v for each sentence/review is stored in this l
    for sent in tqdm(sentences_received): # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length 50, you migh
        cnt_words =0; # num of words with a valid vector in the sentence/review
        for word in sent: # for each word in a review/sentence
            if word in w2v_words:
                vec = w2v_model.wv[word]
                sent_vec += vec
                cnt_words += 1
        if cnt_words != 0:
            sent_vec /= cnt_words
        sent_vectors.append(sent_vec)

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

In [39]:
```
print(len([sent.split() for sent in X_test]))
```

```
20000
```

In [22]:
```
avg_w2v_train = avg_w2vec([sent.split() for sent in X_train])
avg_w2v_cv = avg_w2vec([sent.split() for sent in X_val])
avg_w2v_test = avg_w2vec([sent.split() for sent in X_test])
```

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

In [ ]:
```
import pickle
with open('/content/gdrive/My Drive/Colab Notebooks/Assignment 3/avg_w2v.pkl', 'w
    pickle.dump(Avg_w2v_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

[4.4.1.2] TFIDF weighted W2v

In [79]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_train)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

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

def tfidf_w2v(sentences_received):
    tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored i
    row=0;
    for sent in tqdm(sentences_received): # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length
        weight_sum =0; # num of words with a valid vector in the sentence/review
        for word in sent: # for each word in a review/sentence
            if word in w2v_words and word in tfidf_feat:
                vec = w2v_model.wv[word]
    #             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
                # to reduce the computation we are
                # dictionary[word] = idf value of word in whole courpus
                # sent.count(word) = tf valeus of word in this review
                tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                sent_vec += (vec * tf_idf)
                weight_sum += tf_idf
        if weight_sum != 0:
            sent_vec /= weight_sum
        tfidf_sent_vectors.append(sent_vec)
        row += 1

    return tfidf_sent_vectors
```

In [73]:
```
tfidf_w2v_train = tfidf_w2v([sent.split() for sent in X_train])
tfidf_w2v_cv = tfidf_w2v([sent.split() for sent in X_val])
tfidf_w2v_test = tfidf_w2v([sent.split() for sent in X_test])
```

In [74]:
```
tfidf_w2v_dict = {'X_train_tfidfw2v':tfidf_w2v_train, 'Y_train_tfidfw2v': Y_train
                     'X_val_tfidfw2v': tfidf_w2v_cv, 'Y_val_tfidfw2v': Y_val,
                    'X_test_tfidfw2v': tfidf_w2v_test, 'Y_test_tfidfw2v': Y_test}
```

In [75]:
```
with open('tfidf_w2v.pkl', 'wb') as handle:
    pickle.dump(tfidf_w2v_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

Important Features

In [256]:
```
#https://stackoverflow.com/questions/26976362/how-to-get-most-informative-feature
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=
    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_coe
    pos_df = pd.DataFrame({'Labels': pos_features_labels,'Coeff':pos_features_coe
#     print("Top 10 featues for negative class \n", neg_df)
#     print("Top 10 featues for positive class \n", pos_df)

    return neg_df, pos_df
```

Logistic Regression on BoW

In [182]:
```
import pickle
with open(r"BoW.pkl", "rb") as input_file:
    BoW_dict = pickle.load(input_file)
```

In [183]:
```
#Applying Logistic Regression with L1 regularization on BOW
from sklearn.linear_model import LogisticRegression
from tqdm import tqdm

bow_lgr_train_score_list = []
bow_lgr_val_score_list = []
for c_value in tqdm([0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]):
    bow_lgr=LogisticRegression(C=c_value, penalty='l1')
    bow_lgr.fit(BoW_dict['X_train_vect'],Y_train)

    bow_lgr_train_score = bow_lgr.score(BoW_dict['X_train_vect'], Y_train)
    bow_lgr_train_score_list.append(bow_lgr_train_score)
    bow_lgr_val_score = bow_lgr.score(BoW_dict['X_val_vect'], Y_val)
    bow_lgr_val_score_list.append(bow_lgr_val_score)

c_all = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
bow_train_score = dict(zip(c_all, bow_lgr_train_score_list))
bow_val_score = dict(zip(c_all, bow_lgr_val_score_list))
print(bow_train_score)
print(bow_val_score)
```

```
100%|██████████████████████████████████████████████████
████████████████████████████████████████████| 8/8 [00:35<0
0:00,  4.43s/it]

{0.1: 0.94336666666666669, 1: 0.97496666666666665, 100: 0.99985000000000002, 10
00: 1.0, 0.0001: 0.8858166666666667, 10: 0.99818333333333331, 0.01: 0.906633333
33333329, 0.001: 0.88551666666666662}
{0.1: 0.92879999999999996, 1: 0.93484999999999996, 100: 0.91544999999999999, 10
00: 0.91159999999999997, 0.0001: 0.86280000000000001, 10: 0.92710000000000004,
0.01: 0.89119999999999999, 0.001: 0.86285000000000001}
```

In [184]:
```
#Weight Vector
print(bow_lgr.coef_)
print(type(bow_lgr.coef_))
weight_vector_bow = bow_lgr.coef_
```

```
[[ -2.41838618e-05   1.26487816e+00   0.00000000e+00 ...,   0.00000000e+00
    0.00000000e+00   1.21299189e-02]]
<class 'numpy.ndarray'>
```

In [185]:
```
non_zero = np.count_nonzero(weight_vector_bow)
total_val = np.product(weight_vector_bow.shape)
bow_sparsity = (total_val - non_zero) / total_val
bow_sparsity
```

Out[185]:
```
0.79386149444631438
```

In [186]:
```
noise = np.random.normal(0, 0.01)
print(noise)
print(type(noise))
```

```
-0.00022102768125355397
<class 'float'>
```

In [203]:
```
print(type(BoW_dict['X_train_vect']))
print(type(BoW_dict['X_train_vect'].data))
print(BoW_dict['X_train_vect'].shape)
X_train_dict = BoW_dict['X_train_vect']
print(X_train_dict.shape)
X_train_dict.data = X_train_dict.data + noise
print(X_train_dict.shape)
```

```
<class 'scipy.sparse.coo.coo_matrix'>
<class 'numpy.ndarray'>
(60000, 47536)
(60000, 47536)
(60000, 47536)
```

In [204]:
```
print(X_train_dict.shape)
```

```
(60000, 47536)
```

In [206]:
```
# Pertubation Test
#Fitting the model on X'
for c_value in tqdm([0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]):
    new_bow_lgr=LogisticRegression(C=c_value, penalty='l1')
    new_bow_lgr.fit(X_train_dict,Y_train)
```

```
100%|██████████████████████████████████████████████████████████████████| 8/8 [00:28<00:00,  3.56s/it]
```

In [207]:
```
#Weight Vector of X'
print(new_bow_lgr.coef_)
print(type(new_bow_lgr.coef_))
new_weight_vector_bow = new_bow_lgr.coef_
```

```
[[-0.09693753  1.46580352  0.         ...,  0.          0.          0.011795
]]
<class 'numpy.ndarray'>
```

In [208]:
```
#Adding epsilon to weights to eliminate the divisible by zero error
import sys
epsilon = sys.float_info.epsilon
print(epsilon)
weight_vector_bow = weight_vector_bow + epsilon
new_weight_vector_bow = new_weight_vector_bow + epsilon
```

```
2.220446049250313e-16
```

In [216]:
```
#percentage change between weight_vector_bow and new_weight_vector_bow
perc_change = ((weight_vector_bow - new_weight_vector_bow) / (weight_vector_bow))
ten_percentile = np.percentile(perc_change, 10)
twenty_percentile = np.percentile(perc_change, 20)
thirty_percentile = np.percentile(perc_change, 30)
forty_percentile = np.percentile(perc_change, 40)
fifty_percentile = np.percentile(perc_change, 50)
sixty_percentile = np.percentile(perc_change, 60)
seventy_percentile = np.percentile(perc_change, 70)
eighty_percentile = np.percentile(perc_change, 80)
ninety_percentile = np.percentile(perc_change, 90)
hundred_percentile = np.percentile(perc_change, 100)
print("ten_percentile", ten_percentile)
print("twenty_percentile", twenty_percentile)
print("thirty_percentile", thirty_percentile)
print("forty_percentile", forty_percentile)
print("fifty_percentile", fifty_percentile)
print("sixty_percentile", sixty_percentile)
print("seventy_percentile", seventy_percentile)
print("eighty_percentile", eighty_percentile)
print("ninety_percentile", ninety_percentile)
print("hundred_percentile", hundred_percentile)
```

```
ten_percentile -2.22676499016
twenty_percentile 0.0
thirty_percentile 0.0
forty_percentile 0.0
fifty_percentile 0.0
sixty_percentile 0.0
seventy_percentile 0.0
eighty_percentile 0.0
ninety_percentile 4.35526461639
hundred_percentile 4.88824366587e+18
```

In [225]:
```
print("91 percentile", np.percentile(perc_change, 91))
print("92 percentile", np.percentile(perc_change, 92))
print("93 percentile", np.percentile(perc_change, 93))
print("94 percentile", np.percentile(perc_change, 94))
print("95 percentile", np.percentile(perc_change, 95))
print("96 percentile", np.percentile(perc_change, 96))
print("97 percentile", np.percentile(perc_change, 97))
print("98 percentile", np.percentile(perc_change, 98))
print("98.1 percentile", np.percentile(perc_change, 98.1))
print("98.2 percentile", np.percentile(perc_change, 98.2))
print("98.3 percentile", np.percentile(perc_change, 98.3))
print("98.4 percentile", np.percentile(perc_change, 98.4))
print("99 percentile", np.percentile(perc_change, 99))
```

```
91 percentile 5.83473633284
92 percentile 7.96208613703
93 percentile 10.946089915
94 percentile 15.4456999729
95 percentile 22.1827452568
96 percentile 34.8650517692
97 percentile 62.2288687875
98 percentile 100.0
98.1 percentile 100.446222057
98.2 percentile 4090.06668365
98.3 percentile 2.76873634162e+15
98.4 percentile 6.47906007185e+15
99 percentile 8.77083120277e+16
```

In [236]:
```
# There is a sudden change after 98.2 percentile
# Consider threshold here to be 4090.06668365
# Feature names of whose % change is more than a threshold

ninetyeight_eight = np.percentile(perc_change, 98.3)
negative_features, positive_features = most_informative_feature_for_binary_classi
print("negative_features", negative_features)
print("positive features", positive_features)
```

```
negative_features         Coeff  Labels Negative features
0 -69.698445      0         jivalime
1 -65.856175      0            coils
2 -58.923367      0      maunfacturer
3 -56.072646      0           storge
4 -55.026199      0             hime
5 -52.226542      0        grainiest
6 -50.265468      0      recommendone
7 -49.293616      0        robitussin
8 -46.987000      0            tacky
9 -46.203121      0      yadayadayada
positive features         Coeff  Labels Positive features
0  52.739630      1      occassionaly
1  50.190990      1         somtimes
2  41.770430      1           usualy
3  41.030733      1          deluted
4  40.410199      1          littled
5  39.147634      1          ranting
6  39.031830      1            yummi
7  36.737932      1            glico
8  36.660119      1            rater
9  36.584551      1         cinnaman
```

Applying Logistic Regression with L2 regularization on BOW

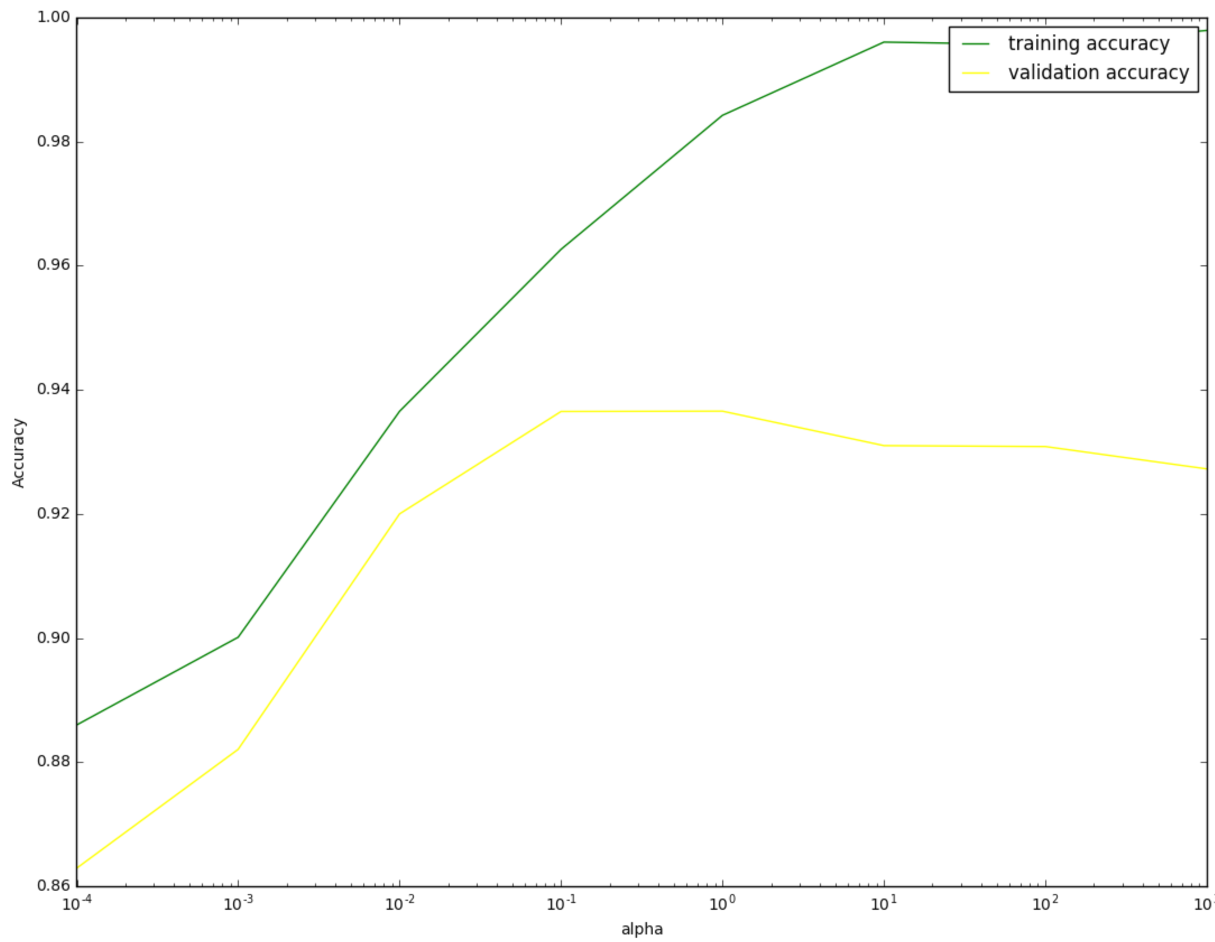
In [249]:
```
bow_lgr_train_score_list = []
bow_lgr_val_score_list = []
for c_value in tqdm([0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]):
    bow_lgr=LogisticRegression(C=c_value, penalty='l2')
    bow_lgr.fit(BoW_dict['X_train_vect'],Y_train)

    bow_lgr_train_score = bow_lgr.score(BoW_dict['X_train_vect'], Y_train)
    bow_lgr_train_score_list.append(bow_lgr_train_score)
    bow_lgr_val_score = bow_lgr.score(BoW_dict['X_val_vect'], Y_val)
    bow_lgr_val_score_list.append(bow_lgr_val_score)

c_all = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
bow_train_score = dict(zip(c_all, bow_lgr_train_score_list))
bow_val_score = dict(zip(c_all, bow_lgr_val_score_list))
print(bow_train_score)
print(bow_val_score)
```

```
100%|█████████████████████████████████████████████████████████████████████████
█████████████████████████████████████████████████████████████████| 8/8 [00:47<0
0:00,  5.90s/it]

{0.1: 0.96263333333333334, 1: 0.98423333333333329, 100: 0.99560000000000004, 10
00: 0.99791666666666667, 0.0001: 0.88598333333333334, 10: 0.99603333333333333,
0.01: 0.93653333333333333, 0.001: 0.90011666666666668}
{0.1: 0.9365, 1: 0.93654999999999999, 100: 0.93084999999999996, 1000: 0.9272500
0000000002, 0.0001: 0.8629, 10: 0.93100000000000005, 0.01: 0.92000000000000004,
0.001: 0.88205}
```

In [250]:
```
best_c = max(bow_val_score, key=bow_val_score.get)
best_c
```

Out[250]: 1

In [251]:
```
import pylab
plt.figure(figsize=(13, 10))
neighbors_settings = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
plt.plot(neighbors_settings, bow_lgr_train_score_list, label="training accuracy",
plt.plot(neighbors_settings, bow_lgr_val_score_list, label="validation accuracy",
# plt.plot(neighbors_settings, auc_test, label="test accuracy", color='red')
plt.xlabel('alpha')
plt.ylabel('Accuracy')
plt.legend()
plt.xscale('log')

plt.show()
```



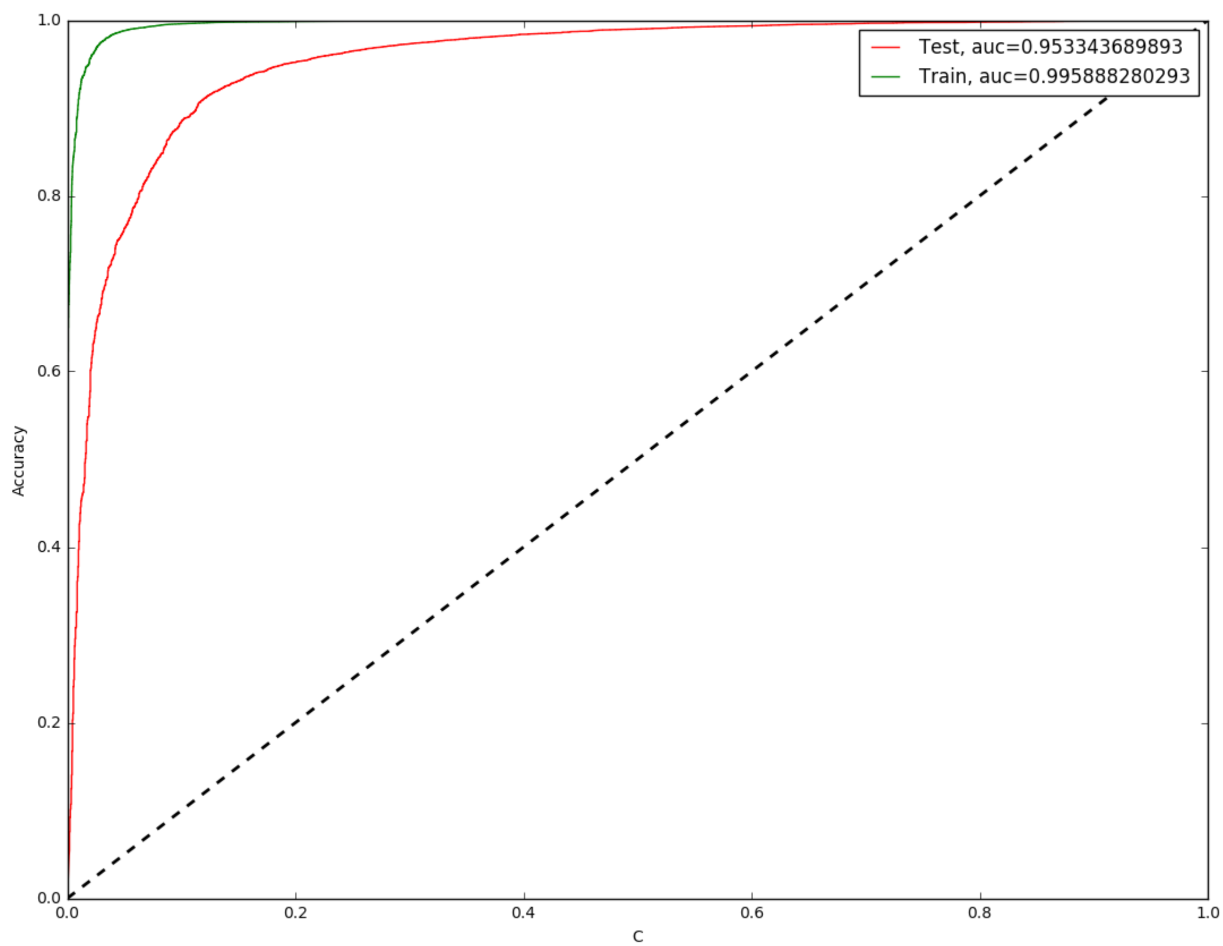
In [252]:
```
bow_lgr=LogisticRegression(C=best_c)
bow_lgr.fit(BoW_dict['X_train_vect'],Y_train)
bow_test_proba = bow_lgr.predict_proba(BoW_dict['X_test_vect'])
bow_train_proba = bow_lgr.predict_proba(BoW_dict['X_train_vect'])
bow_test_proba
```

Out[252]:
```
array([[  4.31712464e-04,   9.99568288e-01],
       [  9.92200284e-01,   7.79971561e-03],
       [  9.18554601e-06,   9.99990814e-01],
       ...,
       [  6.69998480e-02,   9.33000152e-01],
       [  3.86393249e-04,   9.99613607e-01],
       [  9.41828394e-01,   5.81716060e-02]])
```

In [253]:
```
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)
```

```
0.953343689893
0.995888280293
```

In [254]:
```
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
plt.plot(bow_fpr_train, bow_tpr_train, label="Train, auc="+str(bow_train_auc), co

plt.xlabel('C')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



Important Features

In [268]:
```
#https://stackoverflow.com/questions/26976362/how-to-get-most-informative-feature
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=
    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_coe
    pos_df = pd.DataFrame({'Labels': pos_features_labels,'Coeff':pos_features_coe
    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_lgr)
```

```
Top 10 featues for negative class
       Coeff  Labels Negative features
0 -3.479282       0             worst
1 -2.664788       0     disappointing
2 -2.400714       0          terrible
3 -2.369290       0             awful
4 -2.358913       0             hopes
5 -2.275262       0              yuck
6 -2.260761       0           sounded
7 -2.220751       0             threw
8 -2.206621       0          horrible
9 -2.072741       0             bland
Top 10 featues for positive class
       Coeff  Labels Positive features
0  2.488809       1               yum
1  2.456212       1         addictive
2  2.114740       1         delicious
3  2.085181       1        pleasantly
4  2.028609       1         excellent
5  2.017831       1             yummy
6  1.918385       1              beat
7  1.910667       1           perfect
8  1.863848       1           amazing
9  1.854994       1             loves
```

In [258]:
```
bow_test_conf = bow_lgr.predict(BoW_dict['X_test_vect'])
```

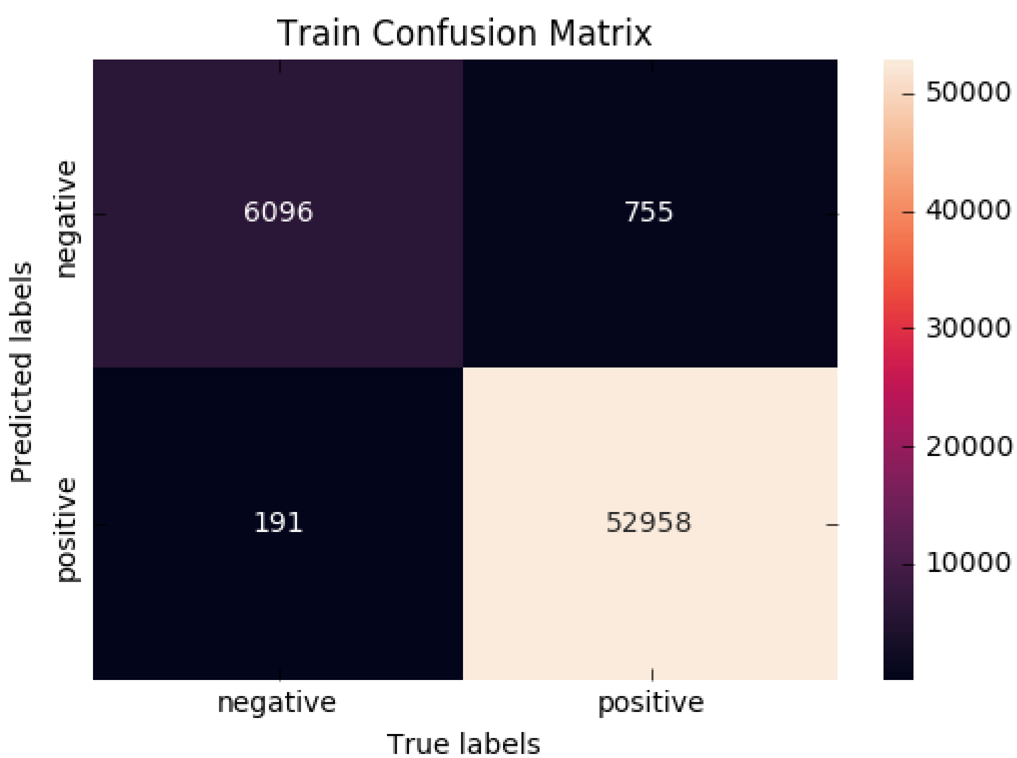
In [259]:
```
bow_train_conf = bow_lgr.predict(BoW_dict['X_train_vect'])
```

In [260]:
```
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)
```

```
[[ 1891   785]
 [  478 16846]]
             precision    recall  f1-score   support

          0       0.80      0.71      0.75      2676
          1       0.96      0.97      0.96     17324

avg / total       0.93      0.94      0.94     20000
```

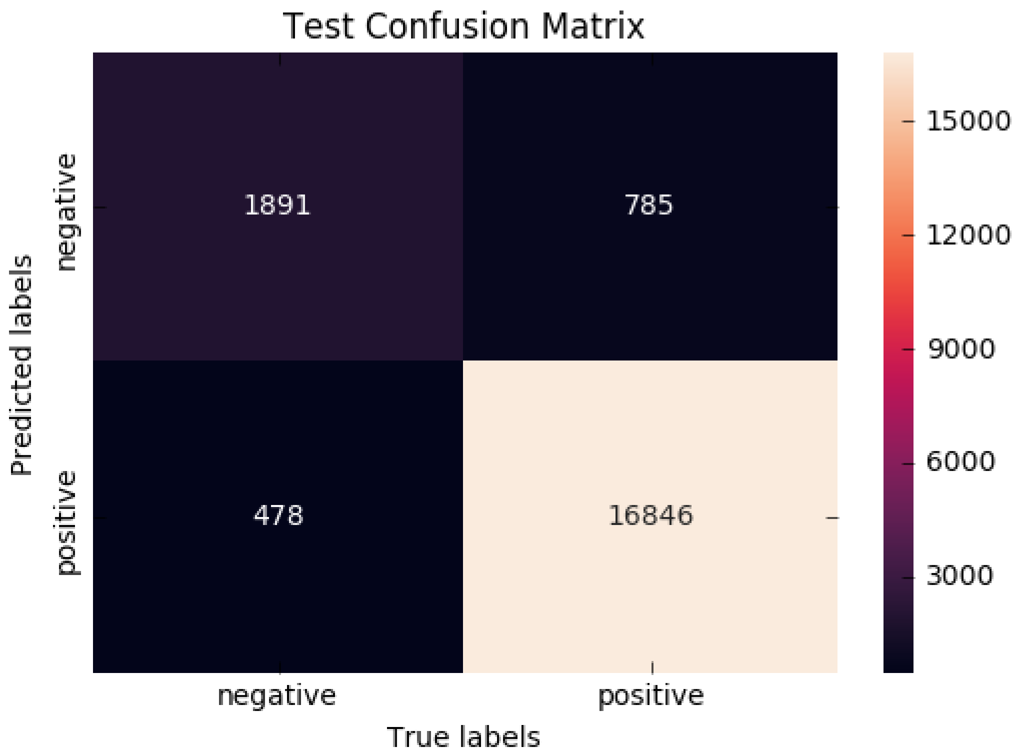
In [261]:
```
ax= plt.subplot()
sns.heatmap(bow_train_conf_matrix, annot=True, ax = ax, fmt='g')

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[261]: [<matplotlib.text.Text at 0x83548940>, <matplotlib.text.Text at 0x2d62e6d8>]



In [262]:
```
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[262]:
```
[<matplotlib.text.Text at 0x9422d198>, <matplotlib.text.Text at 0x885ec908>]
```



Logistic Regression on TF-IDF

In [263]:
```
import pickle
with open(r"tf_idf.pkl", "rb") as input_file:
    tfidf_dict = pickle.load(input_file)
```

In [264]:
```python
from sklearn.linear_model import LogisticRegression
from tqdm import tqdm

tfidf_lgr_train_score_list = []
tfidf_lgr_val_score_list = []
for c_value in tqdm([0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]):
    tfidf_lgr=LogisticRegression(C=c_value)
    tfidf_lgr.fit(tfidf_dict['train_tf_idf'],Y_train)

    tfidf_lgr_train_score = tfidf_lgr.score(tfidf_dict['train_tf_idf'], Y_train)
    tfidf_lgr_train_score_list.append(tfidf_lgr_train_score)
    tfidf_lgr_val_score = tfidf_lgr.score(tfidf_dict['cv_tf_idf'], Y_val)
    tfidf_lgr_val_score_list.append(tfidf_lgr_val_score)

c_all = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
tfidf_train_score = dict(zip(c_all, tfidf_lgr_train_score_list))
tfidf_val_score = dict(zip(c_all, tfidf_lgr_val_score_list))
print(tfidf_train_score)
print(tfidf_val_score)
```

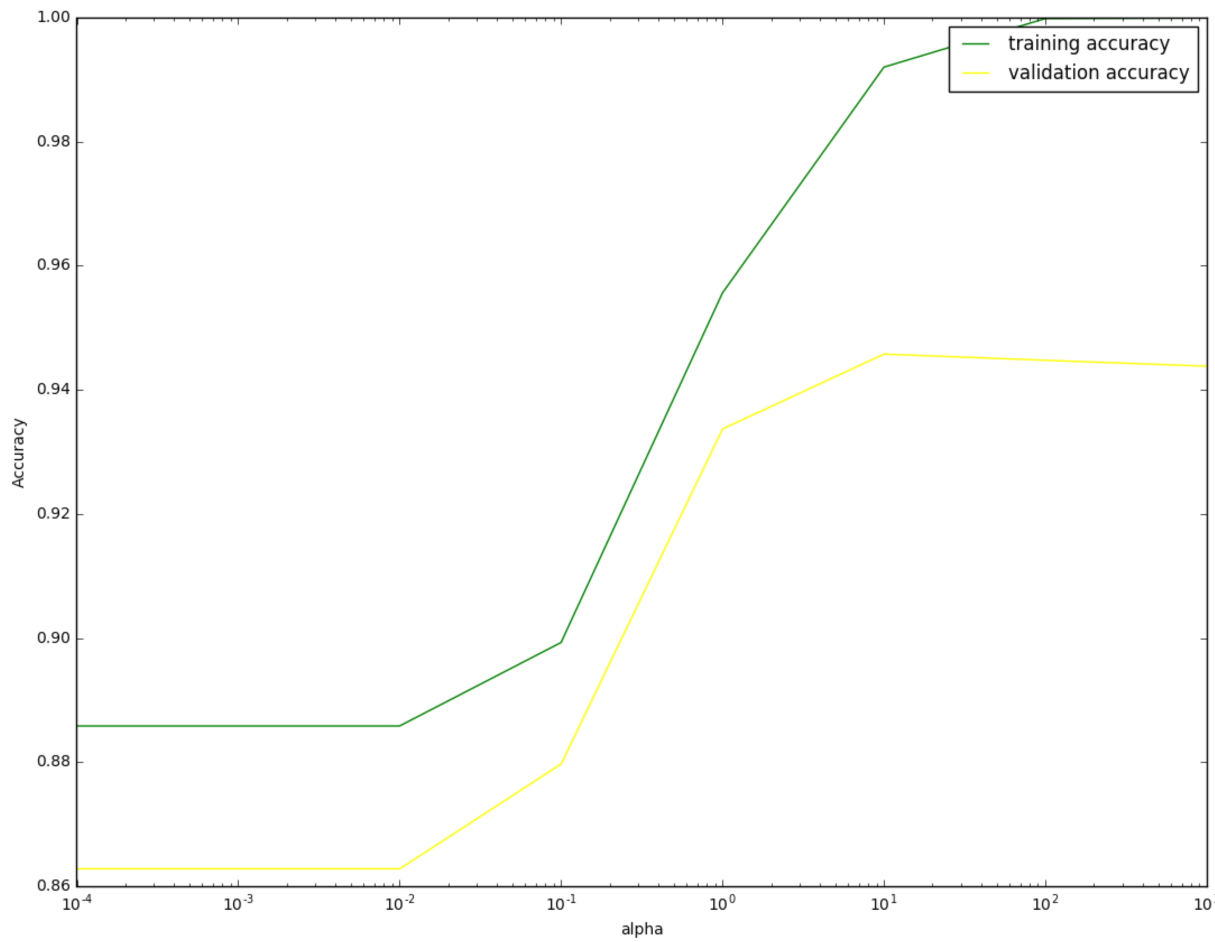
```
100%|██████████████████████████████████████████████████████████████████████| 8/8 [00:16<00:00,  2.06s/it]
```

```
{0.1: 0.8992666666666666, 1: 0.9556166666666667, 100: 0.9998333333333335, 1000: 0.9999333333333334, 0.0001: 0.8858166666666667, 10: 0.9919999999999999, 0.01: 0.8858166666666667, 0.001: 0.8858166666666667}
{0.1: 0.8797000000000004, 1: 0.9336999999999997, 100: 0.9447499999999998, 1000: 0.9437999999999997, 0.0001: 0.8628000000000001, 10: 0.9457499999999998, 0.01: 0.8628000000000001, 0.001: 0.8628000000000001}
```

In [265]:
```python
tfidf_best_c = max(tfidf_val_score, key=tfidf_val_score.get)
tfidf_best_c
```

Out[265]: 10

In [266]:
```
import pylab
plt.figure(figsize=(13, 10))
neighbors_settings = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
plt.plot(neighbors_settings, tfidf_lgr_train_score_list, label="training accuracy
plt.plot(neighbors_settings, tfidf_lgr_val_score_list, label="validation accuracy
# 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 [267]:
```
#https://stackoverflow.com/questions/26976362/how-to-get-most-informative-feature
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=
    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_coe
    pos_df = pd.DataFrame({'Labels': pos_features_labels,'Coeff':pos_features_coe
    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_lgr)
```

```
Top 10 featues for negative class
        Coeff  Labels Negative features
0 -34.732022       0         not worth
1 -33.231000       0             worst
2 -33.048806       0      disappointed
3 -27.741065       0     disappointing
4 -26.608546       0             bland
5 -26.125038       0               not
6 -25.933319       0          not good
7 -25.231868       0     not recommend
8 -23.450598       0         not great
9 -23.385146       0          horrible
Top 10 featues for positive class
        Coeff  Labels Positive features
0  44.037258       1             great
1  35.656608       1              best
2  31.602408       1              good
3  30.191897       1         delicious
4  29.875450       1         excellent
5  29.063318       1  not disappointed
6  27.723844       1             loves
7  27.328193       1           perfect
8  27.234947       1             tasty
9  26.417494       1         wonderful
```

In [269]:

```
tfidf_lgr=LogisticRegression(C=tfidf_best_c)
tfidf_lgr.fit(tfidf_dict['train_tf_idf'], Y_train)
tfidf_test_proba = tfidf_lgr.predict_proba(tfidf_dict['test_tf_idf'])
tfidf_train_proba = tfidf_lgr.predict_proba(tfidf_dict['train_tf_idf'])
tfidf_test_proba
```

Out[269]:

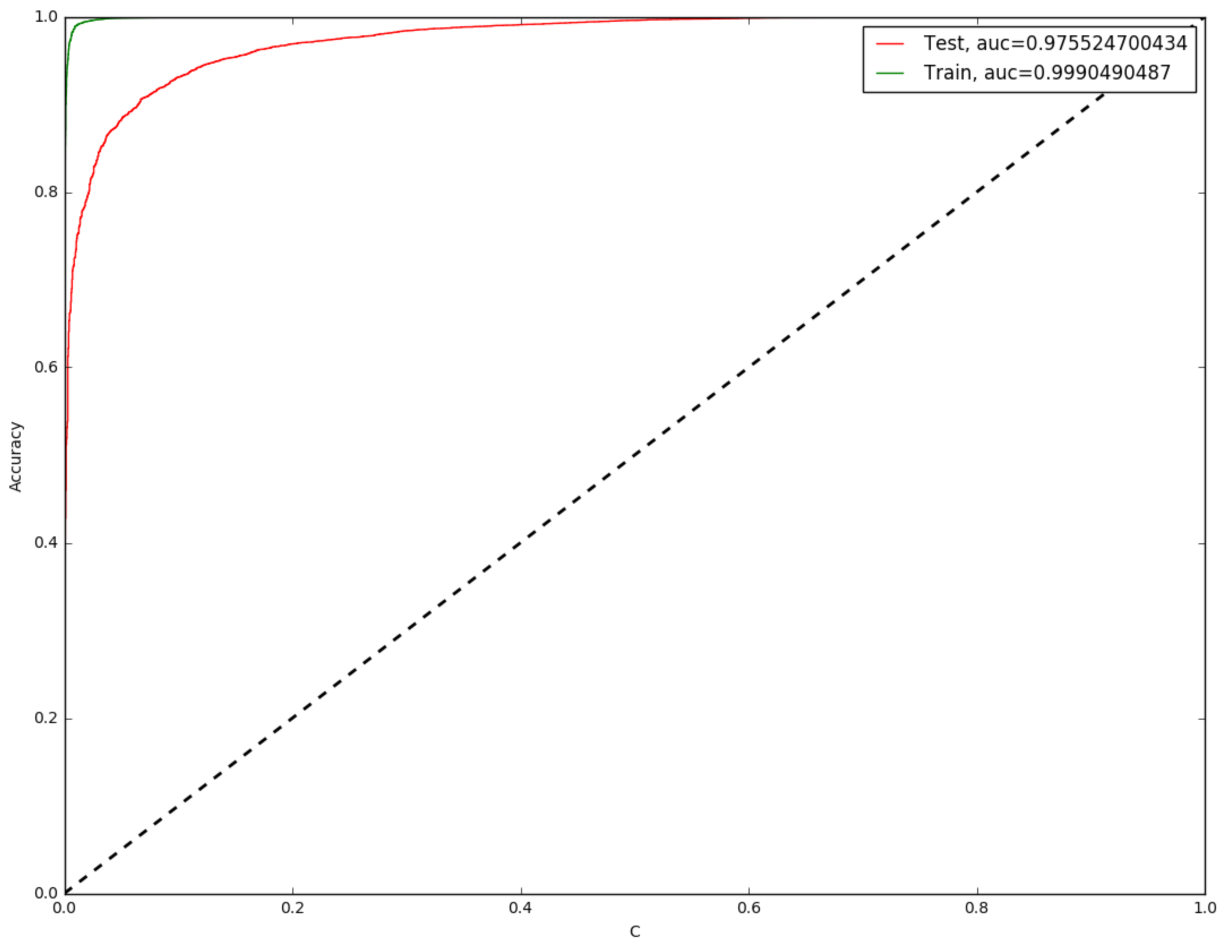
```
array([[  2.67883578e-04,   9.99732116e-01],
       [  9.44307107e-01,   5.56928926e-02],
       [  9.83593080e-03,   9.90164069e-01],
       ...,
       [  2.22213318e-01,   7.77786682e-01],
       [  5.25454493e-05,   9.99947455e-01],
       [  9.88128111e-01,   1.18718887e-02]])
```

In [270]:

```
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)
```

```
0.975524700434
0.9990490487
```

In [271]:
```
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),
plt.plot(tfidf_fpr_train, tfidf_tpr_train, label="Train, auc="+str(tfidf_train_au

plt.xlabel('C')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



In [272]:
```
tfidf_test_conf = tfidf_lgr.predict(tfidf_dict['test_tf_idf'])
```

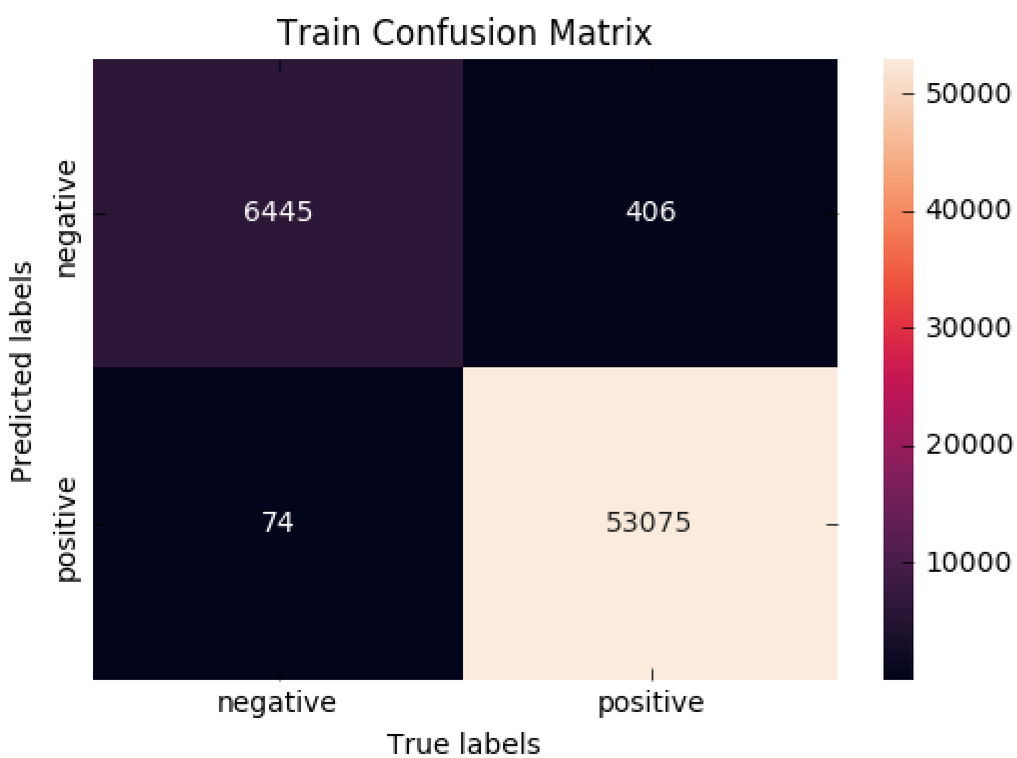
In [273]:
```
tfidf_train_conf = tfidf_lgr.predict(tfidf_dict['train_tf_idf'])
```

In [274]:
```
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)
```

```
[[ 1912   764]
 [  323 17001]]
             precision    recall  f1-score   support

          0       0.86      0.71      0.78      2676
          1       0.96      0.98      0.97     17324

avg / total       0.94      0.95      0.94     20000
```

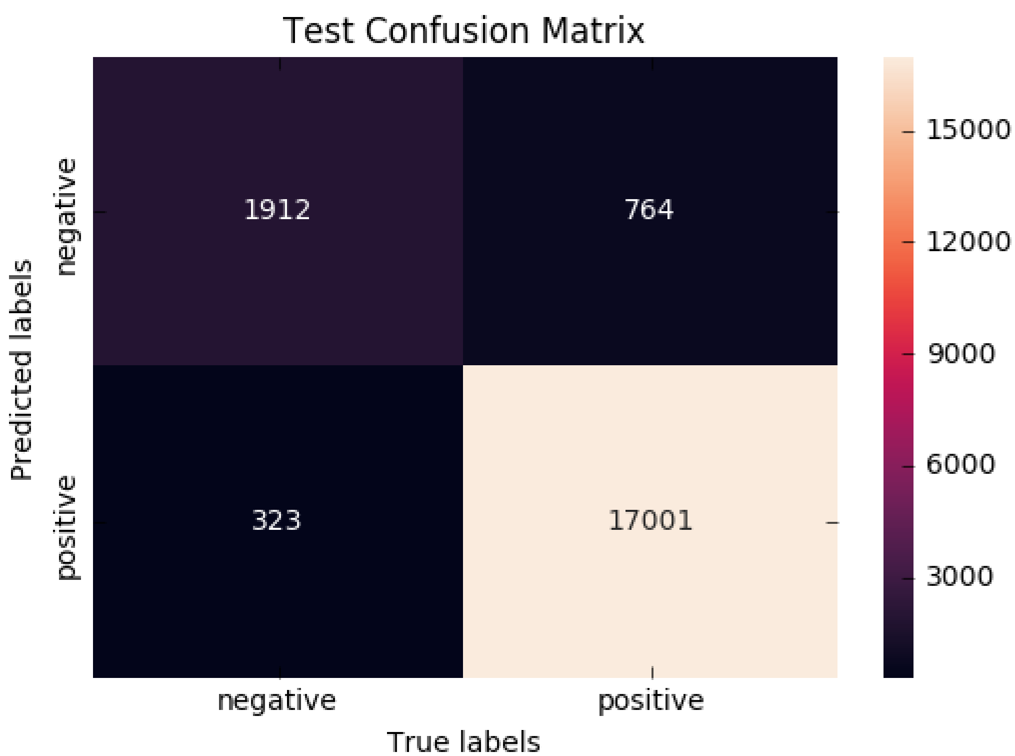
In [275]:
```
ax= plt.subplot()
sns.heatmap(tfidf_train_conf_matrix, annot=True, ax = ax, fmt='g')

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[275]: [<matplotlib.text.Text at 0x28d18828>, <matplotlib.text.Text at 0xb0b80cf8>]



In [276]:
```
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[276]: [<matplotlib.text.Text at 0x90b0a208>, <matplotlib.text.Text at 0x95110f98>]



Logistic Regression on Avg-tfidf

In [277]:
```
import pickle
with open(r"avg_w2v.pkl", "rb") as input_file:
    avg_tfidf_dict = pickle.load(input_file)
```

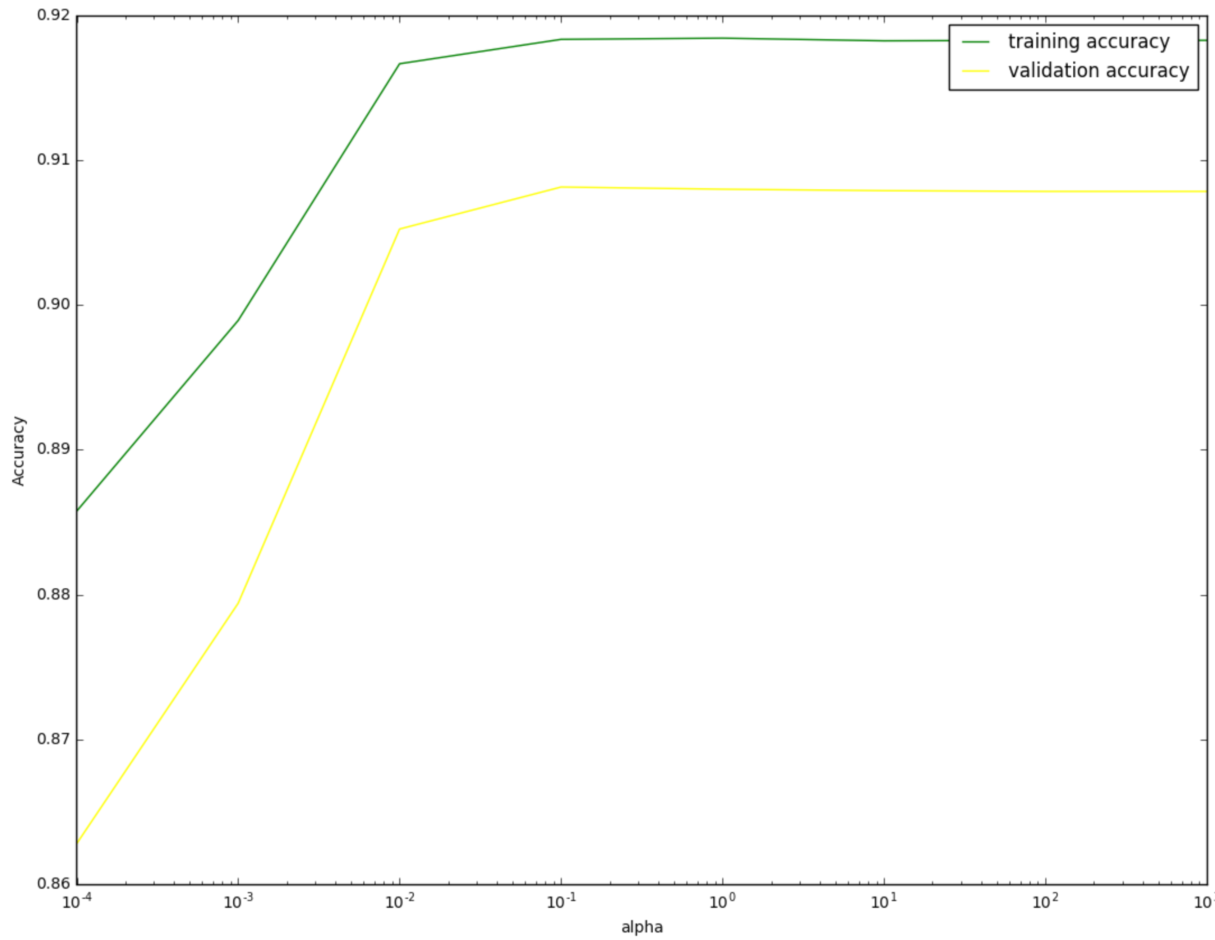
In [278]:
```
from sklearn.linear_model import LogisticRegression

avgtfidf_lgr_train_score_list = []
avgtfidf_lgr_val_score_list = []
for c_value in tqdm([0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]):
    avgtfidf_lgr=LogisticRegression(C=c_value)
    avgtfidf_lgr.fit(avg_tfidf_dict['X_train_avgw2v'],Y_train)

    avgtfidf_lgr_train_score = avgtfidf_lgr.score(avg_tfidf_dict['X_train_avgw2v'
    avgtfidf_lgr_train_score_list.append(avgtfidf_lgr_train_score)
    avgtfidf_lgr_val_score = avgtfidf_lgr.score(avg_tfidf_dict['X_val_avgw2v'], Y
    avgtfidf_lgr_val_score_list.append(avgtfidf_lgr_val_score)

c_all = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
avgtfidf_train_score = dict(zip(c_all, avgtfidf_lgr_train_score_list))
avgtfidf_val_score = dict(zip(c_all, avgtfidf_lgr_val_score_list))
print(avgtfidf_train_score)
print(avgtfidf_val_score)
```

```
100%|██████████████████████████████████████████████████████████████████
█████████████████████████████████████████████████████████████| 8/8 [00:07<0
0:00,  1.14it/s]

{0.1: 0.91835, 1: 0.9184333333333332, 100: 0.9182833333333334, 1000: 0.918283
33333333334, 0.0001: 0.8857666666666667, 10: 0.9182500000000001, 0.01: 0.91666
66666666663, 0.001: 0.8989333333333336}
{0.1: 0.9081500000000001, 1: 0.9080000000000003, 100: 0.9078500000000005, 10
00: 0.9078500000000005, 0.0001: 0.8628000000000001, 10: 0.9079000000000004,
0.01: 0.90525, 0.001: 0.8793999999999996}
```

In [279]:
```
best_c = max(avgtfidf_val_score, key=avgtfidf_val_score.get)
best_c
```

Out[279]:
```
0.1
```

In [280]:
```
import pylab
plt.figure(figsize=(13, 10))
neighbors_settings = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
plt.plot(neighbors_settings, avgtfidf_lgr_train_score_list, label="training accur
plt.plot(neighbors_settings, avgtfidf_lgr_val_score_list, label="validation accur
# plt.plot(neighbors_settings, auc_test, label="test accuracy", color='red')
plt.xlabel('alpha')
plt.ylabel('Accuracy')
plt.legend()
plt.xscale('log')

plt.show()
```



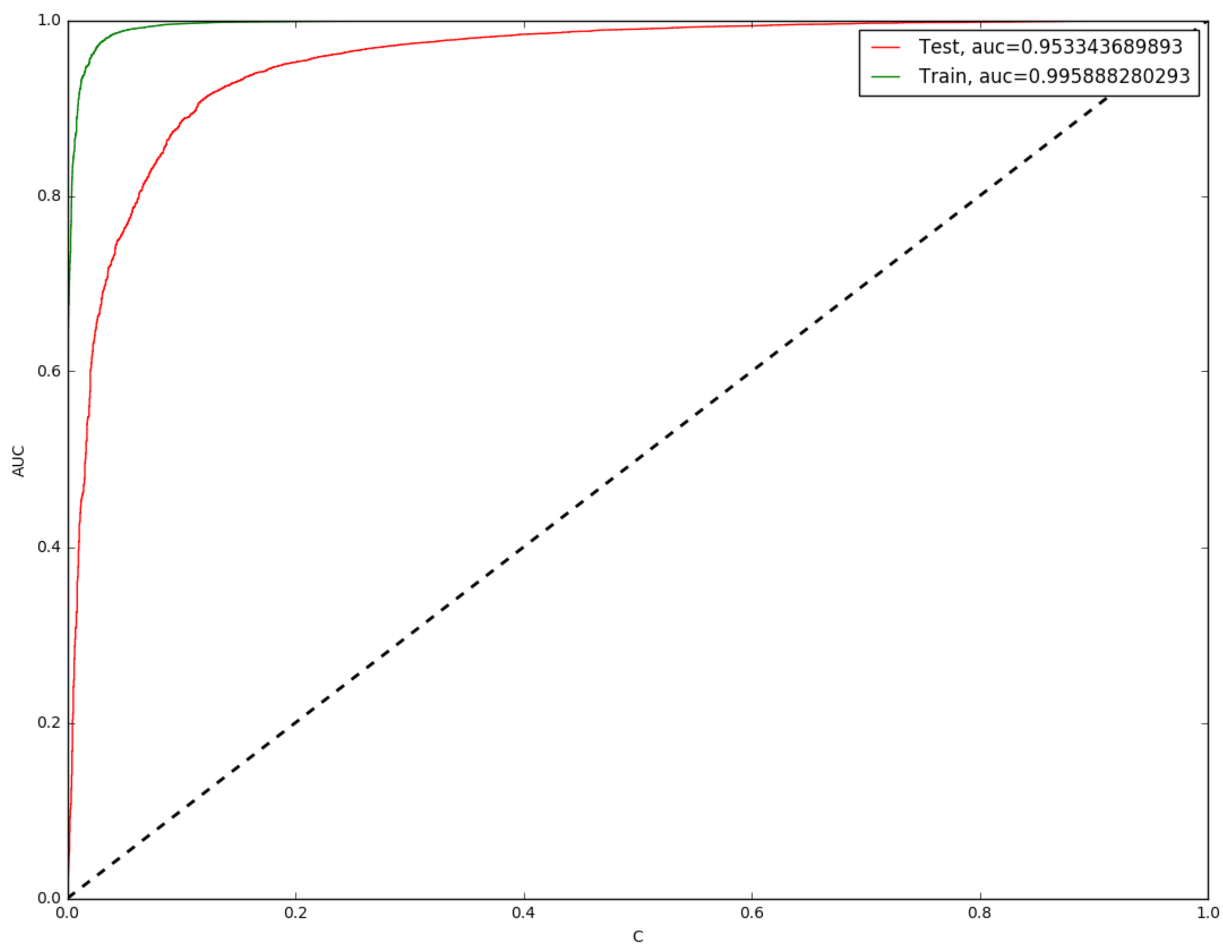
In [281]:
```
avgtfidf_lgr=LogisticRegression(C=best_c)
avgtfidf_lgr.fit(avg_tfidf_dict['X_train_avgw2v'],Y_train)
avgtfidf_test_proba = avgtfidf_lgr.predict_proba(avg_tfidf_dict['X_test_avgw2v'])
avgtfidf_train_proba = avgtfidf_lgr.predict_proba(avg_tfidf_dict['X_train_avgw2v'
avgtfidf_test_proba
```

Out[281]:
```
array([[  1.48373424e-03,   9.98516266e-01],
       [  9.15794492e-01,   8.42055077e-02],
       [  2.09498466e-02,   9.79050153e-01],
       ...,
       [  1.00856340e-01,   8.99143660e-01],
       [  5.68489727e-06,   9.99994315e-01],
       [  1.93096158e-01,   8.06903842e-01]])
```

In [282]:
```
avgtfidf_fpr_train, avgtfidf_tpr_train, _ = roc_curve(Y_train, bow_train_proba[:,
avgtfidf_fpr_test, avgtfidf_tpr_test, _ = roc_curve(Y_test, bow_test_proba[:, 1])
avgtfidf_test_auc = auc(avgtfidf_fpr_test, avgtfidf_tpr_test)
avgtfidf_train_auc = auc(avgtfidf_fpr_train, avgtfidf_tpr_train)
print(avgtfidf_test_auc)
print(avgtfidf_train_auc)
```

```
0.953343689893
0.995888280293
```

In [298]:
```
import pylab
plt.figure(figsize=(13, 10))
plt.plot([0,1], [0,1], color='black', lw=2, linestyle='--')
plt.plot(avgtfidf_fpr_test, avgtfidf_tpr_test, label="Test, auc="+str(avgtfidf_te
plt.plot(avgtfidf_fpr_train, avgtfidf_tpr_train, label="Train, auc="+str(avgtfidf

plt.xlabel('C')
plt.ylabel('AUC')
plt.legend()

plt.show()
```



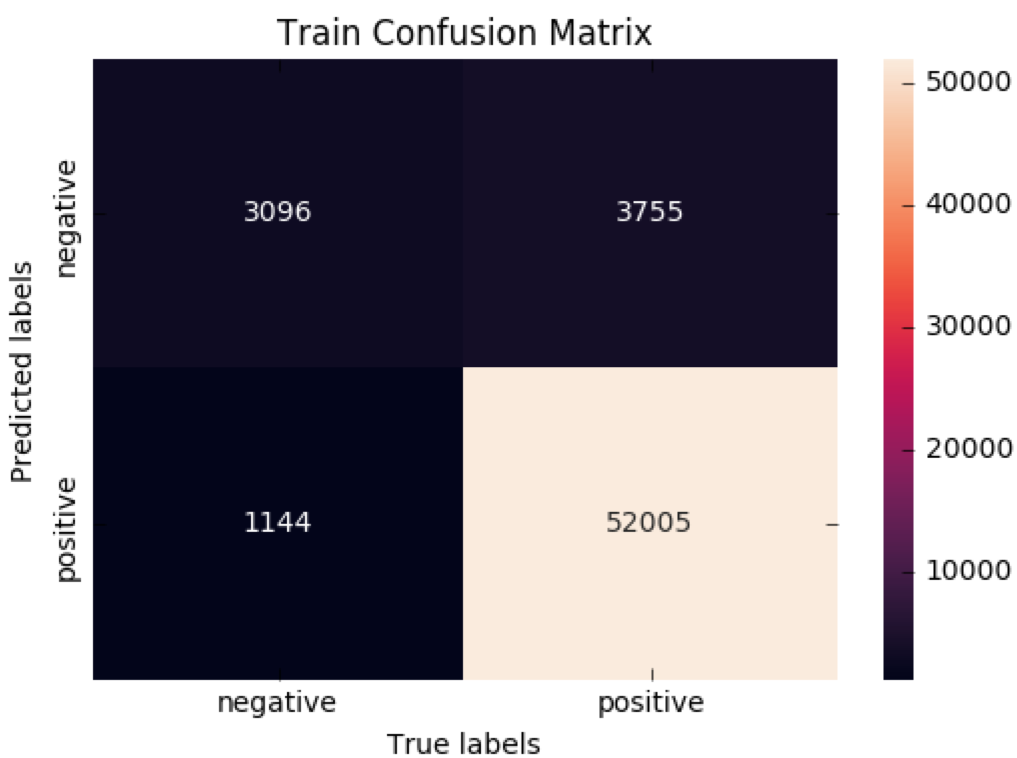
In [285]:
```
avg_test_conf = avgtfidf_lgr.predict(avg_tfidf_dict['X_test_avgw2v'])
avg_train_conf = avgtfidf_lgr.predict(avg_tfidf_dict['X_train_avgw2v'])
```

In [286]:
```
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)
```

```
[[ 1216  1460]
 [  433 16891]]
             precision    recall  f1-score   support

          0       0.74      0.45      0.56      2676
          1       0.92      0.98      0.95     17324

avg / total       0.90      0.91      0.90     20000
```

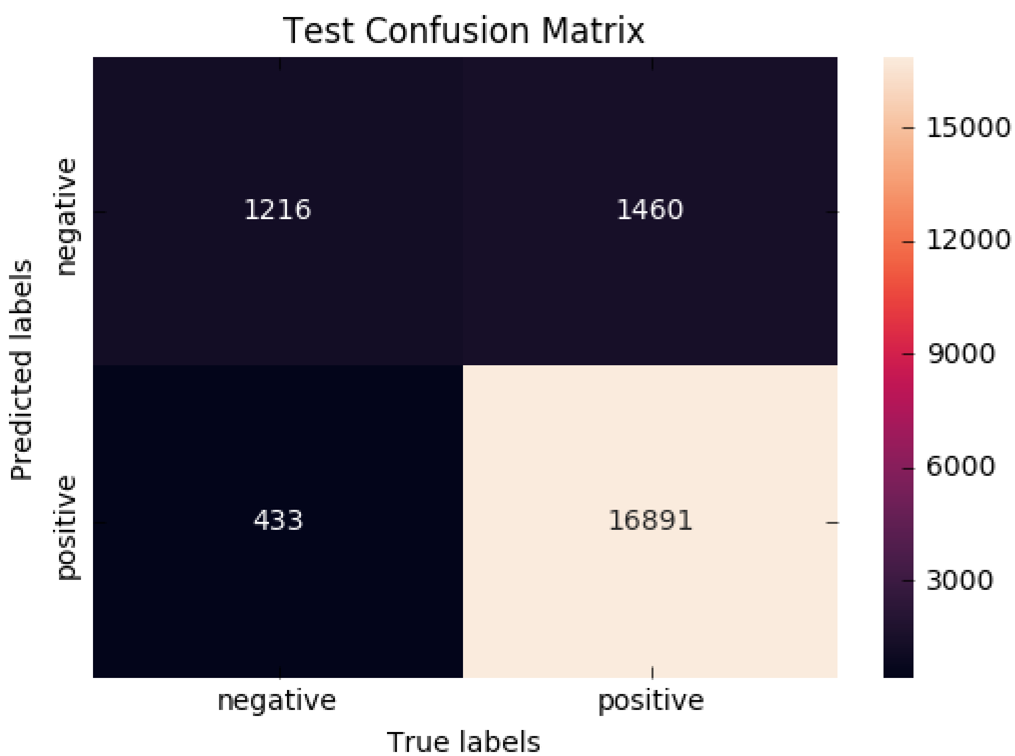
In [287]:
```
ax= plt.subplot()
sns.heatmap(avg_train_conf_matrix, annot=True, ax = ax, fmt='g')

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[287]: [<matplotlib.text.Text at 0x8a207710>, <matplotlib.text.Text at 0x4d968588>]



In [288]:
```
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[288]:
```
[<matplotlib.text.Text at 0x7ef7a780>, <matplotlib.text.Text at 0x828b8438>]
```



Logistic Regression on TFIDF weighted W2V

In [300]:
```
import pickle
with open(r"tfidf_w2v.pkl", "rb") as input_file:
    tfidfw2v_dict = pickle.load(input_file)
```

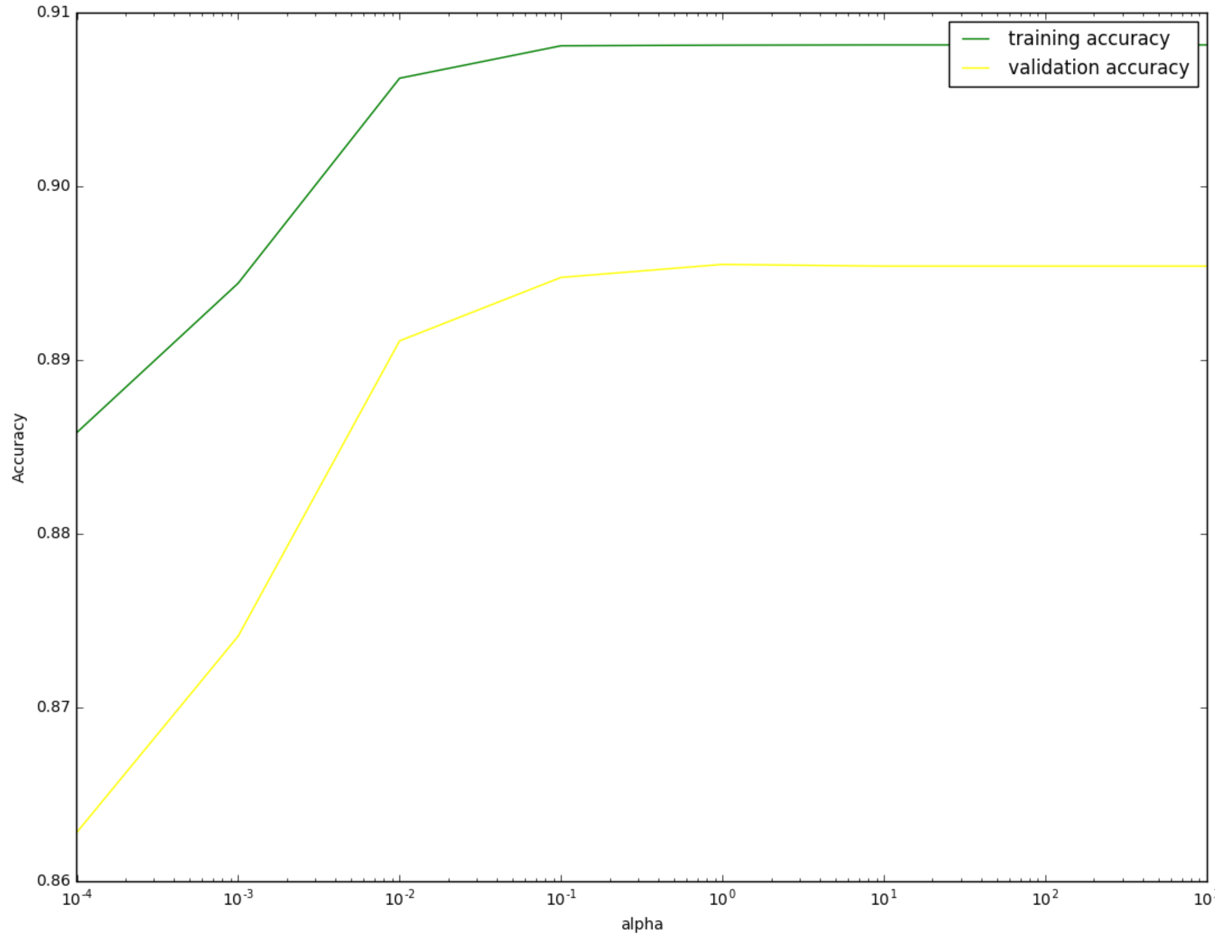
In [301]:
```
from sklearn.linear_model import LogisticRegression
from tqdm import tqdm

tfidfw2v_lgr_train_score_list = []
tfidfw2v_lgr_val_score_list = []
for c_value in tqdm([0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]):
    tfidfw2v_lgr=LogisticRegression(C=c_value)
    tfidfw2v_lgr.fit(tfidfw2v_dict['X_train_tfidfw2v'],Y_train)

    tfidfw2v_lgr_train_score = tfidfw2v_lgr.score(tfidfw2v_dict['X_train_tfidfw2v
    tfidfw2v_lgr_train_score_list.append(tfidfw2v_lgr_train_score)
    tfidfw2v_lgr_val_score = tfidfw2v_lgr.score(tfidfw2v_dict['X_val_tfidfw2v'], 
    tfidfw2v_lgr_val_score_list.append(tfidfw2v_lgr_val_score)

c_all = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
tfidfw2v_train_score = dict(zip(c_all, tfidfw2v_lgr_train_score_list))
tfidfw2v_val_score = dict(zip(c_all, tfidfw2v_lgr_val_score_list))
print(tfidfw2v_train_score)
print(tfidfw2v_val_score)
```

```
100%|████████████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████| 8/8 [00:07<0
0:00,  1.10it/s]

{0.1: 0.9080833333333335, 1: 0.9081166666666668, 100: 0.9081333333333335, 10
00: 0.9081333333333335, 0.0001: 0.8858166666666667, 10: 0.9081333333333335,
0.01: 0.9062166666666667, 0.001: 0.8944166666666664}
{0.1: 0.8947500000000005, 1: 0.8954999999999996, 100: 0.8953999999999997, 10
00: 0.8953999999999997, 0.0001: 0.8628000000000001, 10: 0.8953999999999997,
0.01: 0.8911, 0.001: 0.8740999999999999}
```

In [302]:
```
best_c = max(tfidfw2v_val_score, key=tfidfw2v_val_score.get)
best_c
```

Out[302]: 1

In [303]:
```
import pylab
plt.figure(figsize=(13, 10))
neighbors_settings = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
plt.plot(neighbors_settings, tfidfw2v_lgr_train_score_list, label="training accur
plt.plot(neighbors_settings, tfidfw2v_lgr_val_score_list, label="validation accur
# plt.plot(neighbors_settings, auc_test, label="test accuracy", color='red')
plt.xlabel('alpha')
plt.ylabel('Accuracy')
plt.legend()
plt.xscale('log')

plt.show()
```



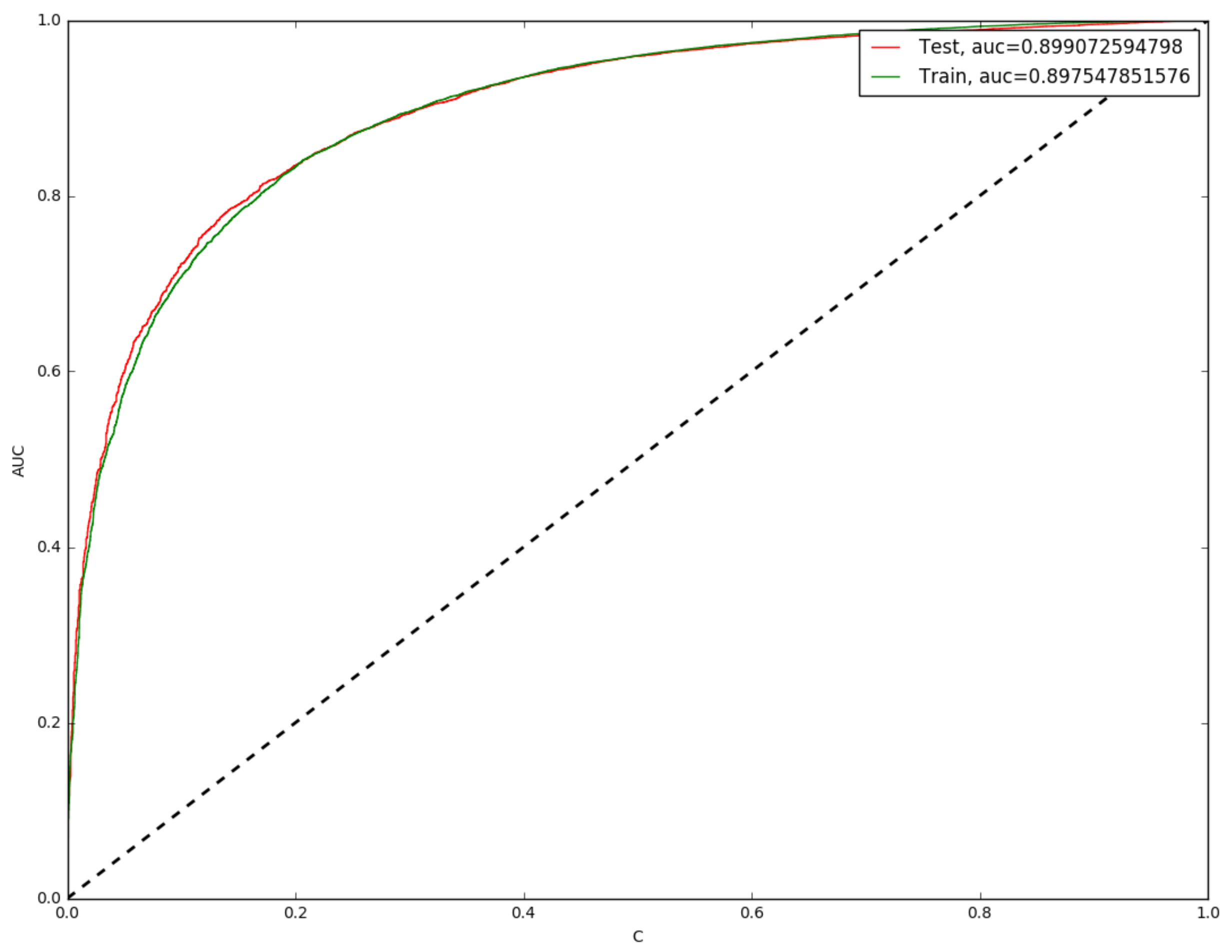
In [315]:
```
tfidfw2v_lgr=LogisticRegression(C=best_c)
tfidfw2v_lgr.fit(tfidfw2v_dict['X_train_tfidfw2v'],Y_train)
tfidfw2v_test_proba = tfidfw2v_lgr.predict_proba(tfidfw2v_dict['X_test_tfidfw2v']
tfidfw2v_train_proba = tfidfw2v_lgr.predict_proba(tfidfw2v_dict['X_train_tfidfw2v
tfidfw2v_test_proba
```

Out[315]:
```
array([[  2.90711453e-02,   9.70928855e-01],
       [  6.84203932e-01,   3.15796068e-01],
       [  1.77343933e-02,   9.82265607e-01],
       ...,
       [  1.40792404e-01,   8.59207596e-01],
       [  1.36527534e-04,   9.99863472e-01],
       [  2.27219536e-01,   7.72780464e-01]])
```

In [316]:
```
tfidfw2v_fpr_train, tfidfw2v_tpr_train, _ = roc_curve(Y_train, tfidfw2v_train_pro
tfidfw2v_fpr_test, tfidfw2v_tpr_test, _ = roc_curve(Y_test, tfidfw2v_test_proba[:
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)
```

```
0.899072594798
0.897547851576
```

In [317]:
```
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_te
plt.plot(tfidfw2v_fpr_train, tfidfw2v_tpr_train, label="Train, auc="+str(tfidfw2v_

plt.xlabel('C')
plt.ylabel('AUC')
plt.legend()

plt.show()
```



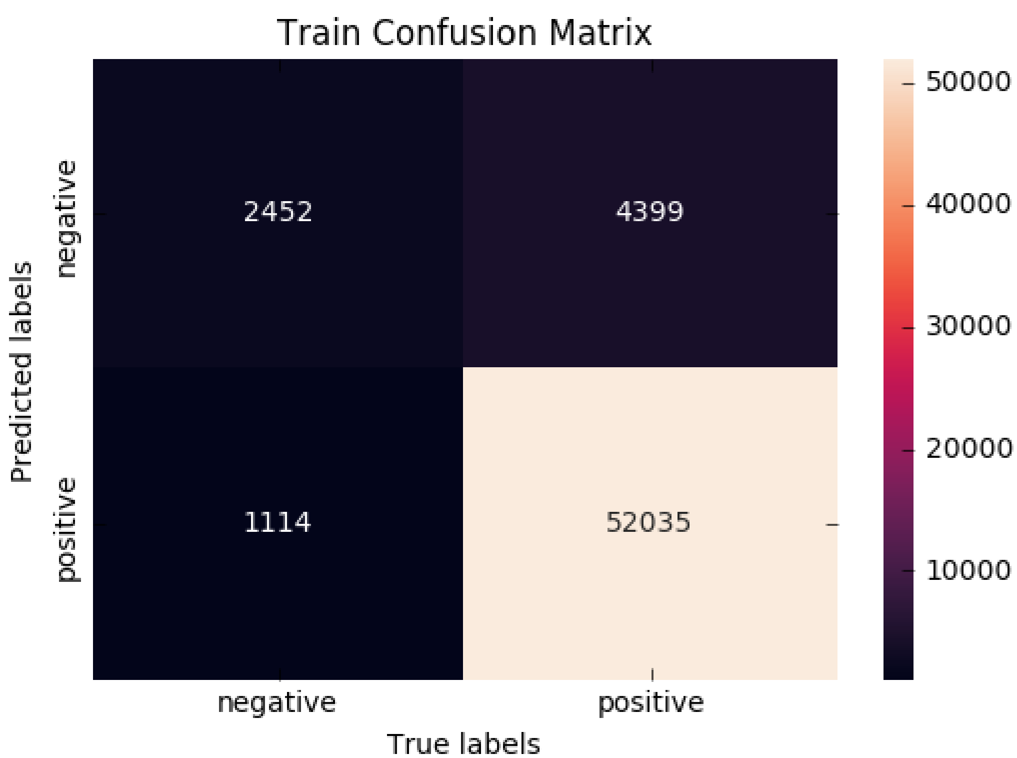
In [322]:
```
tfidfw2v_test_conf = tfidfw2v_lgr.predict(tfidfw2v_dict['X_test_tfidfw2v'])
tfidfw2v_train_conf = tfidfw2v_lgr.predict(tfidfw2v_dict['X_train_tfidfw2v'])
```

In [321]:
```
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)
```

```
[[ 2452  4399]
 [ 1114 52035]]
             precision    recall  f1-score   support

          0       0.71      0.37      0.48      2676
          1       0.91      0.98      0.94     17324

avg / total       0.88      0.90      0.88     20000
```

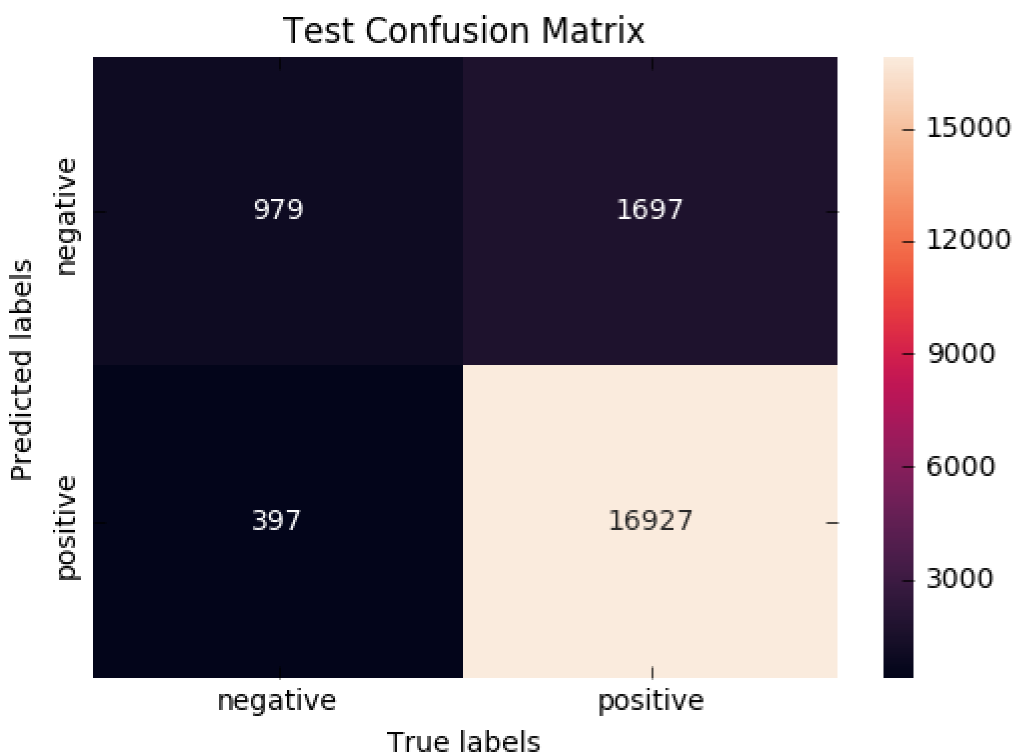
In [324]:
```
ax= plt.subplot()
sns.heatmap(tfidfw2v_train_conf_matrix, annot=True, ax = ax, fmt='g')

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[324]: [<matplotlib.text.Text at 0x82e40358>, <matplotlib.text.Text at 0x9671bef0>]



In [325]:
```
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[325]:
```
[<matplotlib.text.Text at 0x140943c8>, <matplotlib.text.Text at 0xa55ff048>]
```



In [2]:
```
from prettytable import PrettyTable

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

x.add_row(["BoW", 1, 0.995888280293, 0.953343689893])
x.add_row(["Tf-idf",10, 0.9990490487, 0.975524700434])
x.add_row(["Avg-w2v",0.1, 0.995888280293,0.953343689893])
x.add_row(["Tfidf-Avg_w2v",1, 0.897547851576, 0.899072594798])
print(x)
```

```
+---------------+-----+----------------+----------------+
|   Vectorizer  |  C  |     Train      |      Test      |
+---------------+-----+----------------+----------------+
|      BoW      |  1  | 0.995888280293 | 0.953343689893 |
|     Tf-idf    |  10 |  0.9990490487  | 0.975524700434 |
|    Avg-w2v    | 0.1 | 0.995888280293 | 0.953343689893 |
| Tfidf-Avg_w2v |  1  | 0.897547851576 | 0.899072594798 |
+---------------+-----+----------------+----------------+
```

Steps taken to increase accuracy:

i. Did feature engineering like appended summary and text column to preprocess text
ii. Considered number of words

Observations:
i. Accuracy is getting increased by around 2 % when feature engineering is done.