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
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

import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

1.1 Reading Data

Splitting data into Train, Cross Validation and Test

In [2]:

```
prepeocessed_data = pd.read_csv('preprocessed_data.csv')
prepeocessed_data.head(2)
```

Out[2]:

	Unnamed: 0	Unnamed: 0.1	id	teacher_id	teacher
0	0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	
1	1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	

2 rows × 21 columns

In [3]:

```
y = prepeocessed_data['project_is_approved'].values
X = prepeocessed_data.drop(['project_is_approved'], axis=1)
X.shape
```

Out[3]:

```
(109248, 20)
```

In [4]:

```
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, str
X_test.shape
```

Out[4]:

```
(36052, 20)
```

1.4 Encoding Categorical and Numerical features

1.4.1 encoding categorical features: clean_categories

In [5]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only o

# we use the fitted CountVectorizer to convert the text to vector
X_train_cc_ohe = vectorizer.transform(X_train['clean_categories'].values)
#X_cv_cc_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_cc_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_cc_ohe.shape, y_train.shape)
#print(X_cv_cc_ohe.shape, y_cv.shape)
print(X_test_cc_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
```

```
After vectorizations
(73196, 9) (73196,)
(36052, 9) (36052,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_c
ivics', 'literacy_language', 'math_science', 'music_arts', 'sp
ecialneeds', 'warmth']
```

1.4.2 encoding categorical features: clean_subcategories

In [6]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen onl

# we use the fitted CountVectorizer to convert the text to vector
X_train_csc_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
#X_cv_csc_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_csc_ohe = vectorizer.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_csc_ohe.shape, y_train.shape)
#print(X_cv_csc_ohe.shape, y_cv.shape)
print(X_test_csc_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
```

```
After vectorizations
(73196, 30) (73196,)
(36052, 30) (36052,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civi
cs_government', 'college_careerprep', 'communityservice', 'ear
lydevelopment', 'economics', 'environmentalscience', 'esl', 'e
xtracurricular', 'financialliteracy', 'foreignlanguages', 'gym
_fitness', 'health_lifescience', 'health_wellness', 'history_g
eography', 'literacy', 'literature_writing', 'mathematics', 'm
usic', 'nutritioneducation', 'other', 'parentinvolvement', 'pe
rformingarts', 'socialsciences', 'specialneeds', 'teamsports',
'visualarts', 'warmth']
```

1.4.3 encoding categorical features: school_state

In [7]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on tr

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values)
#X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
#print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
```

```
After vectorizations
(73196, 51) (73196,)
(36052, 51) (36052,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl',
'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'm
d', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'n
h', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 's
c', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'w
y']
```

1.4.4 encoding categorical features: teacher_prefix

In [8]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
#X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
#print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
```

```
After vectorizations
(73196, 5) (73196,)
(36052, 5) (36052,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
```

1.4.5 encoding categorical features: project_grade_category

In [9]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].va
#X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].valu

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
#print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
```

```
After vectorizations
(73196, 4) (73196,)
(36052, 4) (36052,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
```

1.4.6 encoding numerical features: price

In [10]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['price'].values.reshape(1,-1))

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(1,
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1)

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_test_price_norm.shape, y_test.shape)
print(X_train_price_norm.shape)
print(X_test_price_norm.shape)
```

```
After vectorizations
(1, 73196) (73196,)
(1, 36052) (36052,)
(1, 73196)
(1, 36052)
```

1.4.7 encoding numerical features: teacher_number_of_previously_posted_projects

In [11]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values

X_train_ppp_norm = normalizer.transform(X_train['teacher_number_of_previously
X_test_ppp_norm = normalizer.transform(X_test['teacher_number_of_previously_p

print("After vectorizations")
print(X_train_ppp_norm.shape, y_train.shape)
print(X_test_ppp_norm.shape, y_test.shape)
```

```
After vectorizations
(1, 73196) (73196,)
(1, 36052) (36052,)
```

1.4.8 encoding numerical features: quantity

In [12]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['quantity'].values.reshape(1,-1))

X_train_quantity_norm = normalizer.transform(X_train['quantity'].values.resha
X_test_quantity_norm = normalizer.transform(X_test['quantity'].values.reshape

print("After vectorizations")
print(X_train_quantity_norm.shape, y_train.shape)
print(X_test_quantity_norm.shape, y_test.shape)
```

```
After vectorizations
(1, 73196) (73196,)
(1, 36052) (36052,)
```

1.4.9 encoding numerical features: sentiment score's of each of the essay

In [13]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# import nltk
# nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()
ss_train = []
ss_test = []
for essay in X_train['essay']:
    ss_train.append(sid.polarity_scores(essay)['pos'])

for essay in X_test['essay']:
    ss_test.append(sid.polarity_scores(essay)['pos'])

# we can use these 4 things as features/attributes (neg, neu, pos, compound)
# neg: 0.0, neu: 0.753, pos: 0.247, compound: 0.93

print(len(ss_train))
print(len(ss_test))
print(ss_train[7])
print(ss_test[7])

ss_train_array = np.array(ss_train)
ss_test_array = np.array(ss_test)
print(ss_train_array.shape)
print(ss_test_array.shape)
```

```
73196
36052
0.296
0.172
(73196,)
(36052,)
```

In [14]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(ss_train_array.reshape(1,-1))

X_train_ss_norm = normalizer.transform(ss_train_array.reshape(1,-1))
X_test_ss_norm = normalizer.transform(ss_test_array.reshape(1,-1))

print("After vectorizations")
print(X_train_ss_norm.shape, y_train.shape)
print(X_test_ss_norm.shape, y_test.shape)
```

```
After vectorizations
(1, 73196) (73196,)
(1, 36052) (36052,)
```

1.4.10 encoding numerical features: number of words in the title

In [15]:

```
title_word_count_train = []
title_word_count_test = []


for i in X_train['project_title']:
    title_word_count_train.append(len(i.split()))

for i in X_test['project_title']:
    title_word_count_test.append(len(i.split()))

print(len(title_word_count_train))
print(len(title_word_count_test))
print(title_word_count_train[7])
print(title_word_count_train[7])

title_word_count_train_array = np.array(title_word_count_train)
title_word_count_test_array = np.array(title_word_count_test)
print(title_word_count_train_array.shape)
print(title_word_count_test_array.shape)
```

```
73196
36052
8
8
(73196,)
(36052,)
```

In [16]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(title_word_count_train_array.reshape(1,-1))

X_train_twc_norm = normalizer.transform(title_word_count_train_array.reshape(
X_test_twc_norm = normalizer.transform(title_word_count_test_array.reshape(1,

print("After vectorizations")
print(X_train_twc_norm.shape, y_train.shape)
print(X_test_twc_norm.shape, y_test.shape)
```

```
After vectorizations
(1, 73196) (73196,)
(1, 36052) (36052,)
```

1.4.11 encoding numerical features: number of words in the combine essays

In [17]:

```
essay_word_count_train = []
essay_word_count_test = []
for i in X_train['essay']:
    essay_word_count_train.append(len(i.split()))


for i in X_test['essay']:
    essay_word_count_test.append(len(i.split()))

print(len(essay_word_count_train))
print(len(essay_word_count_test))
print(essay_word_count_train[7])
print(essay_word_count_test[7])


essay_word_count_train_array = np.array(essay_word_count_train)
essay_word_count_test_array = np.array(essay_word_count_test)
print(essay_word_count_train_array.shape)
print(essay_word_count_test_array.shape)
```

```
73196
36052
215
245
(73196,)
(36052,)
```

In [18]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(essay_word_count_train_array.reshape(1,-1))

X_train_ewc_norm = normalizer.transform(essay_word_count_train_array.reshape(
X_test_ewc_norm = normalizer.transform(essay_word_count_test_array.reshape(1,

print("After vectorizations")
print(X_train_ewc_norm.shape, y_train.shape)
print(X_test_ewc_norm.shape, y_test.shape)
```

```
After vectorizations
(1, 73196) (73196,)
(1, 36052) (36052,)
```

1.5 Vectorizing Text features

1.5.1 Vectorizing using BOW

Essay

In [19]:

```
print(X_train.shape, y_train.shape)
#print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("\n\n")


vectorizer = CountVectorizer(ngram_range=(2,2), min_df=10, max_features=5000)
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train dat

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['essay'].values)
#X_cv_essay_bow = vectorizer.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
#print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
```

```
(73196, 20) (73196,)
(36052, 20) (36052,)



After vectorizations
(73196, 5000) (73196,)
(36052, 5000) (36052,)
```

project_title

In [20]:

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['project_title'].values) # fit has to happen only on t

# we use the fitted CountVectorizer to convert the text to vector
X_train_titles_bow = vectorizer.transform(X_train['project_title'].values)
#X_cv_titles_bow = vectorizer.transform(X_cv['project_title'].values)
X_test_titles_bow = vectorizer.transform(X_test['project_title'].values)

print("After vectorizations")
print(X_train_titles_bow.shape, y_train.shape)
#print(X_cv_titles_bow.shape, y_cv.shape)
print(X_test_titles_bow.shape, y_test.shape)
```

```
After vectorizations
(73196, 5000) (73196,)
(36052, 5000) (36052,)
```

project_resource_summary

In [21]:

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['project_resource_summary'].values) # fit has to happe

# we use the fitted CountVectorizer to convert the text to vector
X_train_psr_bow = vectorizer.transform(X_train['project_resource_summary'].va
#X_cv_psr_bow = vectorizer.transform(X_cv['project_resource_summary'].values)
X_test_psr_bow = vectorizer.transform(X_test['project_resource_summary'].valu

print("After vectorizations")
print(X_train_psr_bow.shape, y_train.shape)
#print(X_cv_psr_bow.shape, y_cv.shape)
print(X_test_psr_bow.shape, y_test.shape)
```

```
After vectorizations
(73196, 5000) (73196,)
(36052, 5000) (36052,)
```

1.5.2 Vectorizing using TFIDF

essay

In [22]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = CountVectorizer(ngram_range=(2,2), min_df=10, max_features=5000)
vectorizer.fit(X_train['essay'].values)

X_train_essay_tfidf = vectorizer.transform(X_train['essay'].values)
#X_cv_essay_tfidf = vectorizer.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
#print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
```

```
After vectorizations
(73196, 5000) (73196,)
(36052, 5000) (36052,)
```

project_title

In [23]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['project_title'].values)

X_train_titles_tfidf = vectorizer.transform(X_train['project_title'].values)
#X_cv_titles_tfidf = vectorizer.transform(X_cv['project_title'].values)
X_test_titles_tfidf = vectorizer.transform(X_test['project_title'].values)

print("After vectorizations")
print(X_train_titles_tfidf.shape, y_train.shape)
#print(X_cv_titles_tfidf.shape, y_cv.shape)
print(X_test_titles_tfidf.shape, y_test.shape)
```

```
After vectorizations
(73196, 5000) (73196,)
(36052, 5000) (36052,)
```

project_resource_summary

In [24]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['project_resource_summary'].values)

X_train_prs_tfidf = vectorizer.transform(X_train['project_resource_summary'].
#X_cv_prs_tfidf = vectorizer.transform(X_cv['project_resource_summary'].value
X_test_prs_tfidf = vectorizer.transform(X_test['project_resource_summary'].va

print("After vectorizations")
print(X_train_prs_tfidf.shape, y_train.shape)
#print(X_cv_prs_tfidf.shape, y_cv.shape)
print(X_test_prs_tfidf.shape, y_test.shape)
```

```
After vectorizations
(73196, 5000) (73196,)
(36052, 5000) (36052,)
```

1.5.3 Vectorizing using AVG W2V

In [25]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/ho
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

essay

In [26]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_essay_train = []; # the avg-w2v for each sentence/review is stored ir
for sentence in tqdm(X_train['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essay_train.append(vector)

print(len(avg_w2v_essay_train))
print(len(avg_w2v_essay_train[0]))
```

```
100%|████████████████████████████████████████████████████████████
████████████████| 73196/73196 [00:28<00:00, 2528.37it/s]

73196
300
```

In [27]:

```
'''avg_w2v_essay_cv = []; # the avg-w2v for each sentence/review is stored ir
for sentence in tqdm(X_cv['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essay_cv.append(vector)'''
```

Out[27]:

```
"avg_w2v_essay_cv = []; # the avg-w2v for each sentence/review
is stored in this list\nfor sentence in tqdm(X_cv['essay'].val
ues): # for each review/sentence\n    vector = np.zeros(300) #
as word vectors are of zero length\n    cnt_words =0; # num of
words with a valid vector in the sentence/review\n    for word
in sentence.split(): # for each word in a review/sentence\n
if word in glove_words:\n            vector += model[word]\n
cnt_words += 1\n    if cnt_words != 0:\n        vector /= cnt_
words\n    avg_w2v_essay_cv.append(vector)"
```

In [28]:

```
avg_w2v_essay_test = []; # the avg-w2v for each sentence/review is stored in
for sentence in tqdm(X_test['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essay_test.append(vector)
```

```
100%|████████████████████████████████████████████████████████████
████████████████| 36052/36052 [00:14<00:00, 2530.70it/s]
```

project_title

In [29]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_titles_train = []; # the avg-w2v for each sentence/review is stored i
for sentence in tqdm(X_train['project_title'].values): # for each review/sent
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_titles_train.append(vector)

print(len(avg_w2v_titles_train))
print(len(avg_w2v_titles_train[0]))
```

```
100%|████████████████████████████████████████████████████████████
████████████████| 73196/73196 [00:00<00:00, 126991.82it/s]

73196
300
```

In [30]:

```
'''
avg_w2v_titles_cv = []; # the avg-w2v for each sentence/review is stored in t
for sentence in tqdm(X_cv['project_title'].values): # for each review/sentenc
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_titles_cv.append(vector)

'''
```

Out[30]:

```
"\navg_w2v_titles_cv = []; # the avg-w2v for each sentence/rev
iew is stored in this list\nfor sentence in tqdm(X_cv['project
_title'].values): # for each review/sentence\n    vector = np.
zeros(300) # as word vectors are of zero length\n    cnt_words
=0; # num of words with a valid vector in the sentence/review
\n    for word in sentence.split(): # for each word in a revie
w/sentence\n        if word in glove_words:\n            vecto
r += model[word]\n            cnt_words += 1\n    if cnt_words
!= 0:\n        vector /= cnt_words\n    avg_w2v_titles_cv.appe
nd(vector)\n    \n"
```

In [31]:

```
avg_w2v_titles_test = []; # the avg-w2v for each sentence/review is stored ir
for sentence in tqdm(X_test['project_title'].values): # for each review/sente
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_titles_test.append(vector)
```

```
100%|████████████████████████████████████████████████████████████
████████████| 36052/36052 [00:00<00:00, 126844.56it/s]
```

project_resource_summary

In [32]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_prs_train = []; # the avg-w2v for each sentence/review is stored in t
for sentence in tqdm(X_train['project_resource_summary'].values): # for each
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_prs_train.append(vector)

print(len(avg_w2v_prs_train))
print(len(avg_w2v_prs_train[0]))
```

```
100%|██████████████████████████████████████████████████████████
██████████████| 73196/73196 [00:02<00:00, 27022.07it/s]

73196
300
```

In [33]:

```
'''
avg_w2v_prs_cv = []; # the avg-w2v for each sentence/review is stored in this
for sentence in tqdm(X_cv['project_resource_summary'].values): # for each rev
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_prs_cv.append(vector)

'''
```

Out[33]:

```
"\navg_w2v_prs_cv = []; # the avg-w2v for each sentence/review
is stored in this list\nfor sentence in tqdm(X_cv['project_res
ource_summary'].values): # for each review/sentence\n    vecto
r = np.zeros(300) # as word vectors are of zero length\n    cn
t_words =0; # num of words with a valid vector in the sentenc
e/review\n    for word in sentence.split(): # for each word in
a review/sentence\n        if word in glove_words:\n
vector += model[word]\n            cnt_words += 1\n    if cnt_
words != 0:\n        vector /= cnt_words\n    avg_w2v_prs_cv.a
ppend(vector)\n    \n"
```

In [34]:

```
avg_w2v_prs_test = []; # the avg-w2v for each sentence/review is stored in th
for sentence in tqdm(X_test['project_resource_summary'].values): # for each r
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_prs_test.append(vector)
```

```
100%|████████████████████████████████████████████████████████████
████████████████| 36052/36052 [00:01<00:00, 27414.00it/s]
```

1.5.4 Vectorizing using TFIDF W2V

project_title

In [35]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['project_title'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)
tfidf_words = set(tfidf_model.get_feature_names())
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_title_train = []; # the avg-w2v for each sentence/review is stored
for sentence in tqdm(X_train['project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/revi
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spli
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_title_train.append(vector)

print(len(tfidf_w2v_title_train))
print(len(tfidf_w2v_title_train[0]))
```

```
100%|██████████████████████████████████████████████████████████████
██████████████| 73196/73196 [00:00<00:00, 87717.43it/s]

73196
300
```

In [36]:

```
'''tfidf_w2v_title_cv = []; # the avg-w2v for each sentence/review is stored
for sentence in tqdm(X_cv['project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/revi
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spli
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_title_cv.append(vector)

print(len(tfidf_w2v_title_cv))
print(len(tfidf_w2v_title_cv[0]))'''
```

Out[36]:

```
"tfidf_w2v_title_cv = []; # the avg-w2v for each sentence/revi
ew is stored in this list\nfor sentence in tqdm(X_cv['project_
title']): # for each review/sentence\n    vector = np.zeros(30
0) # as word vectors are of zero length\n    tf_idf_weight =0;
# num of words with a valid vector in the sentence/review\n
for word in sentence.split(): # for each word in a review/sent
ence\n        if (word in glove_words) and (word in tfidf_word
s):\n            vec = model[word] # getting the vector for ea
ch word\n            # here we are multiplying idf value(dicti
onary[word]) and the tf value((sentence.count(word)/len(senten
ce.split())))\n            tf_idf = dictionary[word]*(sentenc
e.count(word)/len(sentence.split())) # getting the tfidf value
for each word\n            vector += (vec * tf_idf) # calculat
ing tfidf weighted w2v\n            tf_idf_weight += tf_idf\n
if tf_idf_weight != 0:\n        vector /= tf_idf_weight\n    t
fidf_w2v_title_cv.append(vector)\n\nprint(len(tfidf_w2v_title_
cv))\nprint(len(tfidf_w2v_title_cv[0]))"
```

In [37]:

```
tfidf_w2v_title_test = []; # the avg-w2v for each sentence/review is stored i
for sentence in tqdm(X_test['project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/revi
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spli
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_title_test.append(vector)

print(len(tfidf_w2v_title_test))
print(len(tfidf_w2v_title_test[0]))
```

```
100%|██████████████████████████████████████████████████████████
██████████████| 36052/36052 [00:00<00:00, 85895.47it/s]

36052
300
```

essay

In [38]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['essay'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)
tfidf_words = set(tfidf_model.get_feature_names())
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_essay_train = []; # the avg-w2v for each sentence/review is stored
for sentence in tqdm(X_train['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/revi
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spli
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_essay_train.append(vector)

print(len(tfidf_w2v_essay_train))
print(len(tfidf_w2v_essay_train[0]))
```

```
100%|████████████████████████████████████████████████████████████
████████████████| 73196/73196 [05:12<00:00, 234.21it/s]

73196
300
```

In [39]:

```
'''tfidf_w2v_essay_cv = []; # the avg-w2v for each sentence/review is stored
for sentence in tqdm(X_cv['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/revi
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spli
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_essay_cv.append(vector)

print(len(tfidf_w2v_essay_cv))
print(len(tfidf_w2v_essay_cv[0]))'''
```

Out[39]:

```
"tfidf_w2v_essay_cv = []; # the avg-w2v for each sentence/revi
ew is stored in this list\nfor sentence in tqdm(X_cv['essa
y']): # for each review/sentence\n    vector = np.zeros(300) #
as word vectors are of zero length\n    tf_idf_weight =0; # nu
m of words with a valid vector in the sentence/review\n    for
word in sentence.split(): # for each word in a review/sentence
\n        if (word in glove_words) and (word in tfidf_word
s):\n            vec = model[word] # getting the vector for ea
ch word\n            # here we are multiplying idf value(dicti
onary[word]) and the tf value((sentence.count(word)/len(senten
ce.split())))\n            tf_idf = dictionary[word]*(sentenc
e.count(word)/len(sentence.split())) # getting the tfidf value
for each word\n            vector += (vec * tf_idf) # calculat
ing tfidf weighted w2v\n            tf_idf_weight += tf_idf\n
if tf_idf_weight != 0:\n        vector /= tf_idf_weight\n    t
fidf_w2v_essay_cv.append(vector)\n\nprint(len(tfidf_w2v_essay_
cv))\nprint(len(tfidf_w2v_essay_cv[0]))"
```

In [40]:

```
tfidf_w2v_essay_test = []; # the avg-w2v for each sentence/review is stored i
for sentence in tqdm(X_test['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/revi
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spli
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_essay_test.append(vector)

print(len(tfidf_w2v_essay_test))
print(len(tfidf_w2v_essay_test[0]))
```

```
100%|██████████████████████████████████████████████████████████
██████████████████| 36052/36052 [02:39<00:00, 225.66it/s]

36052
300
```

project_resource_summary

In [41]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['project_resource_summary'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)
tfidf_words = set(tfidf_model.get_feature_names())
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_prs_train = []; # the avg-w2v for each sentence/review is stored in
for sentence in tqdm(X_train['project_resource_summary']): # for each review/
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/revi
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spli
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_prs_train.append(vector)

print(len(tfidf_w2v_prs_train))
print(len(tfidf_w2v_prs_train[0]))
```

```
100%|██████████████████████████████████████████████████████████████
████████████████| 73196/73196 [00:07<00:00, 9266.27it/s]

73196
300
```

In [42]:

```
'''tfidf_w2v_prs_cv = []; # the avg-w2v for each sentence/review is stored ir
for sentence in tqdm(X_cv['project_resource_summary']): # for each review/ser
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/revi
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spli
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_prs_cv.append(vector)

print(len(tfidf_w2v_prs_cv))
print(len(tfidf_w2v_prs_cv[0]))'''
```

Out[42]:

```
"tfidf_w2v_prs_cv = []; # the avg-w2v for each sentence/review
is stored in this list\nfor sentence in tqdm(X_cv['project_res
ource_summary']): # for each review/sentence\n    vector = np.
zeros(300) # as word vectors are of zero length\n    tf_idf_we
ight =0; # num of words with a valid vector in the sentence/re
view\n    for word in sentence.split(): # for each word in a r
eview/sentence\n        if (word in glove_words) and (word in
tfidf_words):\n            vec = model[word] # getting the vec
tor for each word\n            # here we are multiplying idf v
alue(dictionary[word]) and the tf value((sentence.count(word)/
len(sentence.split())))\n            tf_idf = dictionary[word]
*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word\n            vector += (vec * tf_idf)
# calculating tfidf weighted w2v\n            tf_idf_weight +=
tf_idf\n    if tf_idf_weight != 0:\n        vector /= tf_idf_w
eight\n    tfidf_w2v_prs_cv.append(vector)\n\nprint(len(tfidf_
w2v_prs_cv))\nprint(len(tfidf_w2v_prs_cv[0]))"
```

In [43]:

```
tfidf_w2v_prs_test = []; # the avg-w2v for each sentence/review is stored in
for sentence in tqdm(X_test['project_resource_summary']): # for each review/s
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/revi
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spli
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_prs_test.append(vector)

print(len(tfidf_w2v_prs_test))
print(len(tfidf_w2v_prs_test[0]))
```

```
100%|██████████████████████████████████████████████████████████
██████████████████| 36052/36052 [00:03<00:00, 9357.93it/s]

36052
300
```

Merging all the categorical and numerical features with variations of text features

In [44]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_train_bow_matrix = hstack((X_train_cc_ohe, X_train_csc_ohe, X_train_grade_c
                             X_train_price_norm.reshape(-1,1), X_train_ppp_nc
                             X_train_essay_bow, X_train_titles_bow, X_train_p


X_test_bow_matrix = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_grade_ohe,
                            X_test_price_norm.reshape(-1,1), X_test_ppp_norm
                            X_test_essay_bow, X_test_titles_bow, X_test_psr_b

print("Final Data matrix")
print(X_train_bow_matrix.shape, y_train.shape)
print(X_test_bow_matrix.shape, y_test.shape)
```

```
Final Data matrix
(73196, 15101) (73196,)
(36052, 15101) (36052,)
```

In [45]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_train_tfidf_matrix = hstack((X_train_cc_ohe, X_train_csc_ohe, X_train_grade
                               X_train_price_norm.reshape(-1,1), X_train_ppp_
                              X_train_titles_tfidf, X_train_essay_tfidf, X_tra

X_test_tfidf_matrix = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_grade_ohe
                              X_test_price_norm.reshape(-1,1), X_test_ppp_nor
                              X_test_titles_tfidf, X_test_essay_tfidf, X_test

print("Final Data matrix")
print(X_train_tfidf_matrix.shape, y_train.shape)
#print(X_cv_tfidf_matrix.shape, y_cv.shape)
print(X_test_tfidf_matrix.shape, y_test.shape)
```

```
Final Data matrix
(73196, 15101) (73196,)
(36052, 15101) (36052,)
```

In [46]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_train_aw2v_matrix = hstack((X_train_cc_ohe, X_train_csc_ohe, X_train_grade_
                              X_train_price_norm.reshape(-1,1), X_train_ppp_r
                             avg_w2v_essay_train, avg_w2v_titles_train, avg_w

X_test_aw2v_matrix = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_grade_ohe_
                            X_test_price_norm.reshape(-1,1), X_test_ppp_norm_
                             avg_w2v_essay_test, avg_w2v_titles_test, avg_w2v

print("Final Data matrix")
print(X_train_aw2v_matrix.shape, y_train.shape)
print(X_test_aw2v_matrix.shape, y_test.shape)
```

```
Final Data matrix
(73196, 1001) (73196,)
(36052, 1001) (36052,)
```

In [47]:

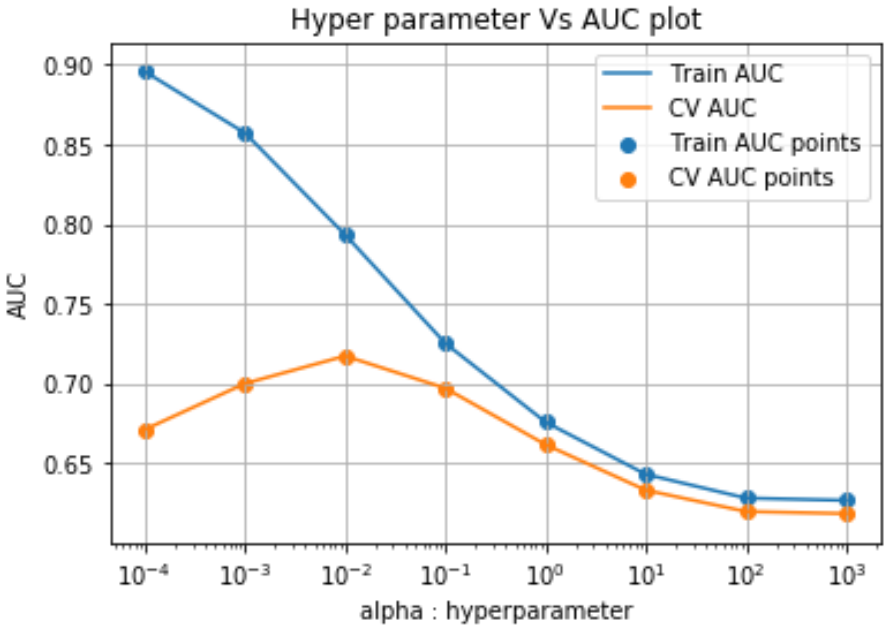
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_train_tw2v_matrix = hstack((X_train_cc_ohe, X_train_csc_ohe, X_train_grade_
                              X_train_price_norm.reshape(-1,1), X_train_ppp_r
                             tfidf_w2v_essay_train, tfidf_w2v_title_train, tf

X_test_tw2v_matrix = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_grade_ohe_
                             X_test_price_norm.reshape(-1,1), X_test_ppp_norm
                             tfidf_w2v_essay_test, tfidf_w2v_title_test, tfid

print("Final Data matrix")
print(X_train_tw2v_matrix.shape, y_train.shape)
#print(X_cv_tw2v_matrix.shape, y_cv.shape)
print(X_test_tw2v_matrix.shape, y_test.shape)
```

```
Final Data matrix
(73196, 1001) (73196,)
(36052, 1001) (36052,)
```

Finding Best Hyper parameter using K-Fold CV on BOW representation of text features

In [48]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.G
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.metrics import roc_auc_score

parameters = {'alpha':[0.0001,0.001,0.01,0.1,1,10,10**2,10**3]}
lg = linear_model.SGDClassifier(loss='log', class_weight =  "balanced")
clf = RandomizedSearchCV(lg, parameters, cv=10, scoring='roc_auc', return_tra
clf.fit(X_train_bow_matrix, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_alpha'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
K =  results['param_alpha']

plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_s

plt.plot(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("alpha : hyperparameter")
plt.xscale('log')
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results
```



Out[48]:

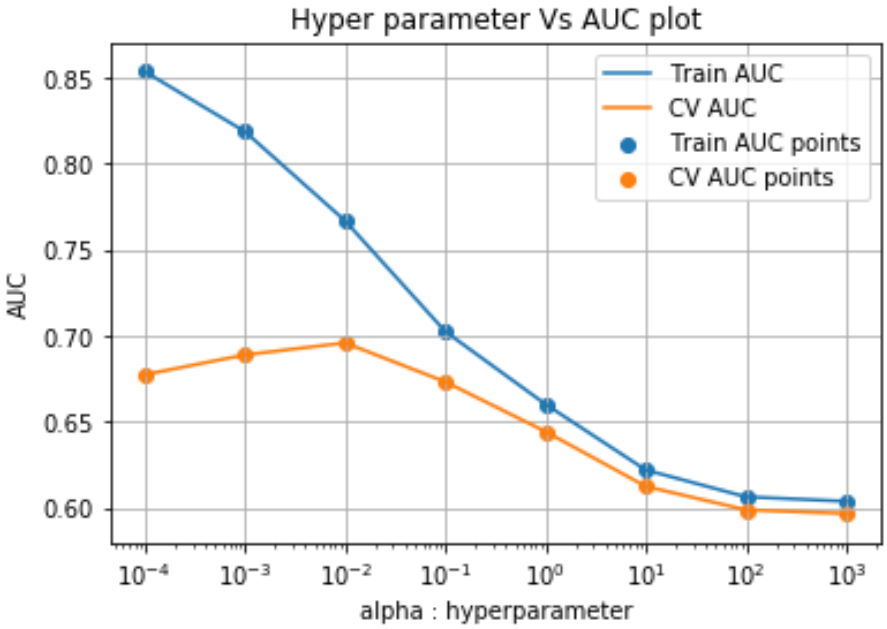
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha
0	3.400181	0.212384	0.006088	0.001038	0.0001
1	1.607698	0.208998	0.006087	0.000702	0.001
2	0.775472	0.119137	0.005187	0.000391	0.01
3	0.530442	0.086121	0.005682	0.000771	0.1
4	0.498349	0.065671	0.005480	0.000669	1
5	0.401686	0.029261	0.006079	0.000528	10
6	0.377685	0.005680	0.005647	0.000718	100
7	0.371282	0.006904	0.006479	0.000662	1000

8 rows × 31 columns

Finding Best Hyper parameter using K-Fold CV on TFIDF representation of text features

In [49]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.G
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.metrics import roc_auc_score

parameters = {'alpha':[0.0001,0.001,0.01,0.1,1,10,10**2,10**3]}
lg = linear_model.SGDClassifier(loss='log', class_weight =  "balanced")
clf = RandomizedSearchCV(lg, parameters, cv=10, scoring='roc_auc', return_tra
clf.fit(X_train_tfidf_matrix, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_alpha'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
K =  results['param_alpha']

plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_s

plt.plot(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("alpha : hyperparameter")
plt.xscale('log')
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```



In [50]:

```
results
```

Out[50]:

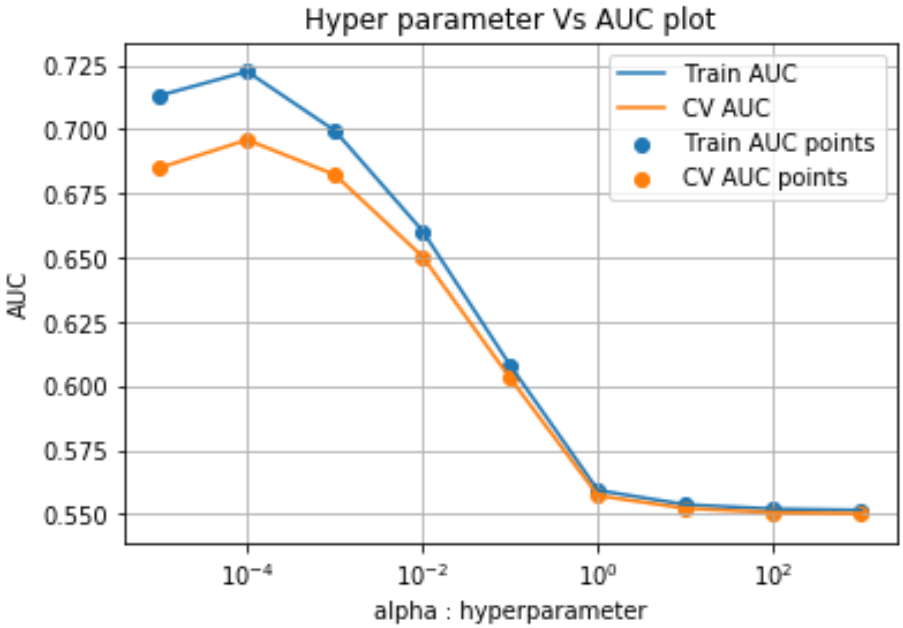
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha
0	3.496891	0.271905	0.006393	0.000662	0.0001
1	1.402952	0.109171	0.006242	0.000886	0.001
2	0.820696	0.079076	0.005790	0.000604	0.01
3	0.528190	0.109299	0.005844	0.000946	0.1
4	0.449862	0.067639	0.005484	0.000923	1
5	0.363410	0.014863	0.005481	0.000502	10
6	0.351892	0.012015	0.005371	0.000481	100
7	0.348597	0.006233	0.005864	0.000840	1000

8 rows × 31 columns

Finding Best Hyper parameter using K-Fold CV on AVG W2V representation of text features

In [74]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.G
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.metrics import roc_auc_score

parameters = {'alpha':[0.00001,0.0001,0.001,0.01,0.1,1,10,10**2,10**3]}
lg = linear_model.SGDClassifier(loss='log', class_weight =  "balanced" )
clf = RandomizedSearchCV(lg, parameters, cv=10, scoring='roc_auc', return_tra
clf.fit(X_train_aw2v_matrix, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_alpha'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
K =  results['param_alpha']

plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_s

plt.plot(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("alpha : hyperparameter")
plt.xscale('log')
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```



In [75]:

```
results
```

Out[75]:

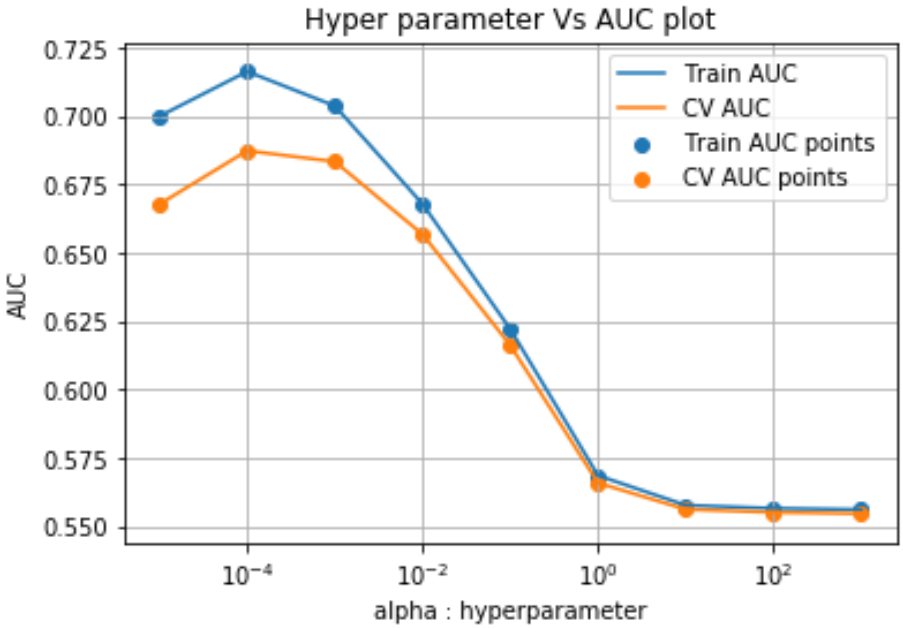
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha
0	26.859048	3.157096	0.015956	0.000475	1e-05
1	9.183159	0.707983	0.016059	0.000308	0.0001
2	3.927870	0.454591	0.018017	0.001044	0.001
3	2.077365	0.265374	0.016218	0.000401	0.01
4	1.508060	0.139357	0.015918	0.000448	0.1
5	1.501867	0.138482	0.015952	0.000441	1
6	1.363403	0.104770	0.016062	0.000289	10
7	1.264981	0.013869	0.016017	0.000507	100
8	1.269457	0.016040	0.016460	0.000679	1000

9 rows × 31 columns

Finding Best Hyper parameter using K-Fold CV on TFIDF W2V representation of text features

In [73]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.G
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.metrics import roc_auc_score

parameters = {'alpha':[0.00001,0.0001,0.001,0.01,0.1,1,10,10**2,10**3]}
lg = linear_model.SGDClassifier(loss='log', class_weight =  "balanced")
clf = RandomizedSearchCV(lg, parameters, cv=10, scoring='roc_auc', return_tra
clf.fit(X_train_tw2v_matrix, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_alpha'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
K =  results['param_alpha']

plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_s

plt.plot(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("alpha : hyperparameter")
plt.xscale('log')
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```



In [76]:

```
results
```

Out[76]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha
0	26.859048	3.157096	0.015956	0.000475	1e-05
1	9.183159	0.707983	0.016059	0.000308	0.0001
2	3.927870	0.454591	0.018017	0.001044	0.001
3	2.077365	0.265374	0.016218	0.000401	0.01
4	1.508060	0.139357	0.015918	0.000448	0.1
5	1.501867	0.138482	0.015952	0.000441	1
6	1.363403	0.104770	0.016062	0.000289	10
7	1.264981	0.013869	0.016017	0.000507	100
8	1.269457	0.016040	0.016460	0.000679	1000

9 rows × 31 columns

In [55]:

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 496
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

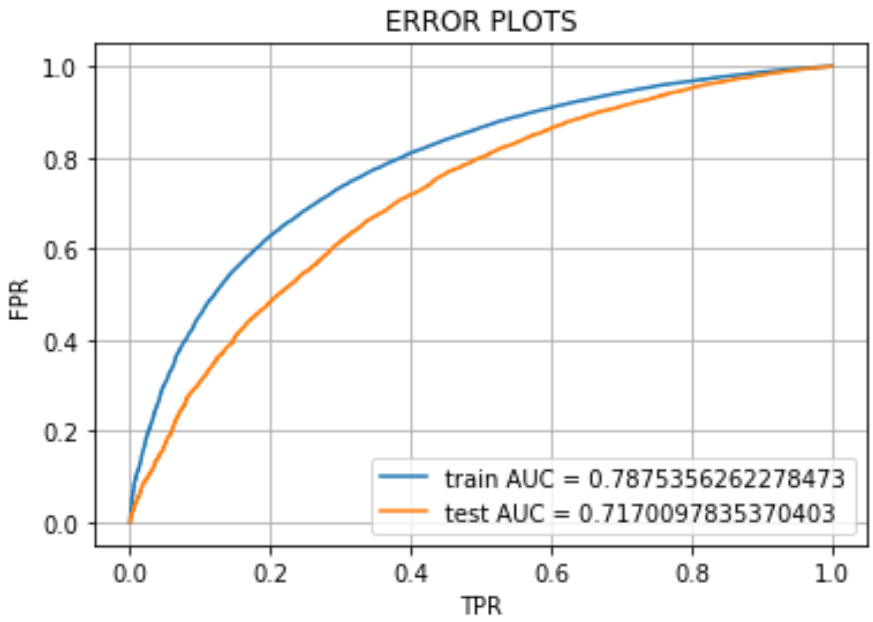
In [56]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very h
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for thresho
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

Applying Logestic Regression with obtained best alpha (Hyper parameter) on BOW

In [57]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


clf = linear_model.SGDClassifier(loss = 'log', alpha = 0.01, class_weight =
clf.fit(X_train_bow_matrix, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability esti
# not the predicted outputs

y_train_pred = batch_predict(clf, X_train_bow_matrix)
y_test_pred = batch_predict(clf, X_test_bow_matrix)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC = "+str(auc(train_fpr, train_
plt.plot(test_fpr, test_tpr, label="test AUC = "+str(auc(test_fpr, test_tpr)
plt.legend()
plt.xlabel("TPR")
plt.ylabel("FPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

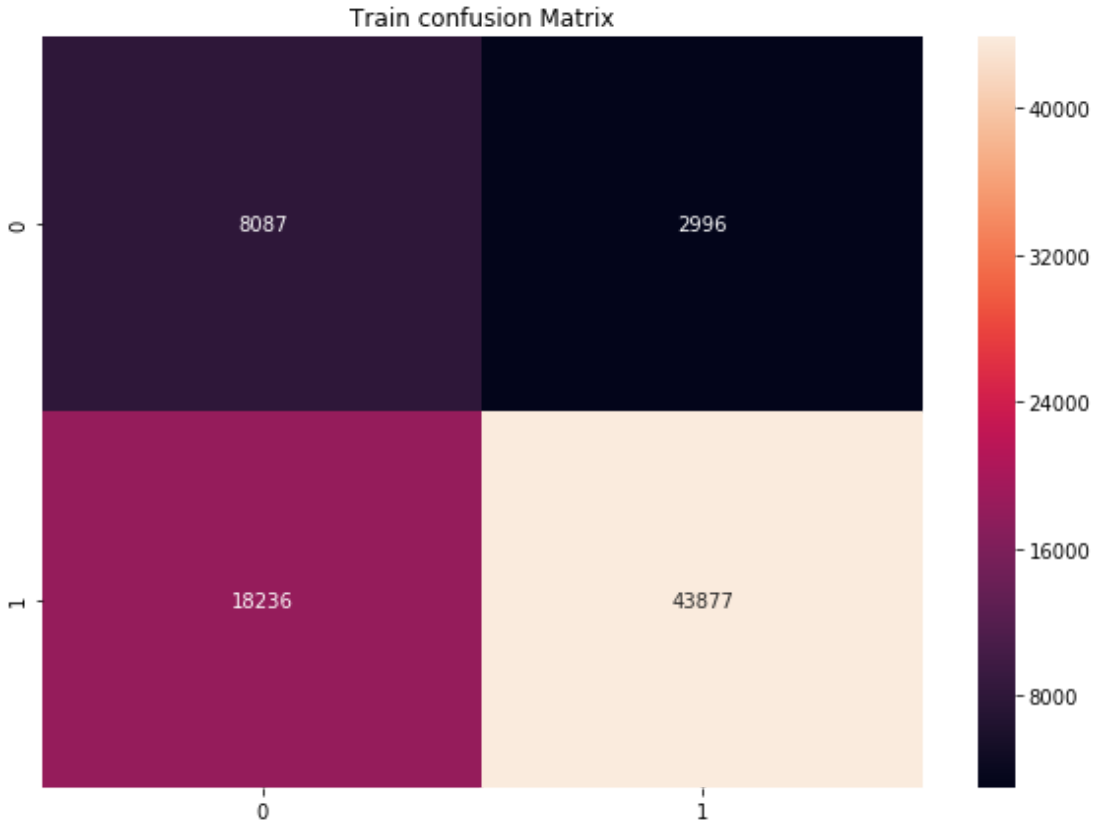


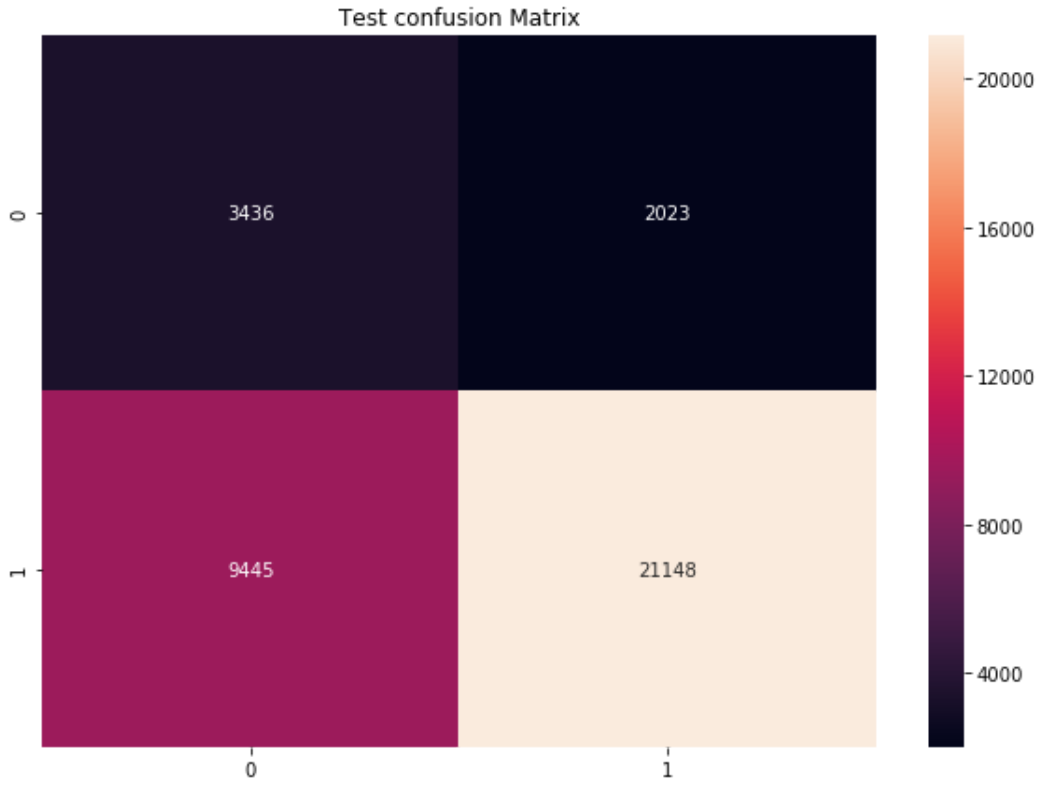
Confusion Matrix with predicted and original labels for BOW

In [58]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
train = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
test = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))

#https://stackoverflow.com/a/35572247

df_cm = pd.DataFrame(train, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Train confusion Matrix')
sns.heatmap(train, annot=True, fmt="d")
plt.show()

df_cm = pd.DataFrame(test, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Test confusion Matrix')
sns.heatmap(test, annot=True, fmt="d")
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.5154476097335494 for thresh
old 0.512
```

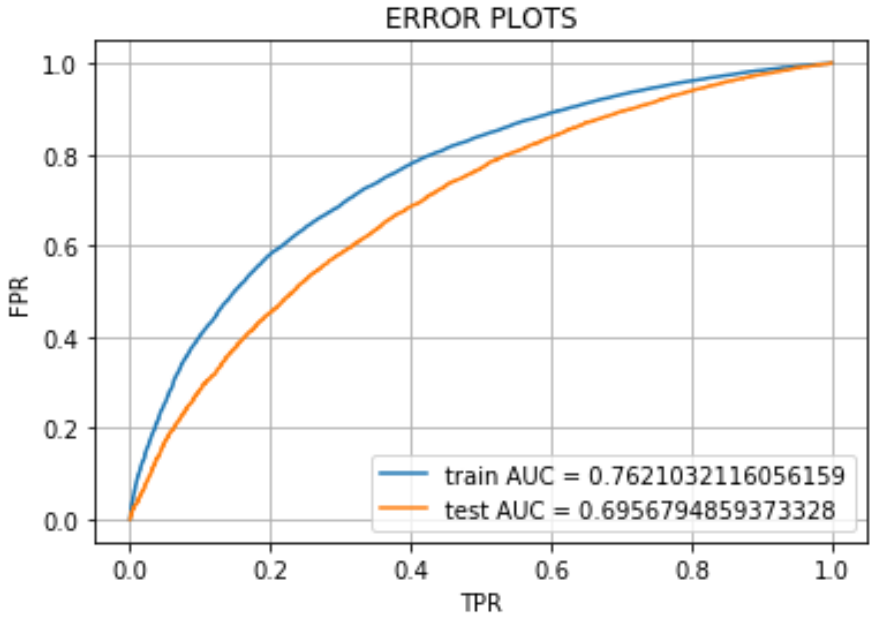




Applying Logestic Regression with obtained best alpha (Hyper parameter) on TFIDF

In [59]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


clf = linear_model.SGDClassifier(loss = 'log', alpha = 0.01, class_weight =
clf.fit(X_train_tfidf_matrix, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability esti
# not the predicted outputs

y_train_pred = batch_predict(clf, X_train_tfidf_matrix)
y_test_pred = batch_predict(clf, X_test_tfidf_matrix)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC = "+str(auc(train_fpr, train_
plt.plot(test_fpr, test_tpr, label="test AUC = "+str(auc(test_fpr, test_tpr)
plt.legend()
plt.xlabel("TPR")
plt.ylabel("FPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

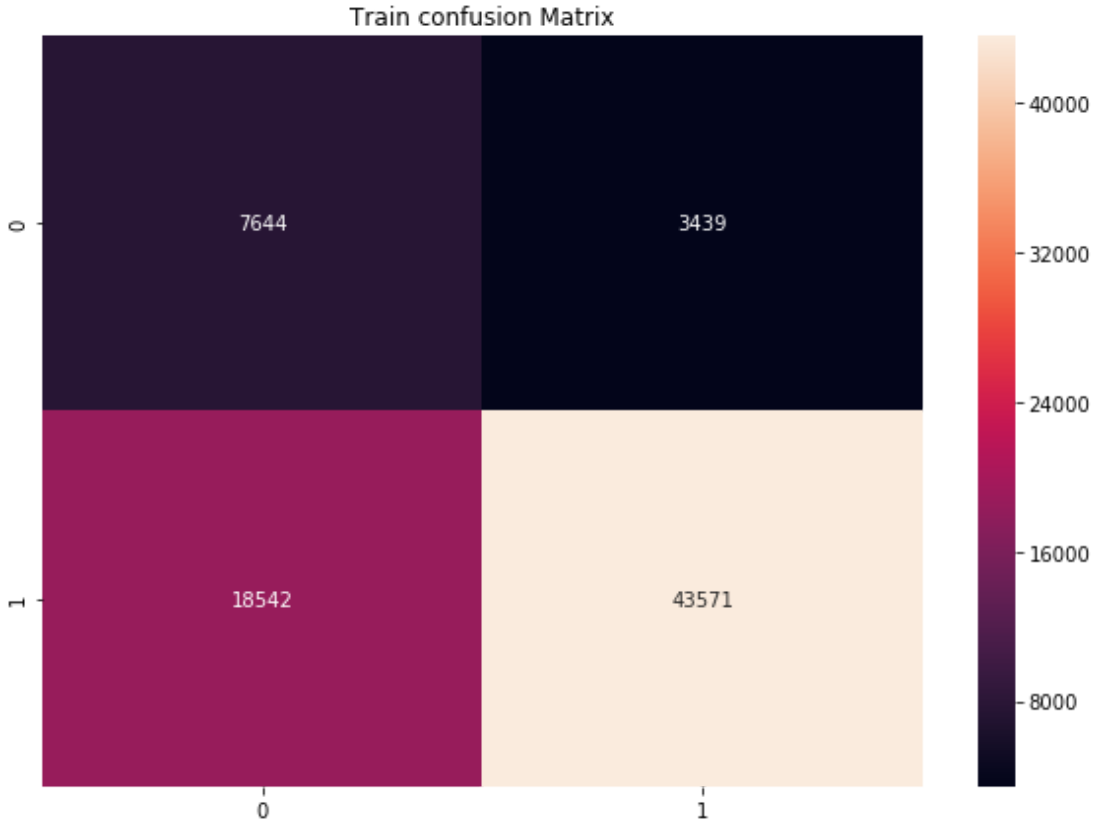


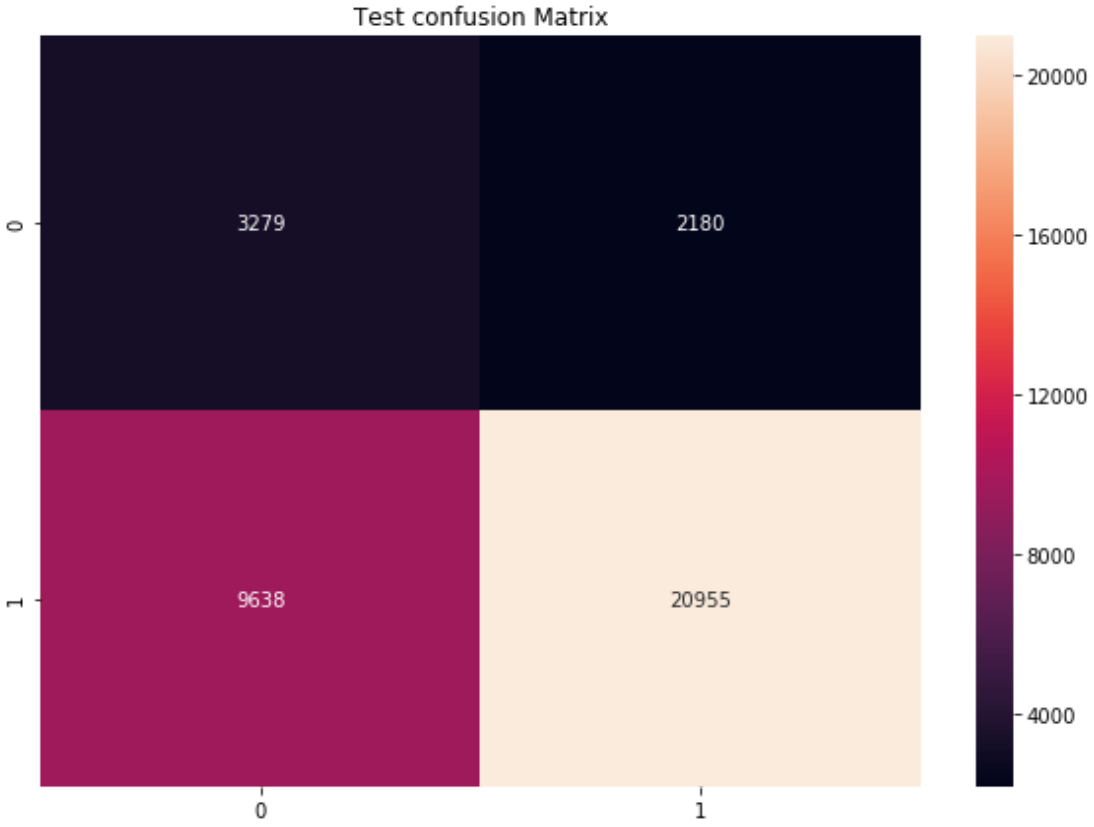
Confusion Matrix with predicted and original labels for TFIDF

In [60]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
train = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
test = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))

#https://stackoverflow.com/a/35572247

df_cm = pd.DataFrame(train, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Train confusion Matrix')
sns.heatmap(train, annot=True, fmt="d")
plt.show()

df_cm = pd.DataFrame(test, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Test confusion Matrix')
sns.heatmap(test, annot=True, fmt="d")
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.48381392833000847 for thres
hold 0.49
```

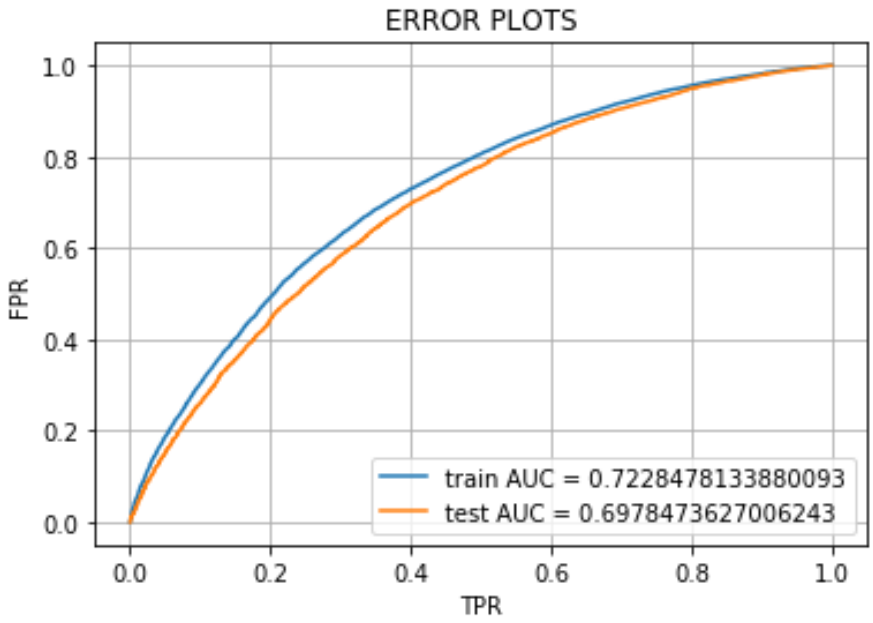




Applying Logestic Regression with obtained best alpha (Hyper parameter) on AVG W2V representation

In [72]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


clf = linear_model.SGDClassifier(loss = 'log', alpha = 0.0001, class_weight =
clf.fit(X_train_aw2v_matrix, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability esti
# not the predicted outputs

y_train_pred = batch_predict(clf, X_train_aw2v_matrix)
y_test_pred = batch_predict(clf, X_test_aw2v_matrix)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC = "+str(auc(train_fpr, train_
plt.plot(test_fpr, test_tpr, label="test AUC = "+str(auc(test_fpr, test_tpr)
plt.legend()
plt.xlabel("TPR")
plt.ylabel("FPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

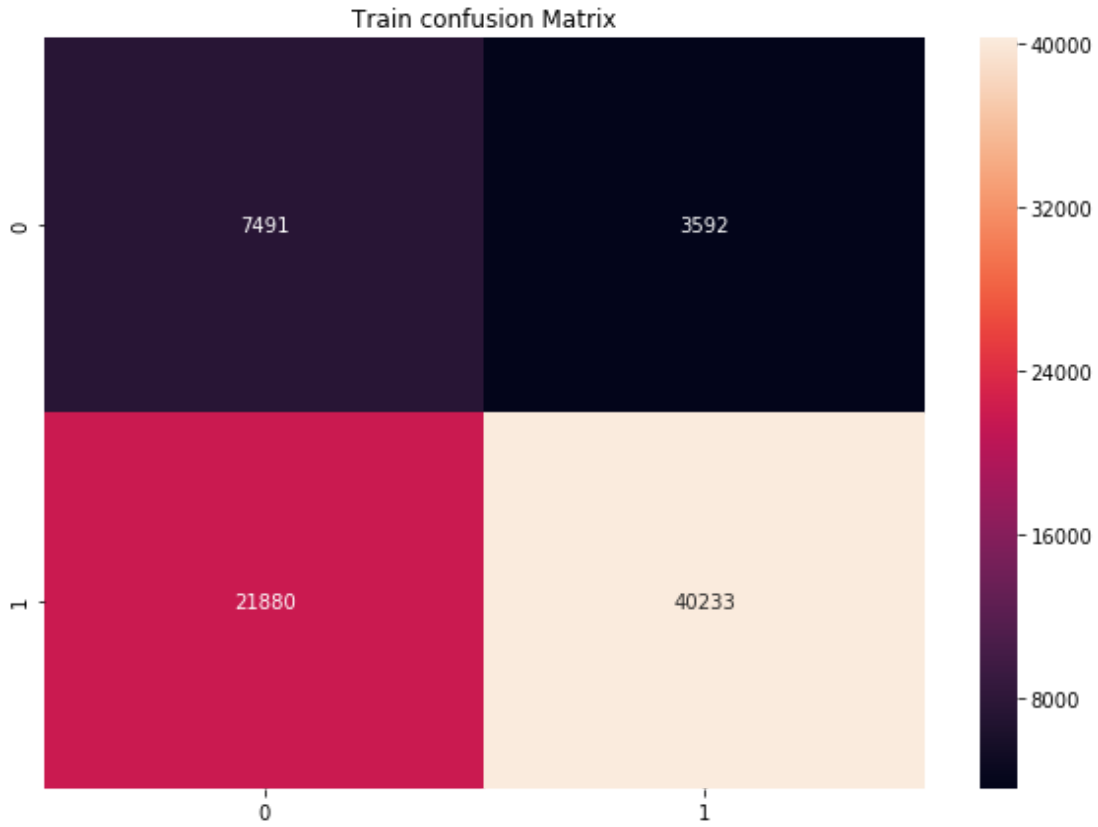


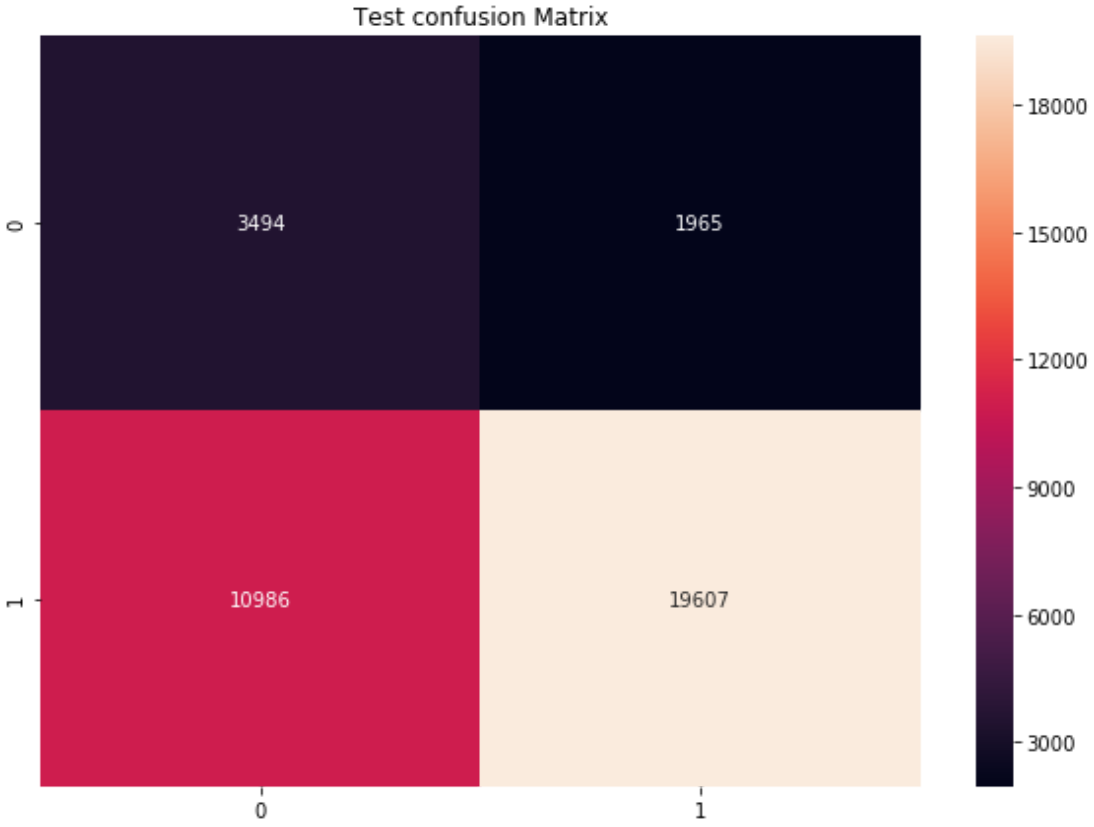
Confusion Matrix with predicted and original labels for AVG W2V

In [78]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
train = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
test = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))

#https://stackoverflow.com/a/35572247

df_cm = pd.DataFrame(train, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Train confusion Matrix')
sns.heatmap(train, annot=True, fmt="d")
plt.show()

df_cm = pd.DataFrame(test, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Test confusion Matrix')
sns.heatmap(test, annot=True, fmt="d")
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.4378066715348846 for thresh
old 0.631
```

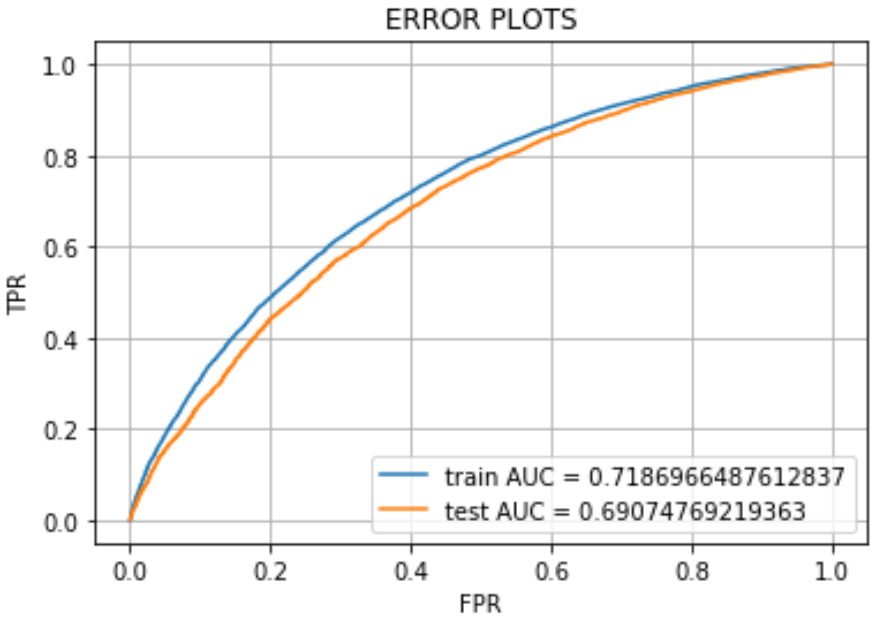




Applying Logestic Regression with obtained best alpha (Hyper parameter) on TFIDF W2V representation

In [77]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


clf = linear_model.SGDClassifier(loss = 'log', alpha = 0.0001, class_weight =
clf.fit(X_train_tw2v_matrix, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability esti
# not the predicted outputs

y_train_pred = batch_predict(clf, X_train_tw2v_matrix)
y_test_pred = batch_predict(clf, X_test_tw2v_matrix)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC = "+str(auc(train_fpr, train_
plt.plot(test_fpr, test_tpr, label="test AUC = "+str(auc(test_fpr, test_tpr)
plt.legend()
plt.ylabel("TPR")
plt.xlabel("FPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

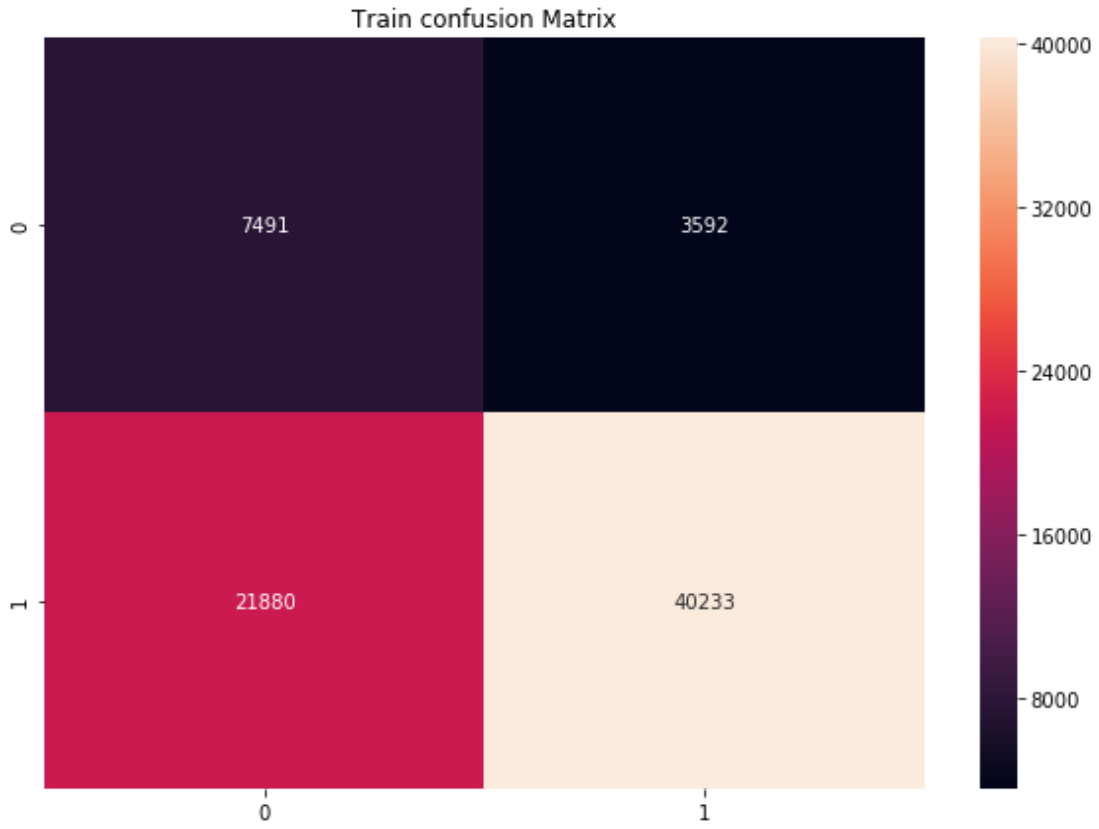


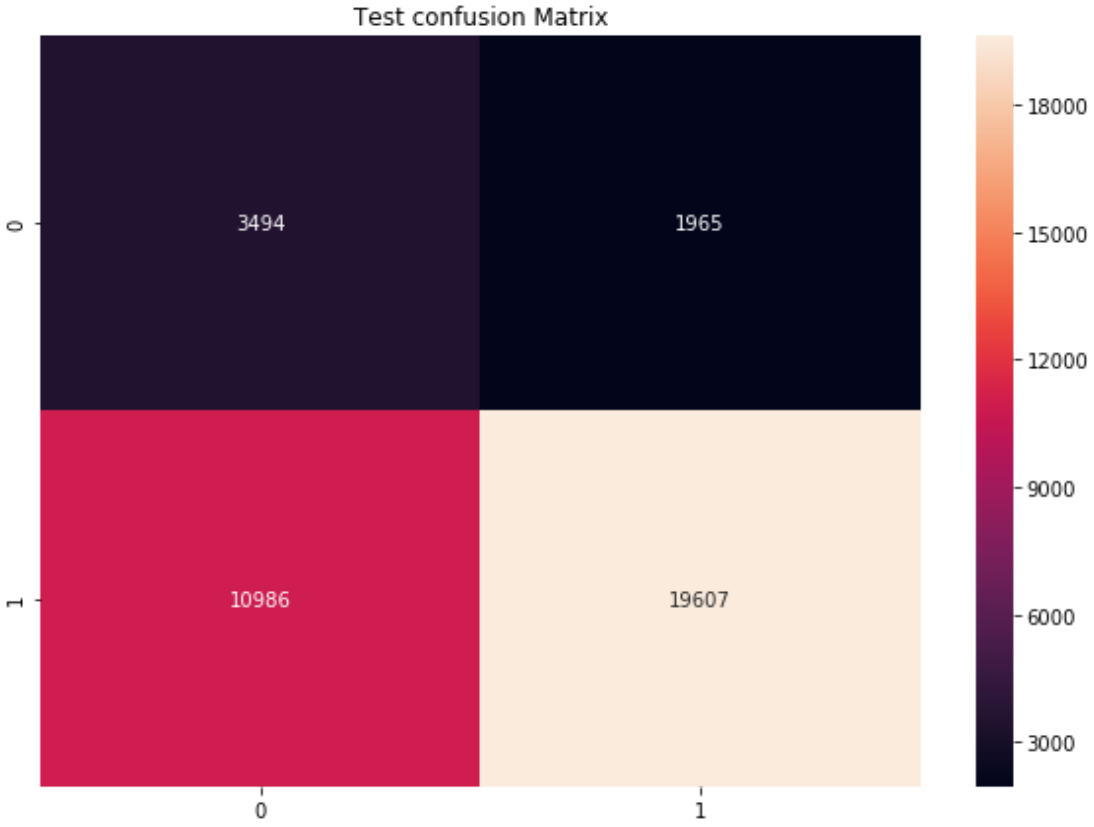
Confusion Matrix with predicted and original labels for TFIDF W2V

In [79]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
train = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
test = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))

#https://stackoverflow.com/a/35572247

df_cm = pd.DataFrame(train, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Train confusion Matrix')
sns.heatmap(train, annot=True, fmt="d")
plt.show()

df_cm = pd.DataFrame(test, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Test confusion Matrix')
sns.heatmap(test, annot=True, fmt="d")
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.4378066715348846 for thresh
old 0.631
```





Matrix without text features

In [65]:

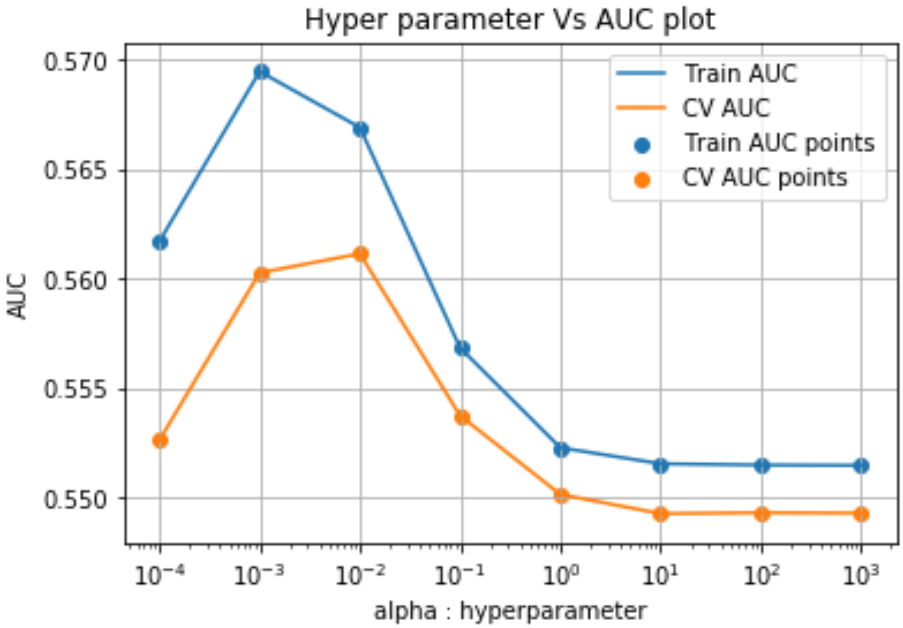
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_train_num_matrix = hstack((X_train_cc_ohe, X_train_csc_ohe, X_train_grade_c
                             X_train_teacher_ohe, X_train_price_norm.reshape(
                             X_train_ppp_norm.reshape(-1,1), X_train_ewc_norm
                             X_train_quantity_norm.reshape(-1,1), X_train_ss_
                             X_train_twc_norm.reshape(-1,1))).tocsr()

X_test_num_matrix = hstack((X_test_cc_ohe, X_test_csc_ohe,X_test_grade_ohe, X
                             X_test_teacher_ohe, X_test_price_norm.reshape(-1
                            X_test_ppp_norm.reshape(-1,1), X_test_ewc_norm.re
                            X_test_quantity_norm.reshape(-1,1), X_test_ss_nor
                             X_test_twc_norm.reshape(-1,1))).tocsr()

print("Final Data matrix")
print(X_train_num_matrix.shape, y_train.shape)
print(X_test_num_matrix.shape, y_test.shape)
```

```
Final Data matrix
(73196, 105) (73196,)
(36052, 105) (36052,)
```

Finding Best Hyper parameter using K-Fold CV

In [66]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.G
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.metrics import roc_auc_score

parameters = {'alpha':[0.0001,0.001,0.01,0.1,1,10,10**2,10**3]}
lg = linear_model.SGDClassifier(loss='log', class_weight =  "balanced")
clf = RandomizedSearchCV(lg, parameters, cv=10, scoring='roc_auc', return_tra
clf.fit(X_train_num_matrix, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_alpha'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
K =  results['param_alpha']

plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_s

plt.plot(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("alpha : hyperparameter")
plt.xscale('log')
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```



In [67]:

```
results
```

Out[67]:

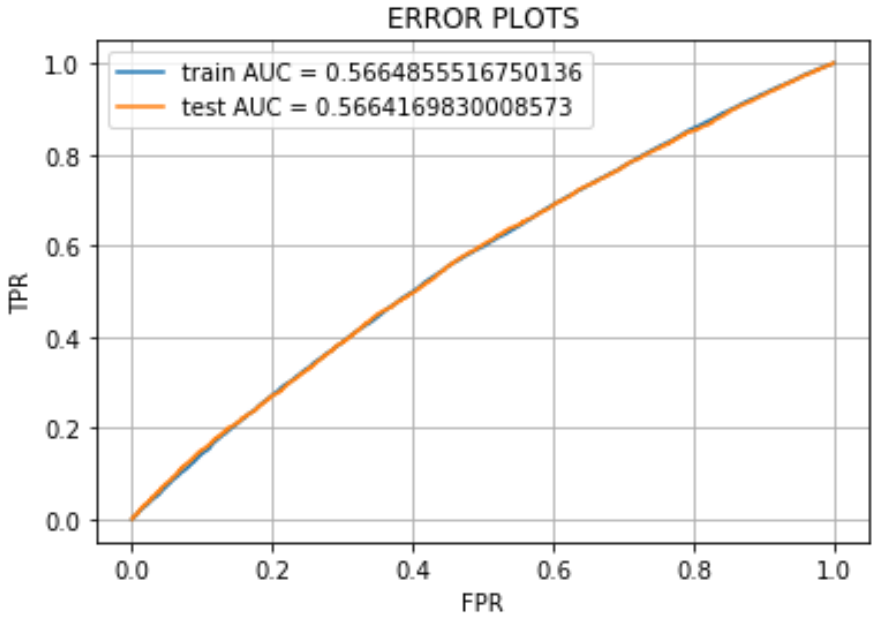
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha
0	0.239190	0.025124	0.002747	0.000402	0.0001
1	0.130242	0.023530	0.002595	0.000489	0.001
2	0.110369	0.019879	0.002791	0.000399	0.01
3	0.094293	0.017072	0.002690	0.000453	0.1
4	0.079370	0.002756	0.002795	0.000400	1
5	0.080168	0.004375	0.002688	0.000453	10
6	0.077542	0.001397	0.002790	0.000396	100
7	0.082415	0.004974	0.002791	0.000589	1000

8 rows × 31 columns

Applying Logestic Regression with obtained best alpha

In [70]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


clf = linear_model.SGDClassifier(loss = 'log', alpha = 0.01, class_weight =
clf.fit(X_train_num_matrix, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability esti
# not the predicted outputs

y_train_pred = batch_predict(clf, X_train_num_matrix)
y_test_pred = batch_predict(clf, X_test_num_matrix)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC = "+str(auc(train_fpr, train_
plt.plot(test_fpr, test_tpr, label="test AUC = "+str(auc(test_fpr, test_tpr))
plt.legend()
plt.ylabel("TPR")
plt.xlabel("FPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

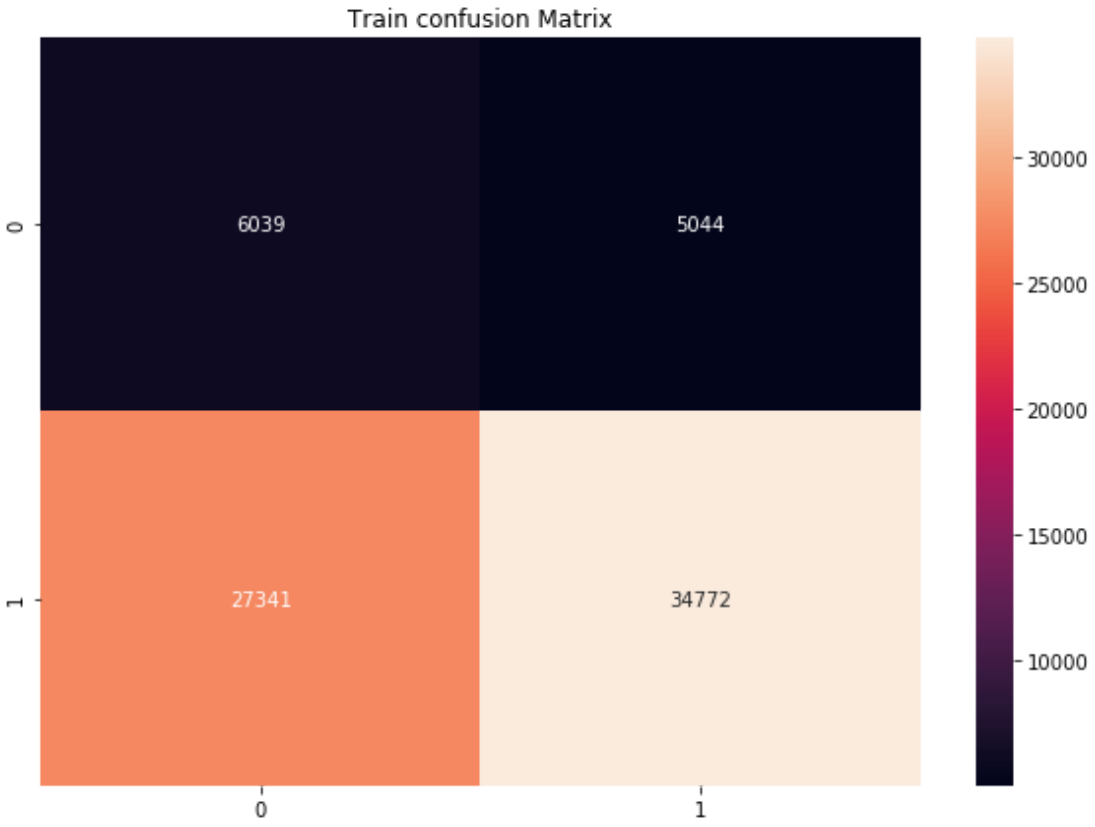


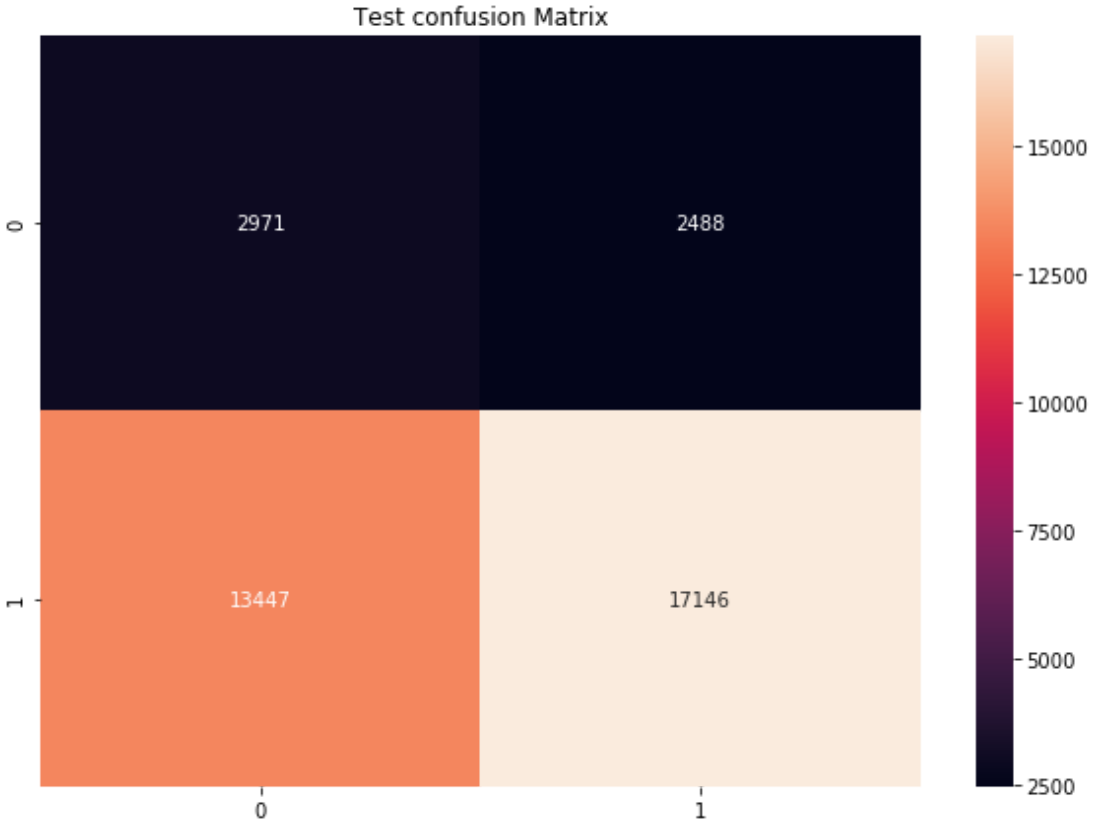
Confusion matrix representation

In [71]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
train = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
test = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))

#https://stackoverflow.com/a/35572247

df_cm = pd.DataFrame(train, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Train confusion Matrix')
sns.heatmap(train, annot=True, fmt="d")
plt.show()

df_cm = pd.DataFrame(test, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Test confusion Matrix')
sns.heatmap(test, annot=True, fmt="d")
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.30503864390999674 for thres
hold 0.499
```




Test confusion Matrix
2971
2488
13447
17146
0
1
0
1
15000
12500
10000
7500
5000
2500

Conclusion

In [80]:

```
# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

table = PrettyTable()
table.field_names = ["Vectorizer", "Model", "Hyper Parameter", "AUC"]

table.add_row(['BOW', 'Logestic Regression', 0.01, 0.717])
table.add_row(['TFIDF', 'Logestic Regression', 0.01, 0.6956])
table.add_row(['AVG W2V', 'Logestic Regression', 0.0001, 0.6978])
table.add_row(['TFIDF W2V', 'Logestic Regression', 0.0001, 0.6907])
table.add_row(['Without Text Features', 'Logestic Regression', 0.01, 0.5664])
print(table)
```

```
+----------------------+---------------------+--------------
--+--------+
|       Vectorizer     |        Model        | Hyper Paramete
r |  AUC   |
+----------------------+---------------------+--------------
--+--------+
|          BOW         | Logestic Regression |       0.01
| 0.717  |
|         TFIDF        | Logestic Regression |       0.01
| 0.6956 |
|        AVG W2V       | Logestic Regression |      0.0001
| 0.6978 |
|       TFIDF W2V      | Logestic Regression |      0.0001
| 0.6907 |
| Without Text Features | Logestic Regression |       0.01
| 0.5664 |
+----------------------+---------------------+--------------
--+--------+
```

Summary

- BOW vectorizer gave AUC 0.717 with the best hyper parameter 0.01
- TFIDF vectorizer gave AUC 0.6956 with the best hyper parameter 0.01
- AVG W2V vectorizer gave AUC 0.6978 with the best hyper parameter 0.0001
- TFIDF W2V vectorizer gave AUC 0.6907 with the best hyper parameter 0.0001
- Text-less features gave AUC 0.5664 with the best hyper parameter 0.01
- BOW vectorizer has the best AUC
- AVG W2V, TFIDF and TFIDF W2V has the next best AUC respectively
- AUC for text-less features is less compared to text features, which prooves that text features are important in determining the class label