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
```

Splitting data into Train and Test

In [2]:

```
prepeocessed_data = pd.read_csv('preprocessed_data.csv', nrows=50000)
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]:

```
(50000, 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]:

```
(16500, 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 c

# 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
(33500, 7) (33500,)
(16500, 7) (16500,)
['appliedlearning', 'health_sports', 'history_civics', 'litera
cy_language', 'math_science', 'music_arts', 'specialneeds']
```

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
(33500, 28) (33500,)
(16500, 28) (16500,)
['appliedsciences', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment',
'economics', 'environmentalscience', 'esl', 'extracurricular',
'financialliteracy', 'foreignlanguages', 'gym_fitness', 'healt
h_lifescience', 'health_wellness', 'history_geography', 'liter
acy', 'literature_writing', 'mathematics', 'music', 'nutrition
education', 'other', 'parentinvolvement', 'performingarts', 's
ocialsciences', 'specialneeds', 'teamsports', 'visualarts']
```

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
(33500, 51) (33500,)
(16500, 51) (16500,)
['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
(33500, 5) (33500,)
(16500, 5) (16500,)
['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
(33500, 4) (33500,)
(16500, 4) (16500,)
['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.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)
print(X_test_price_norm)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
[[0.00082349]
 [0.00170606]
 [0.00091177]
 ...
 [0.00048251]
 [0.00551218]
 [0.00178942]]
[[0.00069124]
 [0.00179517]
 [0.00710507]
 ...
 [0.00461667]
 [0.00196602]
 [0.01534895]]
```

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)
print(X_train_ppp_norm)
print(X_test_ppp_norm)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
[[0.        ]
 [0.00041902]
 [0.00419024]
 ...
 [0.00020951]
 [0.        ]
 [0.00062854]]
[[0.00156123]
 [0.00156123]
 [0.00249796]
 ...
 [0.00562042]
 [0.00062449]
 [0.00124898]]
```

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)
print(X_train_quantity_norm)
print(X_test_quantity_norm)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
[[0.00199952]
 [0.00083313]
 [0.00133301]
 ...
 [0.00616517]
 [0.00116638]
 [0.00049988]]
[[0.00136343]
 [0.00090895]
 [0.00068172]
 ...
 [0.00113619]
 [0.00181791]
 [0.00022724]]
```

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)
```

```
33500
16500
0.174
0.193
(33500,)
(16500,)
```

In [14]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(ss_train_array.reshape(1,-1))

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

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

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
[[0.00637785]
 [0.0034135 ]
 [0.00604848]
 ...
 [0.00479087]
 [0.0036231 ]
 [0.0048807 ]]
[[0.00145316]
 [0.00653921]
 [0.00820607]
 ...
 [0.00649647]
 [0.00624003]
 [0.00880443]]
```

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)
```

```
33500
16500
5
5
(33500,)
(16500,)
```

In [16]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
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)
print(X_train_twc_norm)
print(X_test_twc_norm)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
[[0.00292188]
 [0.00389584]
 [0.00194792]
 ...
 [0.00584376]
 [0.0048698 ]
 [0.00292188]]
[[0.00553584]
 [0.00553584]
 [0.00553584]
 ...
 [0.01107168]
 [0.00415188]
 [0.00553584]]
```

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)
```

```
33500
16500
215
189
(33500,)
(16500,)
```

In [18]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
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)
print(X_train_ewc_norm)
print(X_test_ewc_norm)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
[[0.00431107]
 [0.00512833]
 [0.00367769]
 ...
 [0.00474013]
 [0.00508747]
 [0.00661984]]
[[0.00629001]
 [0.01252151]
 [0.00731396]
 ...
 [0.00772355]
 [0.00857197]
 [0.00634852]]
```

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(min_df=10,ngram_range=(1,4), 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)
```

```
(33500, 20) (33500,)
(16500, 20) (16500,)



After vectorizations
(33500, 5000) (33500,)
(16500, 5000) (16500,)
```

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
(33500, 4006) (33500,)
(16500, 4006) (16500,)
```

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
(33500, 5000) (33500,)
(16500, 5000) (16500,)
```

1.5.2 Vectorizing using TFIDF

essay

In [22]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), 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
(33500, 5000) (33500,)
(16500, 5000) (16500,)
```

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
(33500, 4006) (33500,)
(16500, 4006) (16500,)
```

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
(33500, 5000) (33500,)
(16500, 5000) (16500,)
```

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%|████████████████████████████████████████████████████████████
████████████████| 33500/33500 [00:13<00:00, 2544.03it/s]

33500
300
```

In [27]:

```
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%|████████████████████████████████████████████████████████████
████████████████| 16500/16500 [00:06<00:00, 2563.53it/s]
```

project_title

In [28]:

```
# 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%|████████████████████████████████████████████████████████████
████████████| 33500/33500 [00:00<00:00, 109023.43it/s]

33500
300
```

In [29]:

```
avg_w2v_titles_test = []; # the avg-w2v for each sentence/review is stored in
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%|████████████████████████████████████████████████████████████
████████████| 16500/16500 [00:00<00:00, 129222.74it/s]
```

project_resource_summary

In [30]:

```
# 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%|████████████████████████████████████████████████████████████
████████████████| 33500/33500 [00:01<00:00, 27859.48it/s]

33500
300
```

In [31]:

```
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%|████████████████████████████████████████████████████████████
████████████████| 16500/16500 [00:00<00:00, 28178.86it/s]
```

1.5.4 Vectorizing using TFIDF W2V

project_title

In [32]:

```
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%|████████████████████████████████████████████████████████████
████████████████| 33500/33500 [00:00<00:00, 90612.68it/s]


33500
300
```

In [33]:

```
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%|██████████████████████████████████████████████████████████
██████████████| 16500/16500 [00:00<00:00, 87748.95it/s]

16500
300
```

essay

In [34]:

```
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%|████████████████████████████████████████████████████████████
████████████████| 33500/33500 [02:23<00:00, 234.20it/s]

33500
300
```

In [35]:

```
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%|████████████████████████████████████████████████████████████
████████████████| 16500/16500 [01:09<00:00, 236.19it/s]

16500
300
```

project_resource_summary

In [36]:

```
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%|██████████████████████████████████████████████████████████
██████████████| 33500/33500 [00:03<00:00, 9392.73it/s]

33500
300
```

In [37]:

```
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%|██████████████████████████████████████████████████████████
██████████████████| 16500/16500 [00:01<00:00, 9359.72it/s]

16500
300
```

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

In [38]:

```
# 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_teacher_ohe, X_train_price_norm, X_train
                             X_train_titles_bow, X_train_psr_bow)).tocsr()


X_test_bow_matrix = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_grade_ohe,
                            X_test_price_norm, X_test_ppp_norm, X_test_essay_

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
(33500, 14103) (33500,)
(16500, 14103) (16500,)
```

In [39]:

```
# 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_teacher_ohe, X_train_price_norm, X_train
                             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, X_test_ppp_norm, X_test_titles
                            X_test_essay_tfidf, X_test_prs_tfidf)).tocsr()

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

```
Final Data matrix
(33500, 14103) (33500,)
(16500, 14103) (16500,)
```

In [40]:

```
# 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_teacher_ohe, X_train_price_norm, X_trair
                              avg_w2v_essay_train, avg_w2v_titles_train, avg_v

X_test_aw2v_matrix = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_grade_ohe,
                             X_test_price_norm, X_test_ppp_norm, avg_w2v_essay
                               avg_w2v_titles_test, avg_w2v_prs_test)).tocsr()

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
(33500, 997) (33500,)
(16500, 997) (16500,)
```

In [41]:

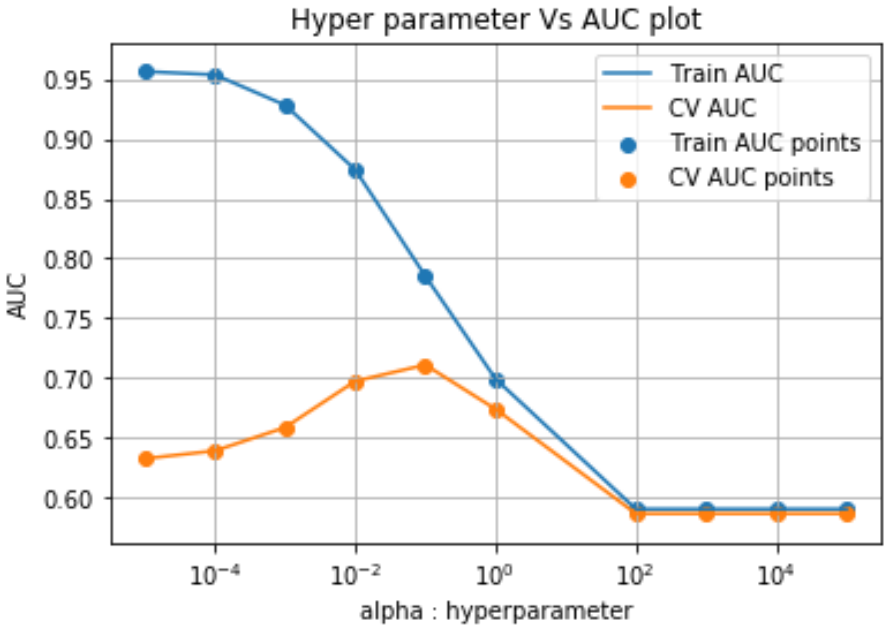
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_train_tw2v_matrix = hstack((X_train_cc_ohe, X_train_csc_ohe, X_train_grade_
                              X_train_teacher_ohe, X_train_price_norm, X_trair
                              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, X_test_ppp_norm, tfidf_w2v_es
                              tfidf_w2v_prs_test)).tocsr()

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

```
Final Data matrix
(33500, 997) (33500,)
(16500, 997) (16500,)
```

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

In [42]:

```
# 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**3,10**4,1
lg = linear_model.SGDClassifier(loss='hinge', 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[42]:

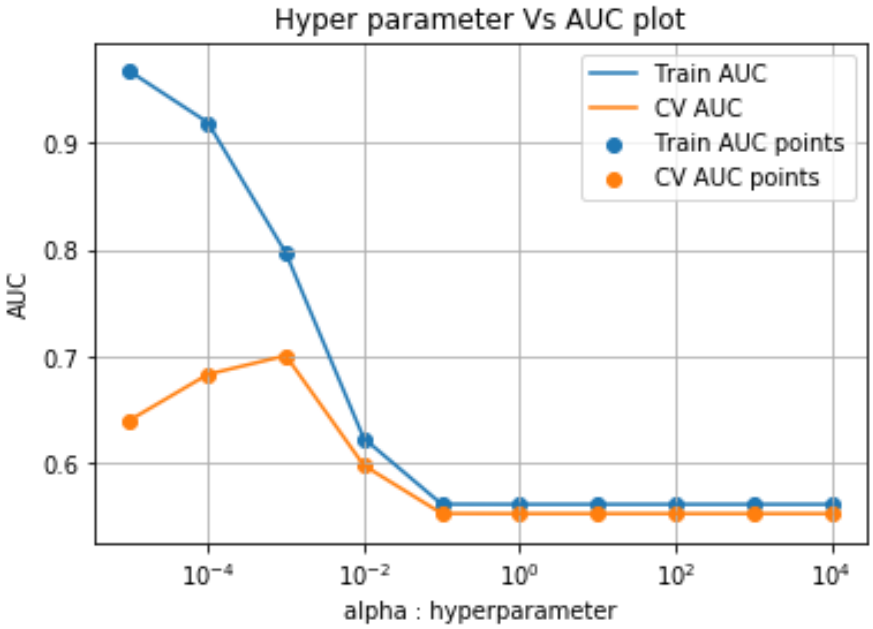
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha
4	3.705551	0.407134	0.003790	0.000589	1e-05
3	3.623908	0.517933	0.004192	0.001069	0.0001
5	2.106709	0.398396	0.003887	0.000531	0.001
9	1.134348	0.187876	0.004284	0.000459	0.01
7	0.654094	0.053021	0.004379	0.000901	0.1
0	0.405269	0.077176	0.003989	0.000445	1
1	0.319217	0.082710	0.004089	0.000538	100
2	0.271590	0.018076	0.004294	0.000639	1000
6	0.250843	0.011475	0.004087	0.000532	10000
8	0.247913	0.007620	0.004382	0.001010	100000

10 rows × 31 columns

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

In [43]:

```
# 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**3,10**4,1
lg = linear_model.SGDClassifier(loss='hinge', 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 [44]:

```
results
```

Out[44]:

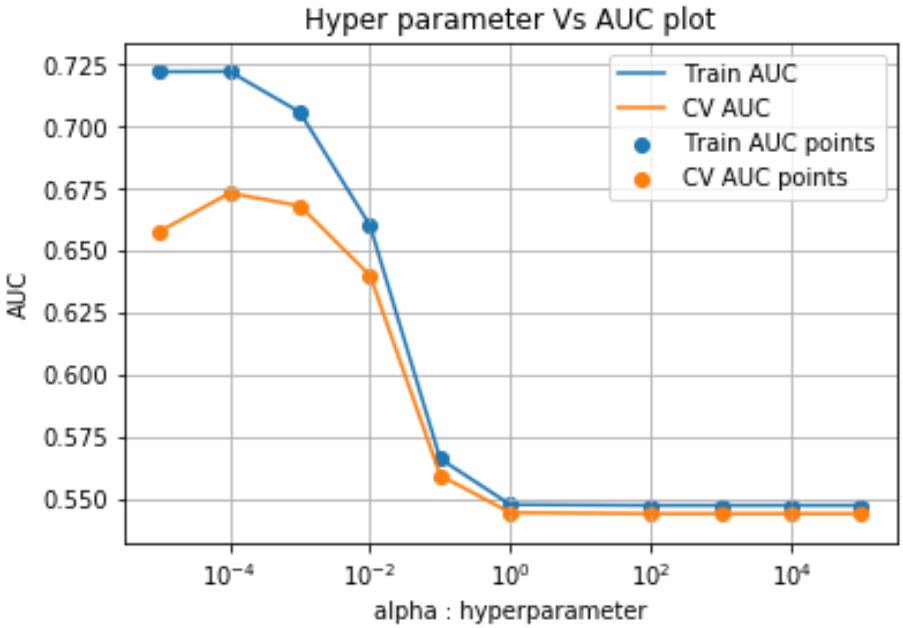
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha
8	1.831711	0.254472	0.004286	0.000636	1e-05
2	1.083532	0.120336	0.004088	0.000817	0.0001
5	0.738935	0.101035	0.004388	0.000482	0.001
1	0.440023	0.093186	0.003978	0.000626	0.01
9	0.306921	0.052768	0.003992	0.000631	0.1
7	0.261198	0.022549	0.004206	0.000624	1
6	0.257705	0.013555	0.004246	0.000407	10
3	0.244402	0.002363	0.004288	0.000896	100
4	0.247644	0.004284	0.003987	0.000883	1000
0	0.250729	0.007051	0.004088	0.000699	10000

10 rows × 31 columns

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

In [45]:

```
# 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**3,10**4,1
lg = linear_model.SGDClassifier(loss='hinge', 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 [46]:

```
results
```

Out[46]:

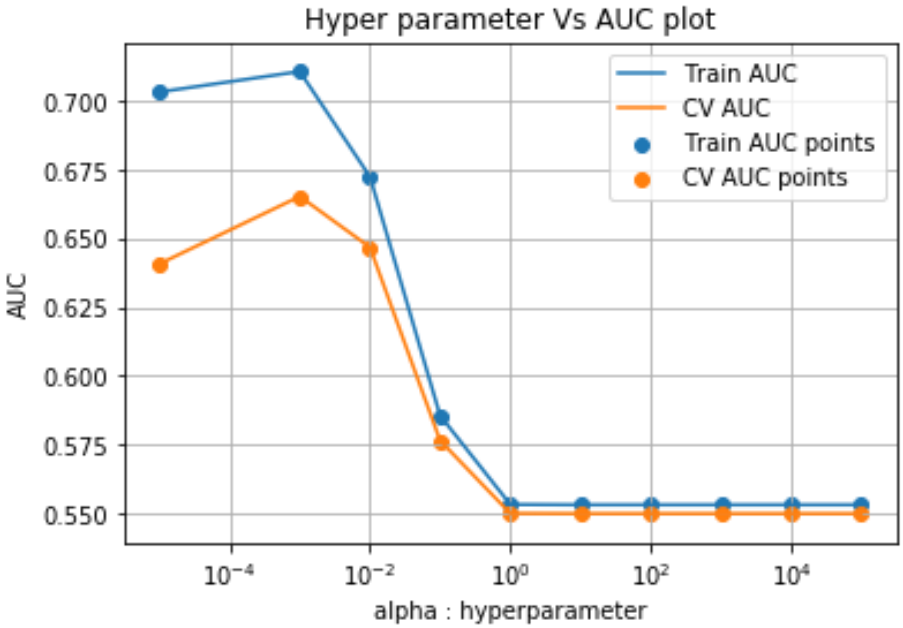
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha
6	9.086499	1.317810	0.008247	0.000657	1e-05
4	4.671954	0.862725	0.008378	0.000645	0.0001
3	2.556680	0.405230	0.008072	0.000292	0.001
9	1.663790	0.214790	0.008588	0.001117	0.01
8	1.066887	0.239478	0.008330	0.000774	0.1
2	0.721862	0.081607	0.007987	0.000631	1
5	0.650294	0.041080	0.007979	0.000625	100
0	0.607905	0.008111	0.008574	0.000799	1000
1	0.596696	0.005529	0.008180	0.000391	10000
7	0.593705	0.005063	0.008436	0.000925	100000

10 rows × 31 columns

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

In [47]:

```
# 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**3,10**4,1
lg = linear_model.SGDClassifier(loss='hinge', 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 [48]:

```
results
```

Out[48]:

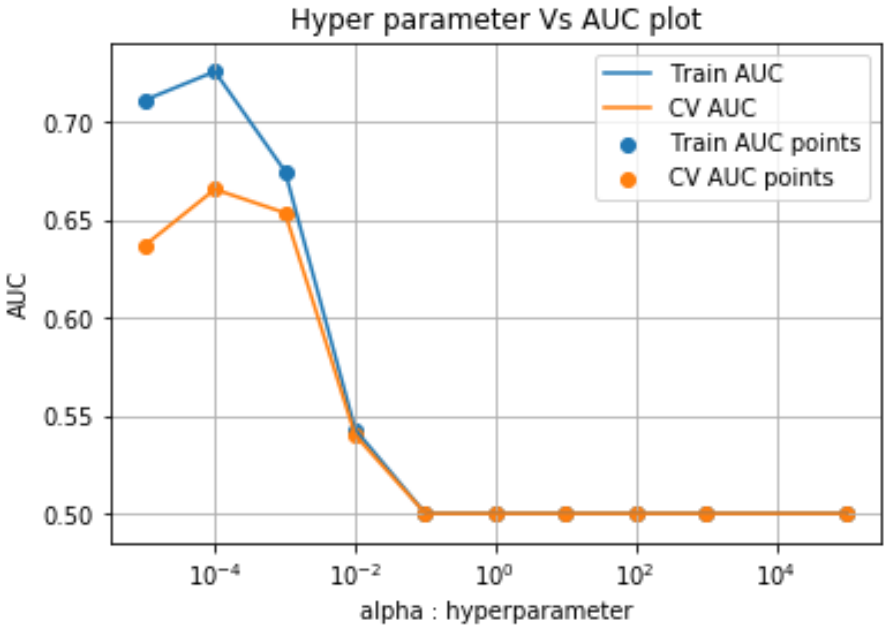
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha
0	9.200544	1.038807	0.008114	0.000592	1e-05
6	2.617115	0.473142	0.008075	0.000929	0.001
1	1.863555	0.347652	0.008279	0.000891	0.01
2	0.978980	0.211754	0.008280	0.000787	0.1
9	0.968030	0.276316	0.008770	0.001072	1
4	0.676764	0.124037	0.009068	0.001207	10
7	0.618917	0.032692	0.008785	0.001239	100
5	0.612464	0.020794	0.008375	0.000795	1000
8	0.595820	0.020974	0.007980	0.000773	10000
3	0.590044	0.009093	0.008380	0.000492	100000

10 rows × 31 columns

Finding Best Hyper parameter using K-Fold CV on TFIDF W2V representation of text features with penalty='l1'

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':[0.00001,0.0001,0.001,0.01,0.1,1,10,10**2,10**3,10**4,1
lg = linear_model.SGDClassifier(loss='hinge', penalty='l1', class_weight =  '
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 [50]:

```
results
```

Out[50]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha
0	38.092996	5.857603	0.008377	0.000789	1e-05
6	23.907859	4.098395	0.008468	0.000918	0.0001
4	8.864495	2.308474	0.008321	0.000840	0.001
1	4.612566	1.986619	0.008474	0.001018	0.01
3	1.885079	0.036889	0.007778	0.000592	0.1
5	1.771614	0.149717	0.007772	0.000594	1
9	1.657567	0.047683	0.008226	0.001277	10
7	1.643434	0.028024	0.007579	0.000661	100
2	1.688293	0.031760	0.008564	0.000998	1000
8	1.658562	0.034443	0.008279	0.001168	100000

10 rows × 31 columns

In [ ]:

```
```

In [51]:

```
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 - 490
    # 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
```

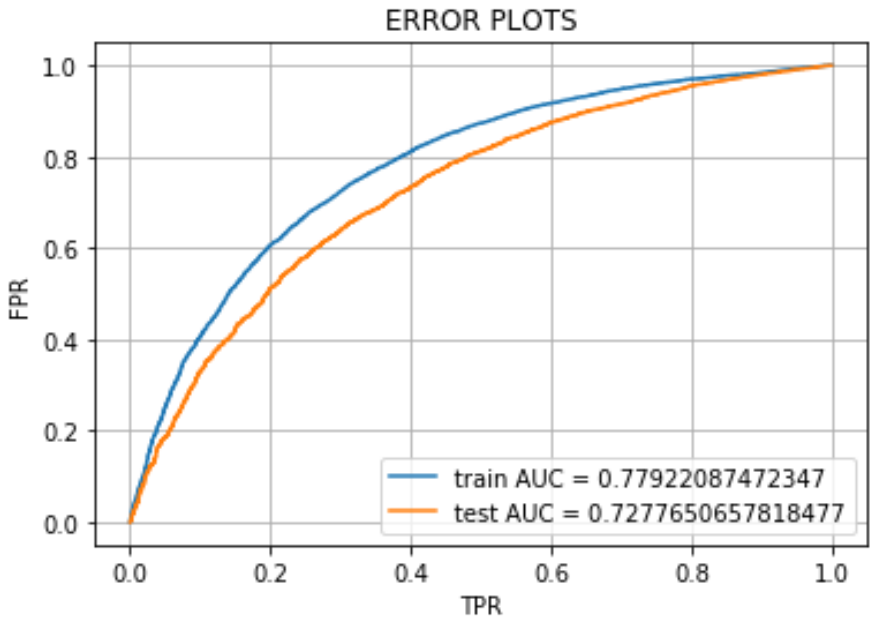
In [52]:

```
# 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 SVM with obtained best Hyper parameter on BOW

In [53]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve
# https://stackoverflow.com/a/57789235
#https://stackoverflow.com/a/56747024

from sklearn.metrics import roc_curve, auc
from sklearn.calibration import CalibratedClassifierCV

clf = linear_model.SGDClassifier(loss = 'hinge', alpha = 0.1, 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
calibrator = CalibratedClassifierCV(clf, cv='prefit')
model=calibrator.fit(X_train_bow_matrix, y_train)

y_train_pred = clf.decision_function (X_train_bow_matrix)
y_test_pred = clf.decision_function (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()
```

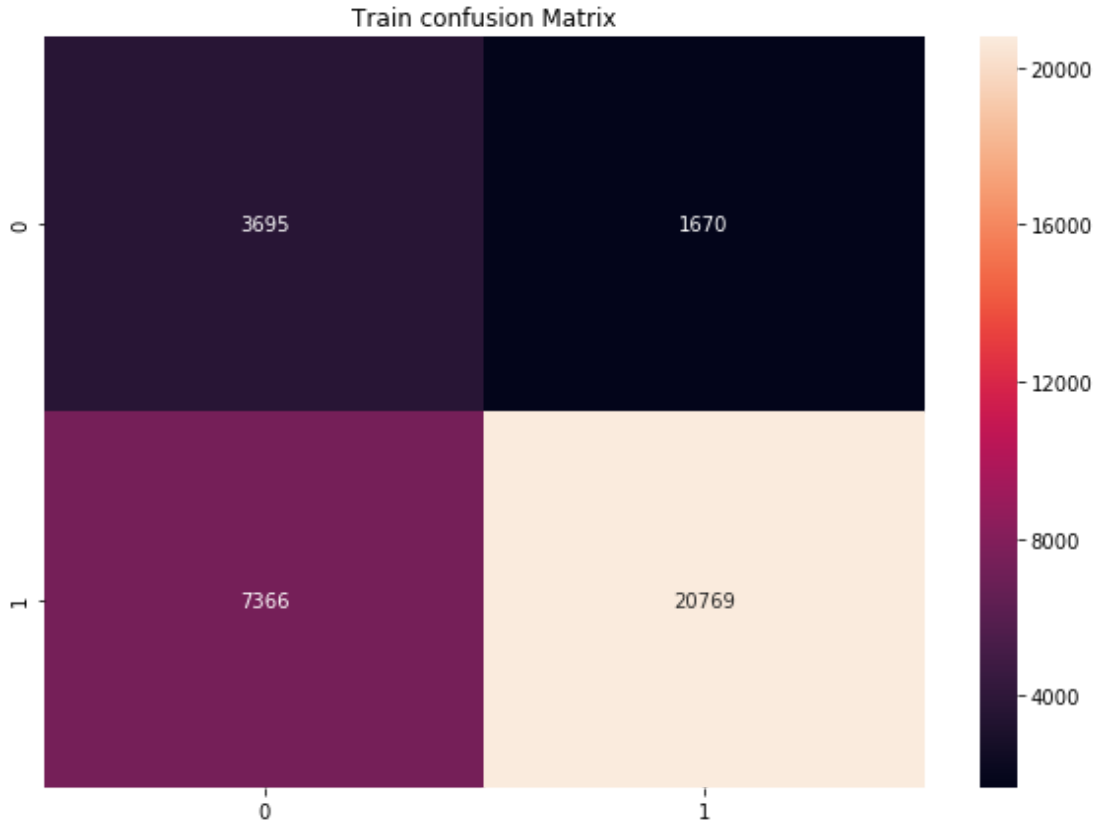


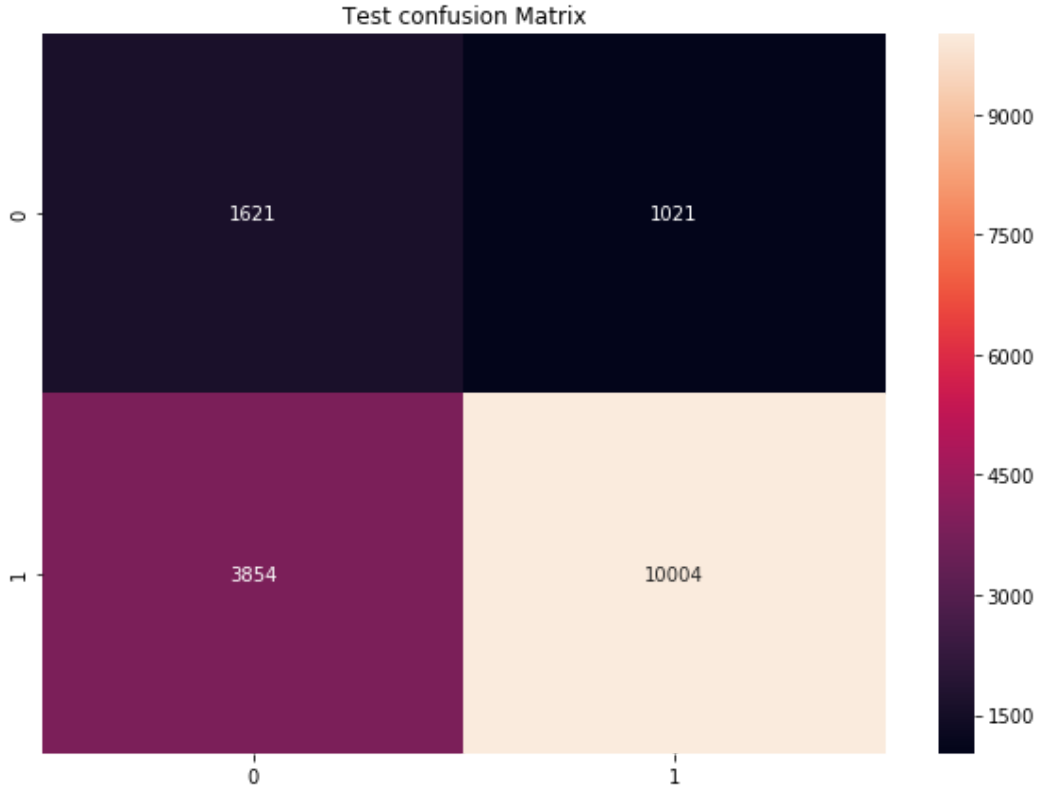
Confusion Matrix with predicted and original labels for BOW

In [54]:

```
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) 0.5084091794803082 for thresh
old -0.272
```

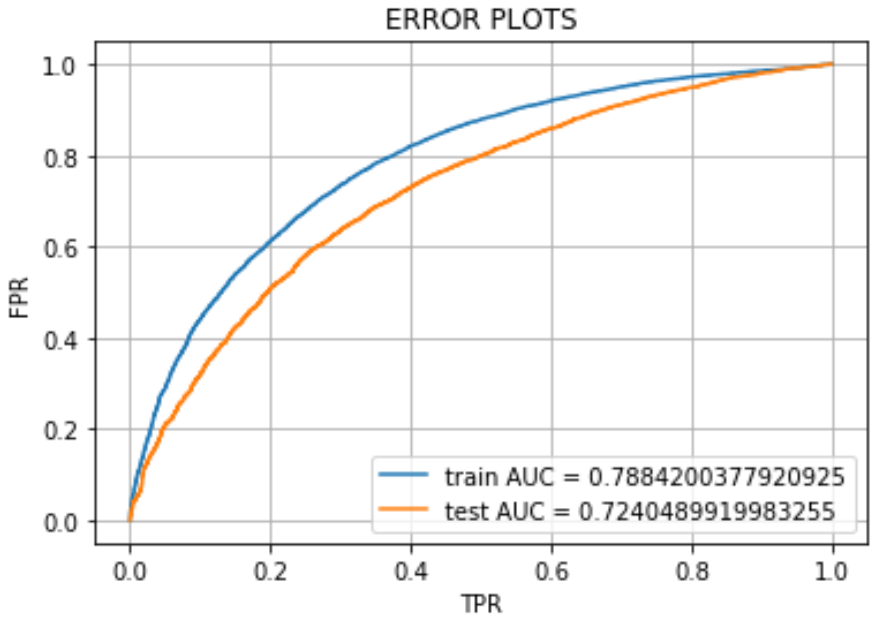




Applying SVM with obtained best Hyper parameter on TFIDF

In [55]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve
# https://stackoverflow.com/a/57789235
#https://stackoverflow.com/a/56747024

from sklearn.metrics import roc_curve, auc
from sklearn.calibration import CalibratedClassifierCV

clf = linear_model.SGDClassifier(loss = 'hinge', alpha = 0.001, 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
calibrator = CalibratedClassifierCV(clf, cv='prefit')
model=calibrator.fit(X_train_tfidf_matrix, y_train)

y_train_pred = clf.decision_function (X_train_tfidf_matrix)
y_test_pred = clf.decision_function (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()
```

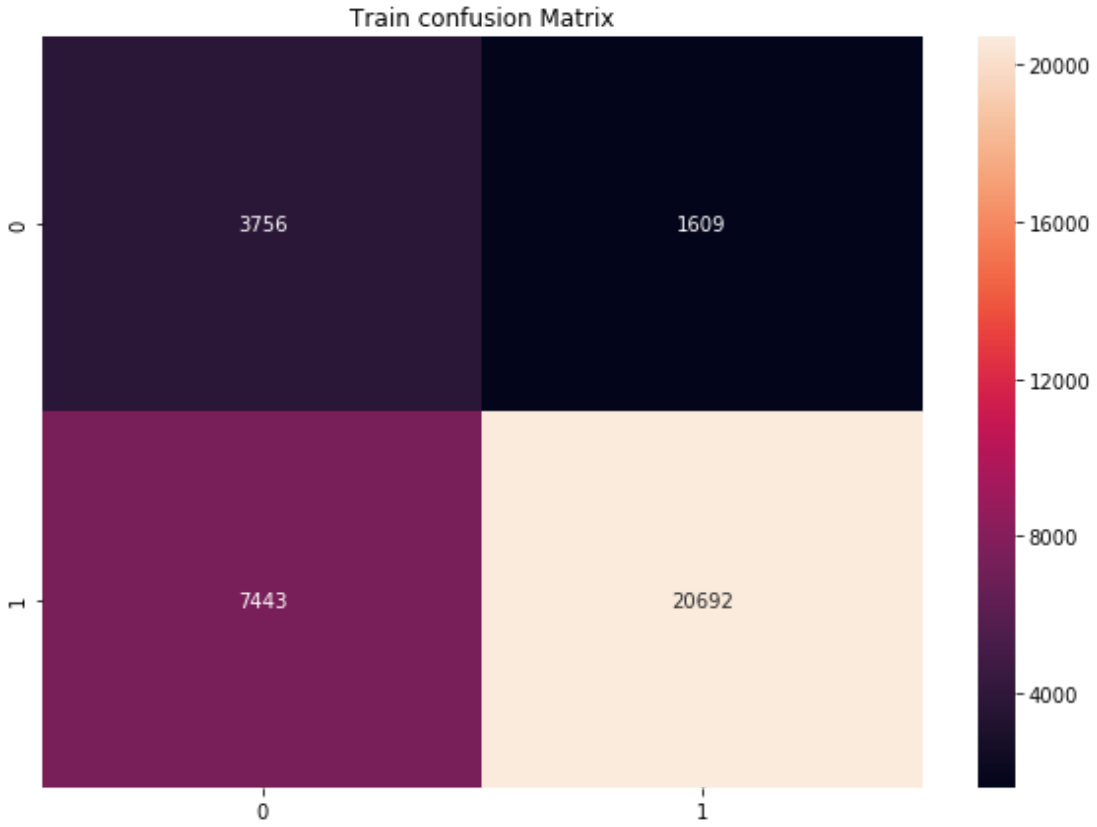


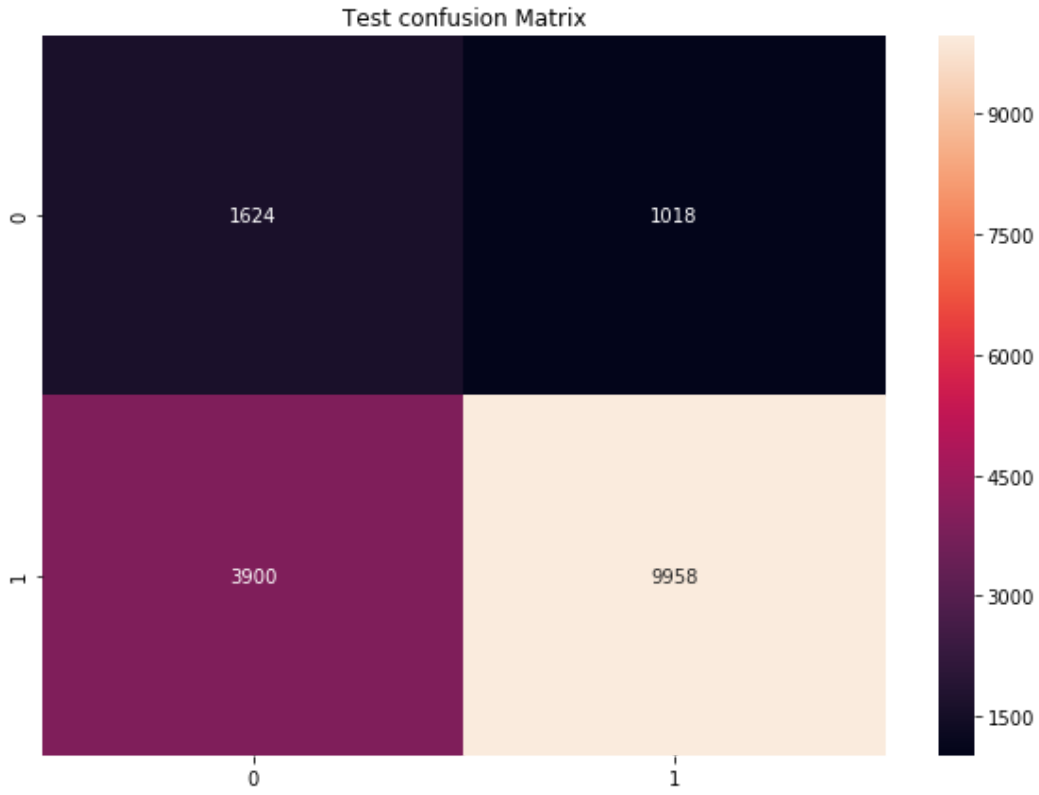
Confusion Matrix with predicted and original labels for TFIDF

In [56]:

```
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) 0.5148863843958308 for thresh
old -0.015
```

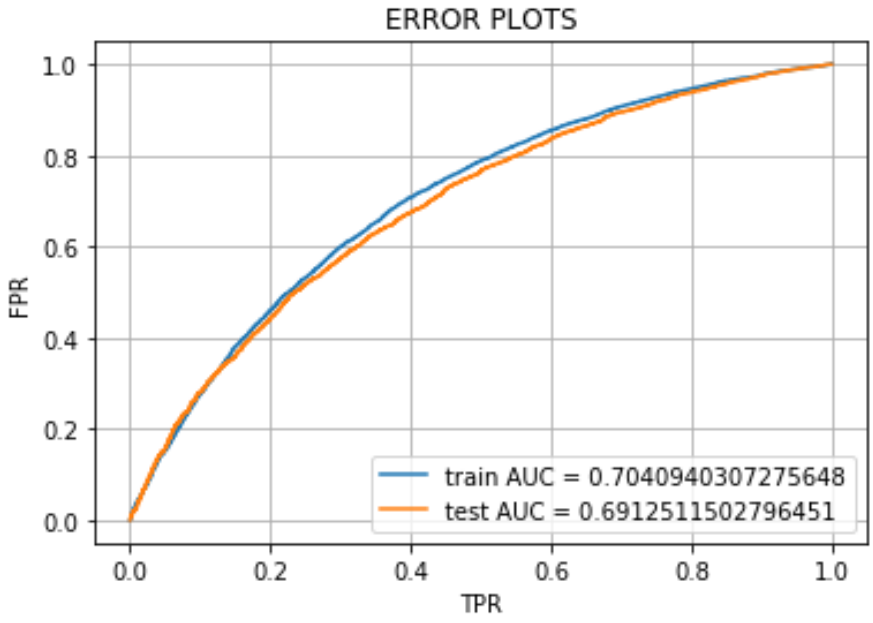




Applying SVM with obtained best Hyper parameter on AVG W2V representation

In [57]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve
# https://stackoverflow.com/a/57789235
#https://stackoverflow.com/a/56747024

from sklearn.metrics import roc_curve, auc
from sklearn.calibration import CalibratedClassifierCV

clf = linear_model.SGDClassifier(loss = 'hinge', alpha = 0.001, 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
calibrator = CalibratedClassifierCV(clf, cv='prefit')
model=calibrator.fit(X_train_aw2v_matrix, y_train)

y_train_pred = clf.decision_function (X_train_aw2v_matrix)
y_test_pred = clf.decision_function (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()
```

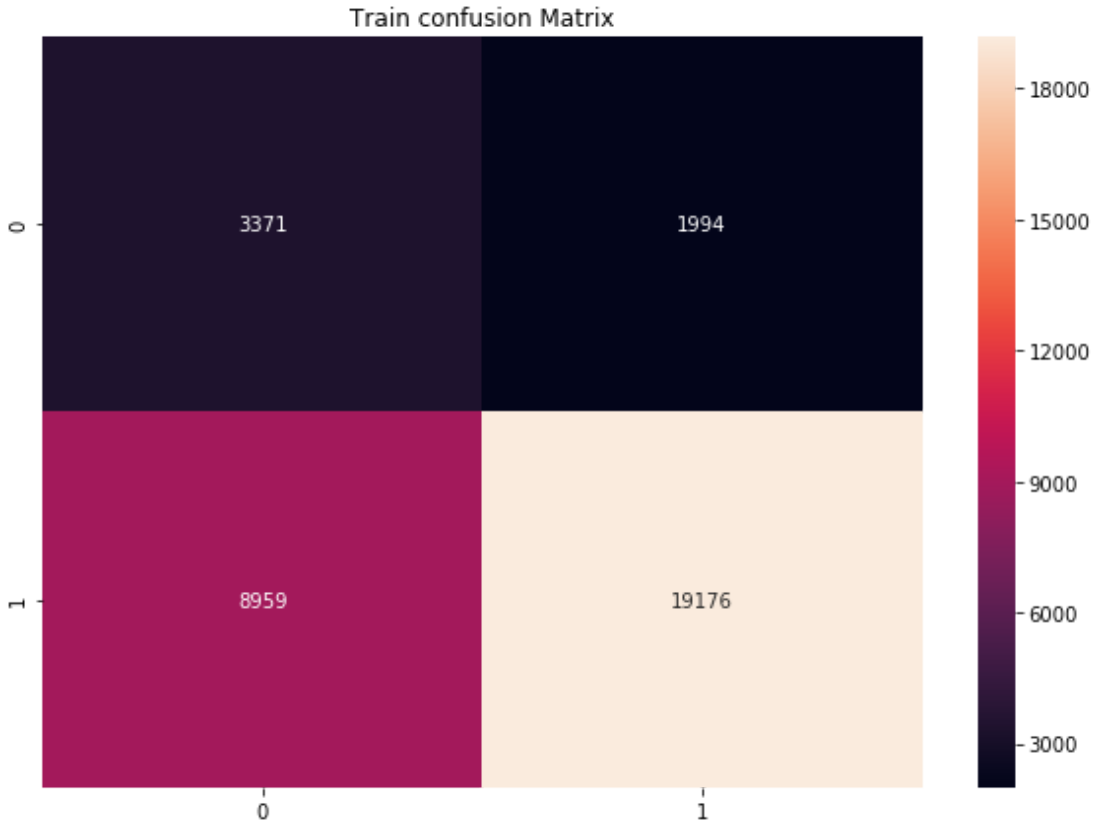


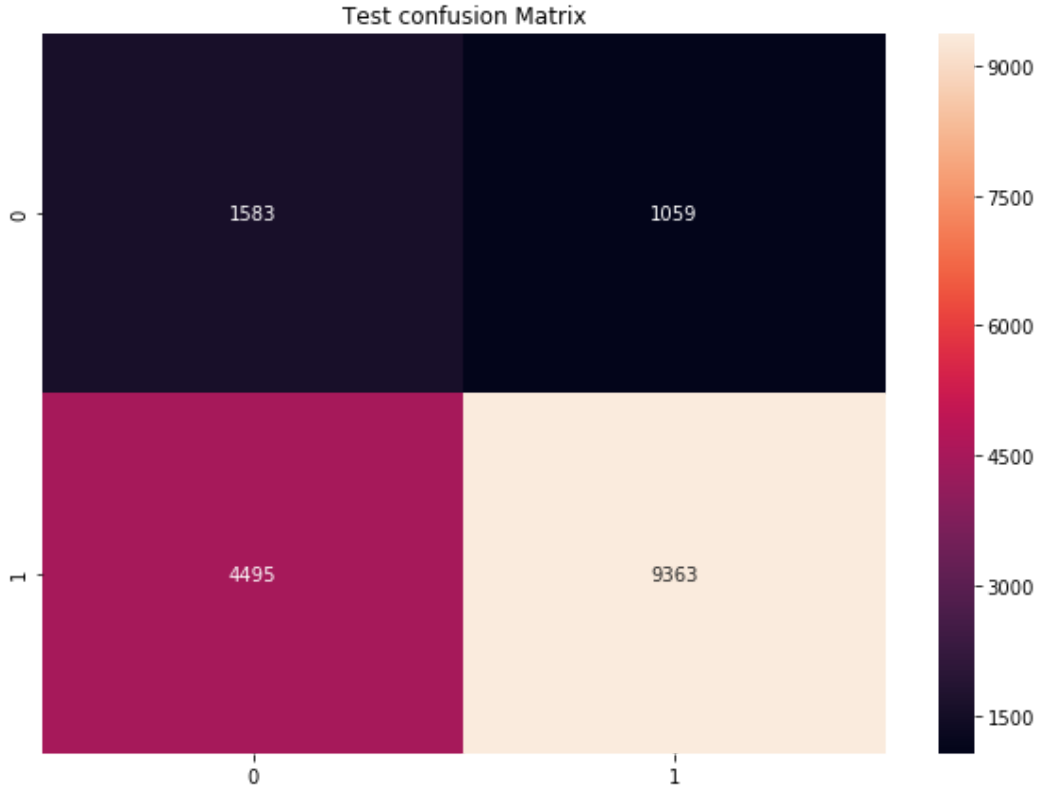
Confusion Matrix with predicted and original labels for AVG W2V

In [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) 0.42825271776620877 for thres
hold 0.071
```

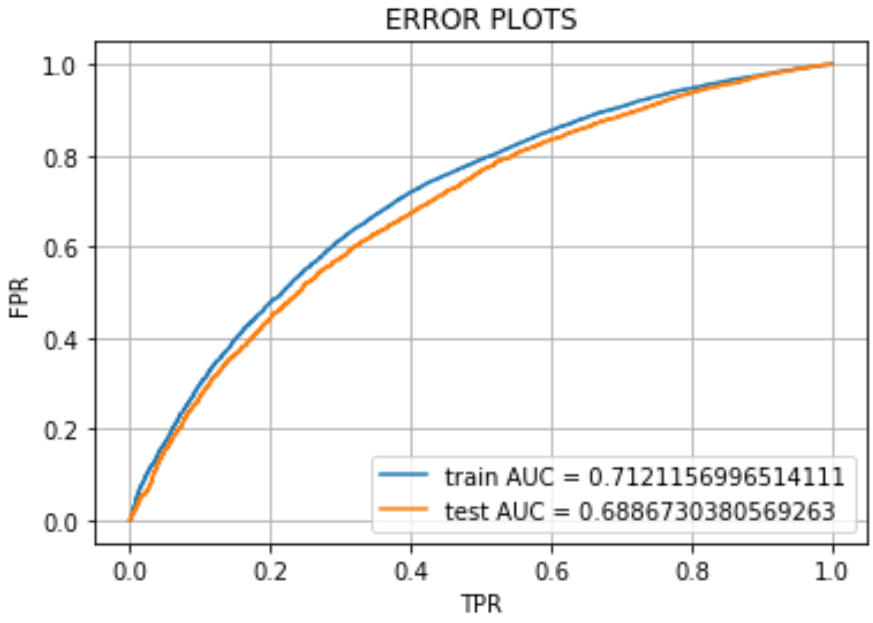




Applying SVM with obtained best Hyper parameter on TFIDF W2V representation

In [59]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve
# https://stackoverflow.com/a/57789235
#https://stackoverflow.com/a/56747024

from sklearn.metrics import roc_curve, auc
from sklearn.calibration import CalibratedClassifierCV

clf = linear_model.SGDClassifier(loss = 'hinge', alpha = 0.001, 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
calibrator = CalibratedClassifierCV(clf, cv='prefit')
model=calibrator.fit(X_train_tw2v_matrix, y_train)

y_train_pred = clf.decision_function (X_train_tw2v_matrix)
y_test_pred = clf.decision_function (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.xlabel("TPR")
plt.ylabel("FPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

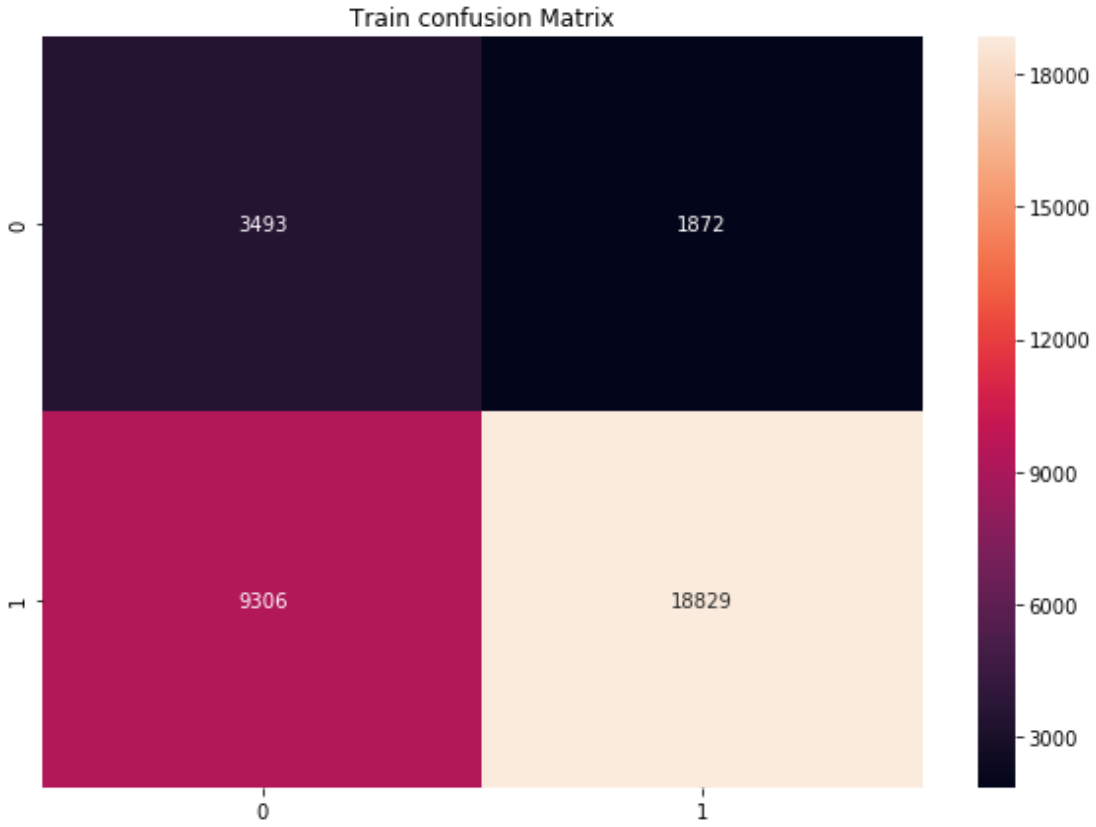


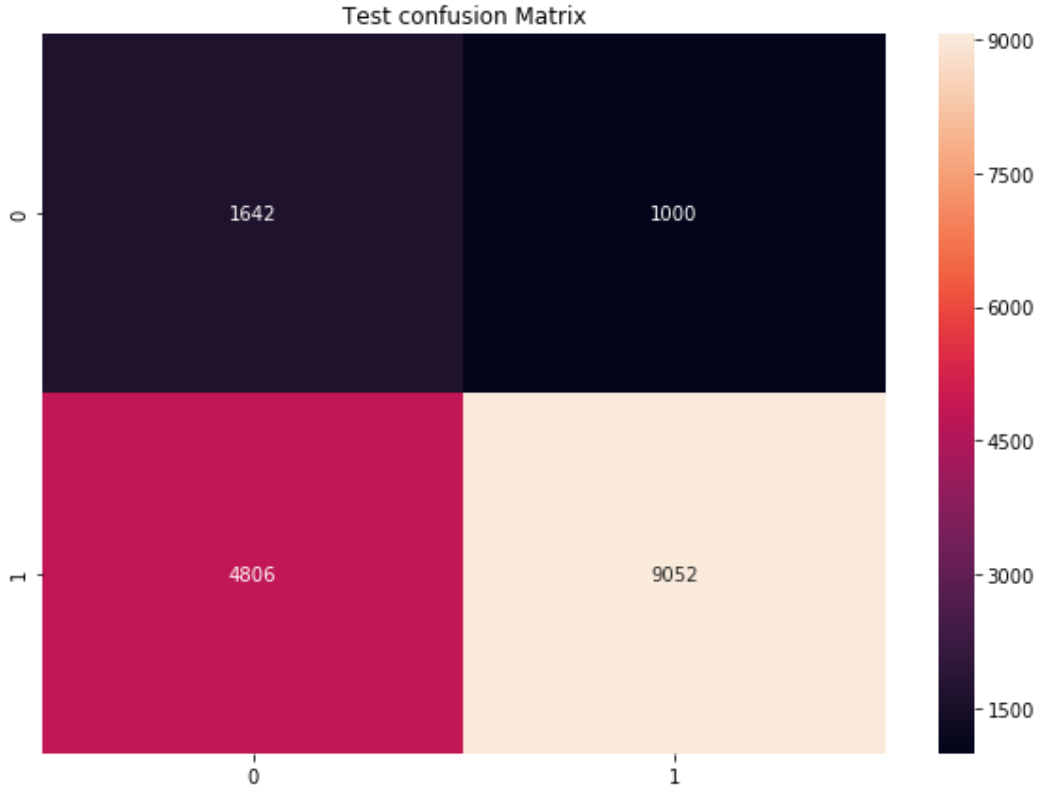
Confusion Matrix with predicted and original labels for TFIDF W2V

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) 0.43572170590769344 for thres
hold -0.369
```

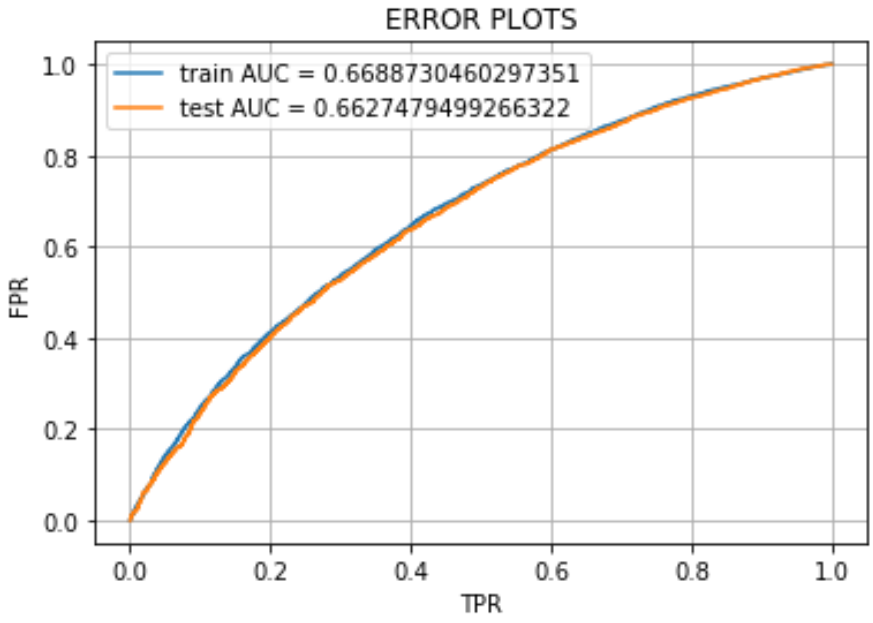




Applying SVM with obtained best Hyper parameter on TFIDF W2V representation with penalty='l1'

In [61]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve
# https://stackoverflow.com/a/57789235
#https://stackoverflow.com/a/56747024

from sklearn.metrics import roc_curve, auc
from sklearn.calibration import CalibratedClassifierCV

clf = linear_model.SGDClassifier(loss = 'hinge', penalty='l1', alpha = 0.001,
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
calibrator = CalibratedClassifierCV(clf, cv='prefit')
model=calibrator.fit(X_train_tw2v_matrix, y_train)

y_train_pred = clf.decision_function (X_train_tw2v_matrix)
y_test_pred = clf.decision_function (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.xlabel("TPR")
plt.ylabel("FPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

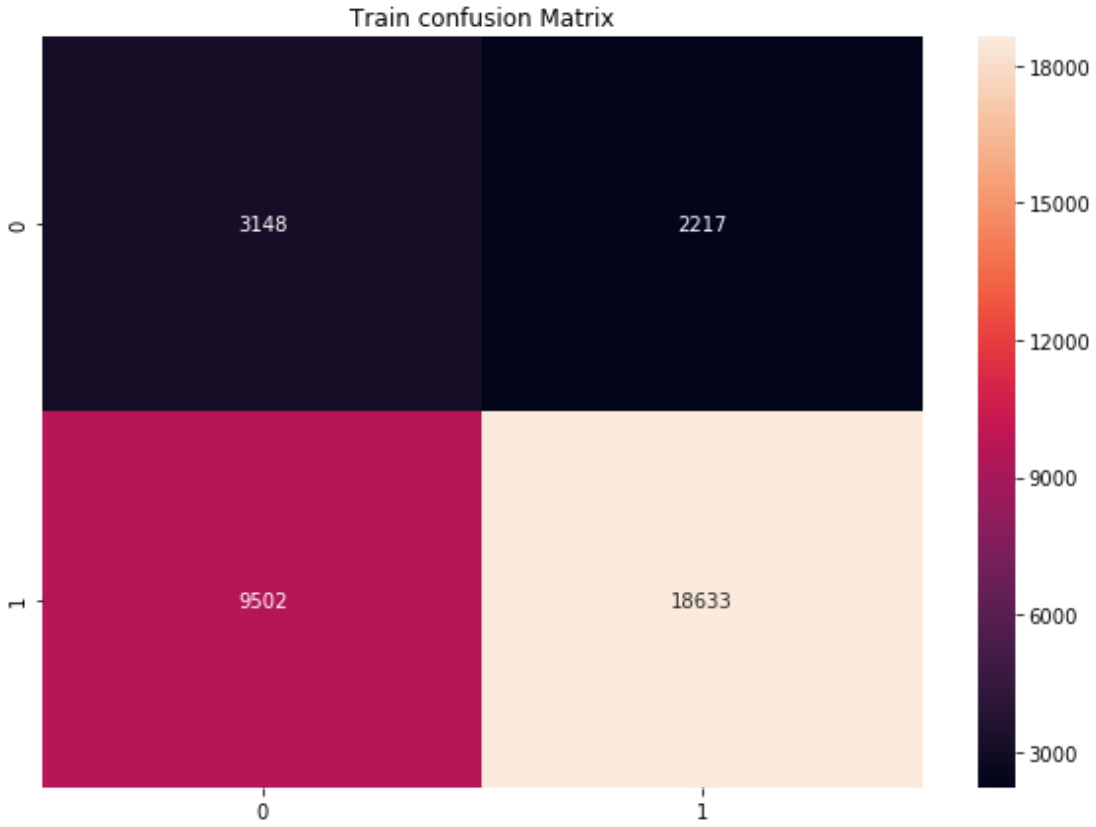


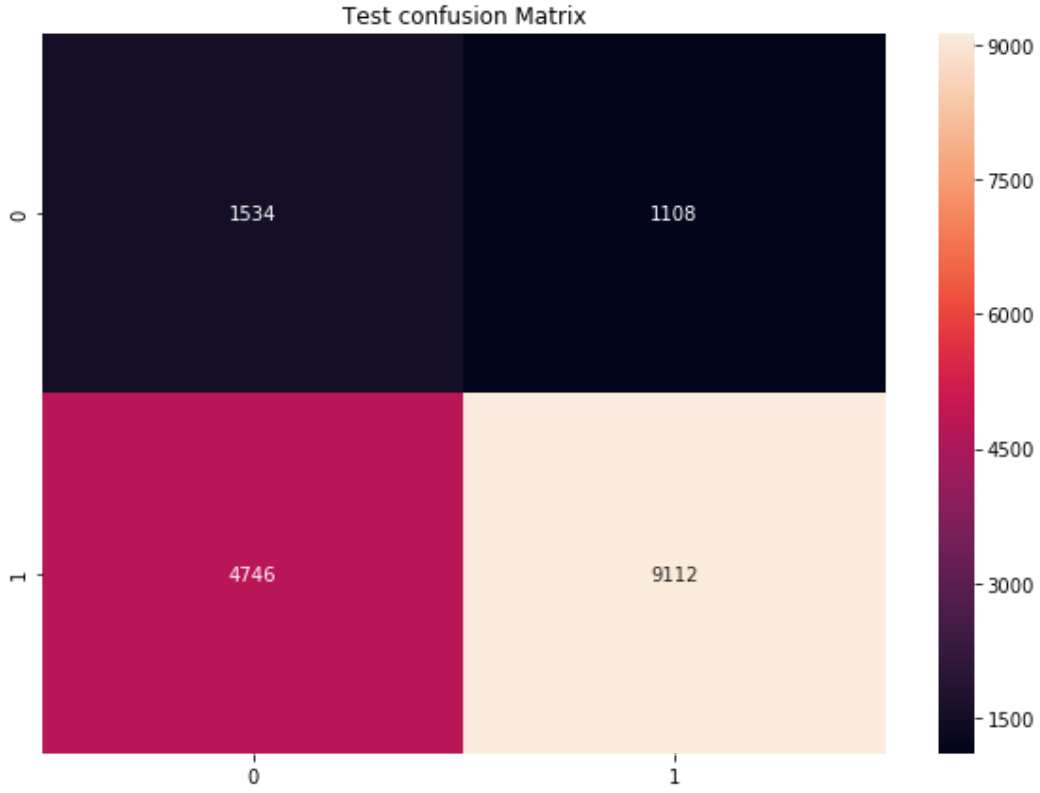
Confusion Matrix with predicted and original labels for TFIDF W2V with penalty='l1'

In [62]:

```
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) 0.3885982691294519 for thresh
old 0.861
```





Task-2

In [63]:

```
print(X_train_essay_tfidf.shape)
print(X_test_essay_tfidf.shape)
```

```
(33500, 5000)
(16500, 5000)
```

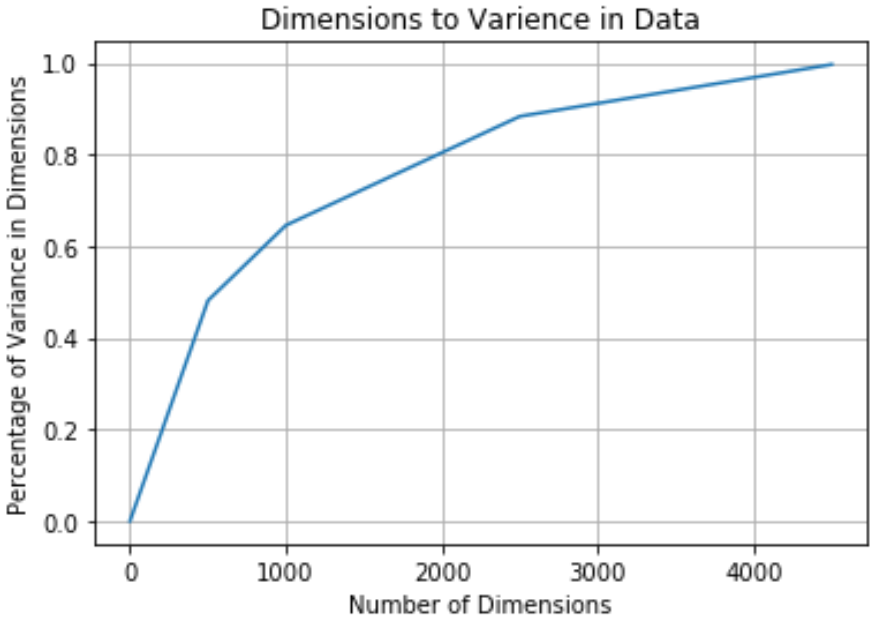
In [64]:

```
from sklearn.decomposition import TruncatedSVD
percentage=[]
interval = [0, 500, 1000, 2500, 4500]
for i in tqdm(interval):
    svd = TruncatedSVD(n_components = i)
    svd.fit(X_train_essay_tfidf)
    percentage.append(svd.explained_variance_ratio_.sum())
```

```
100%|██████████| 5/5 [13:23<00:00, 160.64s/it]
```

In [65]:

```
plt.xlabel("Number of Dimensions")
plt.ylabel("Percentage of Variance in Dimensions")
plt.title("Dimensions to Varience in Data")
plt.grid()
plt.plot(interval,percentage)
plt.show()
```



In [66]:

```
#Choosing n_componets = 3000 from above result
svd = TruncatedSVD(n_components= 3000)
svd.fit(X_train_essay_tfidf)
X_train_essay_tfidf= svd.transform(X_train_essay_tfidf)
X_test_essay_tfidf = svd.transform(X_test_essay_tfidf)
```

In [67]:

```
print(X_train_essay_tfidf.shape)
print(X_test_essay_tfidf.shape)
```

```
(33500, 3000)
(16500, 3000)
```

Matrix with all catagorical, numerical and 3000 TFIDF text

features

In [68]:

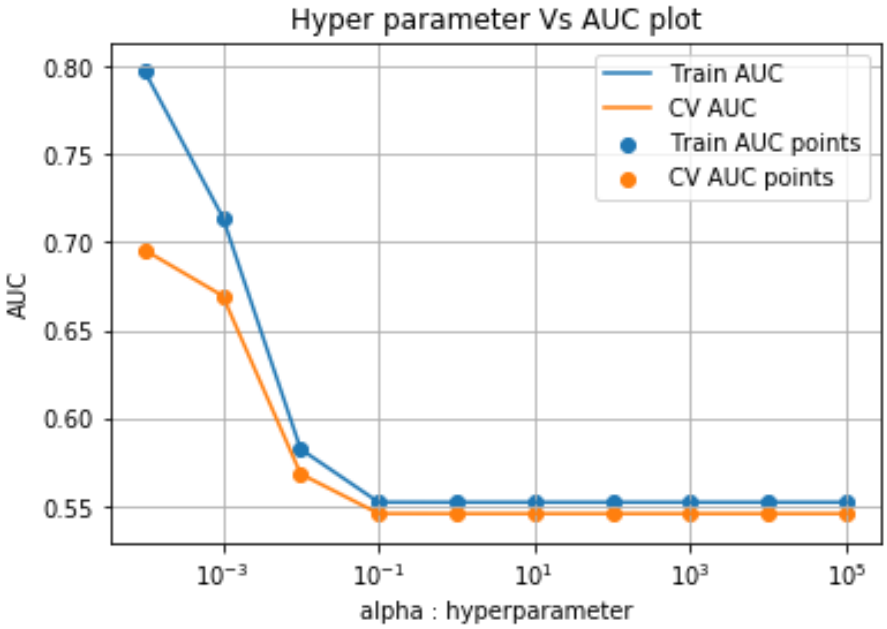
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

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,
                             X_train_ppp_norm, X_train_ewc_norm,
                             X_train_quantity_norm, X_train_ss_norm,
                             X_train_twc_norm, X_train_essay_tfidf)).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,
                            X_test_ppp_norm, X_test_ewc_norm,
                            X_test_quantity_norm, X_test_ss_norm,
                             X_test_twc_norm,X_test_essay_tfidf)).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
(33500, 3101) (33500,)
(16500, 3101) (16500,)
```

Finding Best Hyper parameter using K-Fold CV

In [69]:

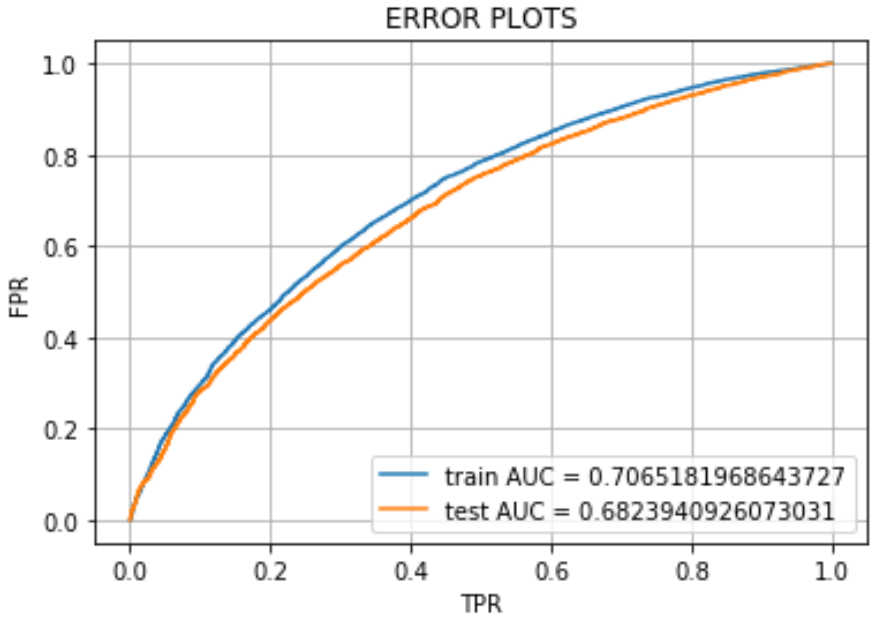
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.G
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt
from sklearn import linear_model
from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

parameters = {'alpha':[0.00001,0.0001,0.001,0.01,0.1,1,10,10**2,10**3,10**4,1
lg = linear_model.SGDClassifier(loss='hinge', 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()
```



Applying SVM with obtained best Hyper parameter

In [70]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve
# https://stackoverflow.com/a/57789235
#https://stackoverflow.com/a/56747024

from sklearn.metrics import roc_curve, auc
from sklearn.calibration import CalibratedClassifierCV

clf = linear_model.SGDClassifier(loss = 'hinge', alpha = 0.001, 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
calibrator = CalibratedClassifierCV(clf, cv='prefit')
model=calibrator.fit(X_train_num_matrix, y_train)

y_train_pred = clf.decision_function (X_train_num_matrix)
y_test_pred = clf.decision_function (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.xlabel("TPR")
plt.ylabel("FPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

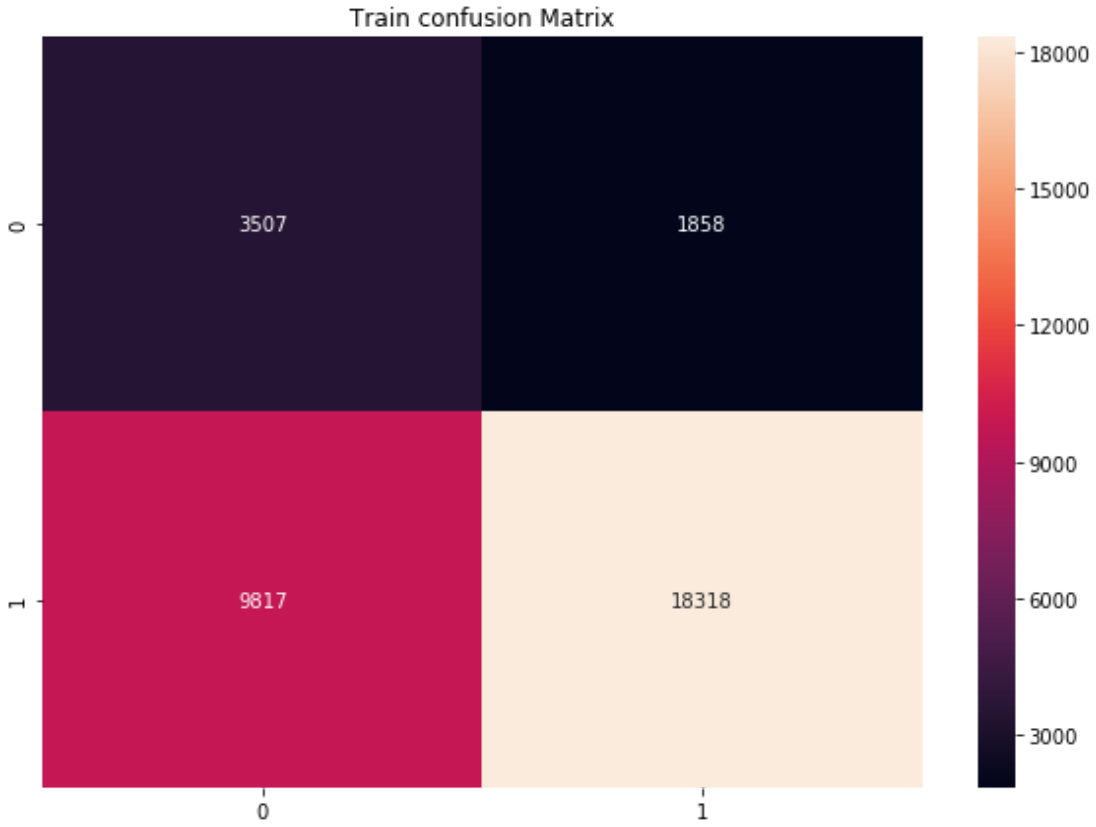


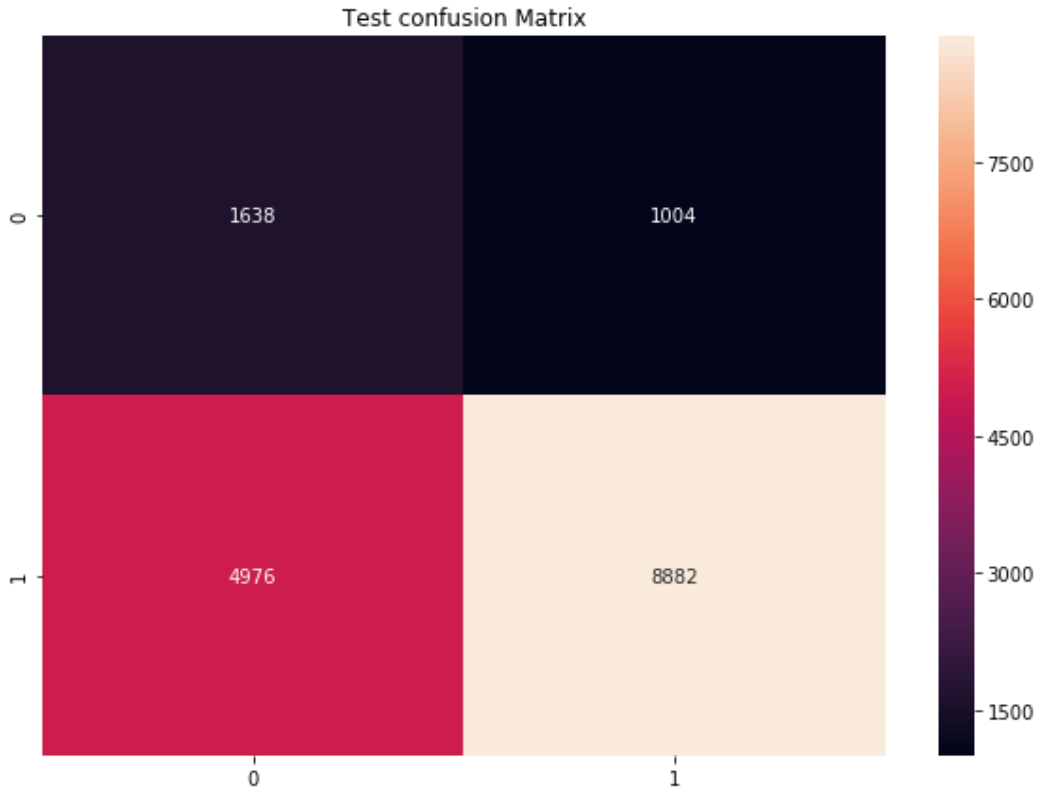
Confusion Matrix with predicted and original labels

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) 0.42559564448535725 for thres
hold 0.034
```

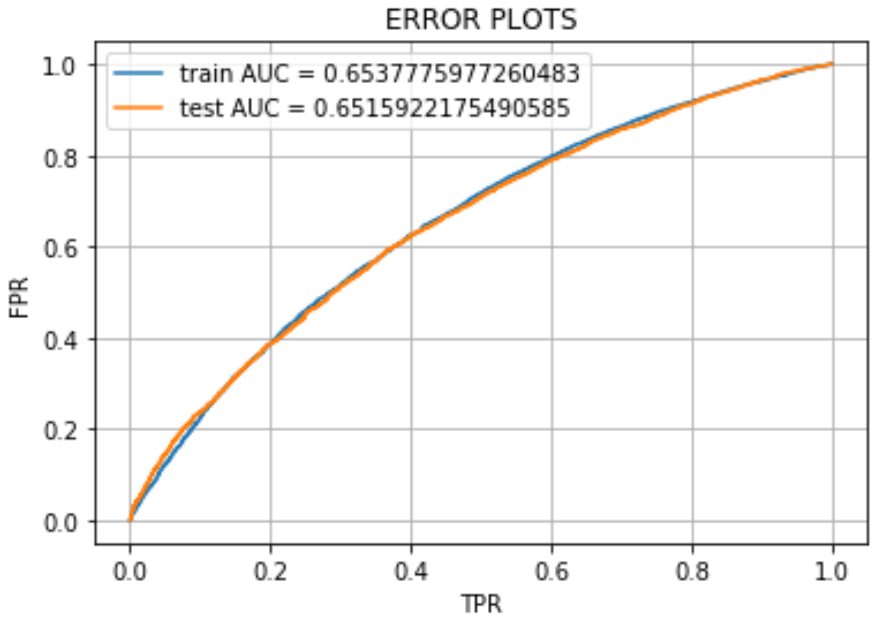




Applying SVM with obtained best Hyper parameter and with penalty='l1'

In [72]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve
# https://stackoverflow.com/a/57789235
#https://stackoverflow.com/a/56747024

from sklearn.metrics import roc_curve, auc
from sklearn.calibration import CalibratedClassifierCV

clf = linear_model.SGDClassifier(loss = 'hinge', penalty='l1', alpha = 0.001,
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
calibrator = CalibratedClassifierCV(clf, cv='prefit')
model=calibrator.fit(X_train_num_matrix, y_train)

y_train_pred = clf.decision_function (X_train_num_matrix)
y_test_pred = clf.decision_function (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.xlabel("TPR")
plt.ylabel("FPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

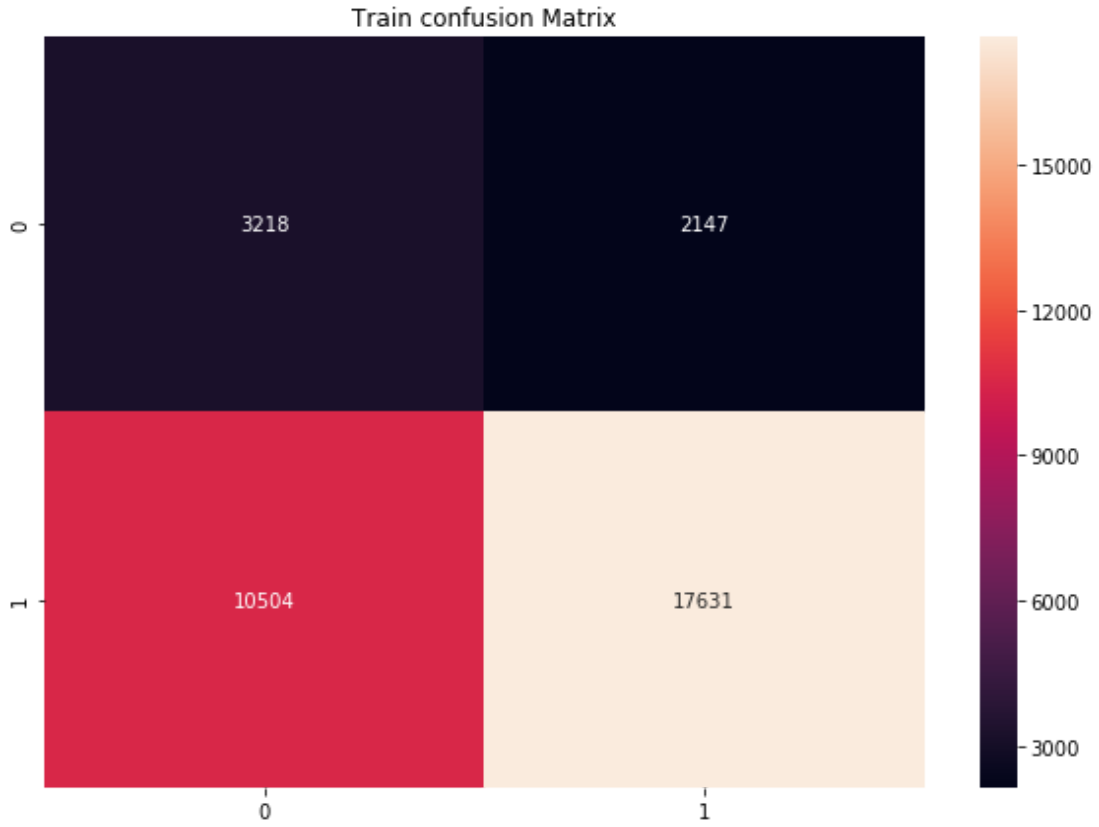


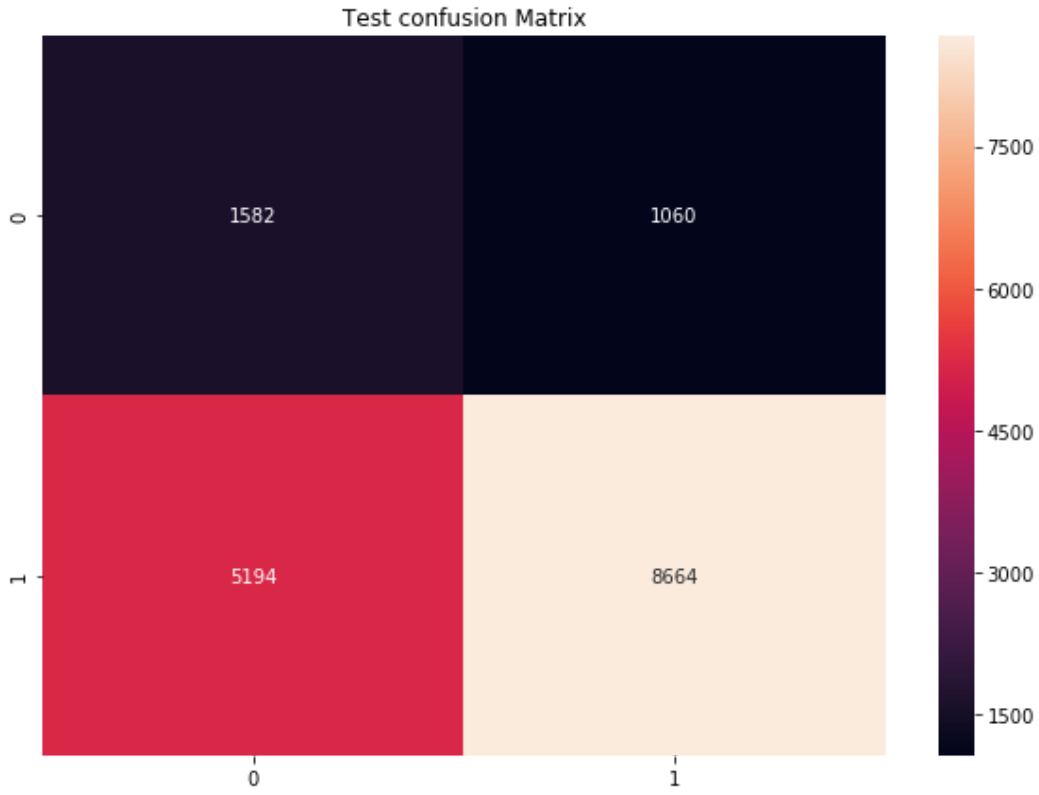
Confusion Matrix with predicted and original labels and with penalty='l1'

In [73]:

```
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) 0.3758775084381306 for thresh
old -0.018
```




Test confusion Matrix
1582
1060
5194
8664
0
1
0
1
7500
6000
4500
3000
1500

Conclusion

In [74]:

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

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

table.add_row(['BOW', 'Linear SVM', 0.1, 0.7277])
table.add_row(['TFIDF', 'Linear SVM', 0.001, 0.724])
table.add_row(['AVG W2V', 'Linear SVM', 0.001, 0.6912])
table.add_row(['TFIDF W2V', 'Linear SVM', 0.001, 0.6886])
table.add_row(['TFIDF W2V', 'Linear SVM, L1 reg', 0.001, 0.6627])
table.add_row(['TFIDF with 3K Features', 'Linear SVM', 0.001, 0.6823])
table.add_row(['TFIDF with 3K Features', 'Linear SVM, L1 reg', 0.001, 0.6515]
print(table)
```

```
+------------------------+--------------------+--------------
--+--------+
|       Vectorizer       |       Model        | Hyper Paramete
r |  AUC   |
+------------------------+--------------------+--------------
--+--------+
|          BOW           |     Linear SVM     |       0.1
| 0.7277 |
|         TFIDF          |     Linear SVM     |      0.001
| 0.724  |
|        AVG W2V         |     Linear SVM     |      0.001
| 0.6912 |
|       TFIDF W2V        |     Linear SVM     |      0.001
| 0.6886 |
|       TFIDF W2V        | Linear SVM, L1 reg |      0.001
| 0.6627 |
| TFIDF with 3K Features |     Linear SVM     |      0.001
| 0.6823 |
| TFIDF with 3K Features | Linear SVM, L1 reg |      0.001
| 0.6515 |
+------------------------+--------------------+--------------
--+--------+
```

Summary

- BOW vectorizer gave AUC 0.7277 with the best hyper parameter 0.1
- TFIDF vectorizer gave AUC 0.724 with the best hyper parameter 0.001
- AVG W2V vectorizer gave AUC 0.6912 with the best hyper parameter 0.001
- TFIDF W2V vectorizer gave AUC 0.6886 with the best hyper parameter 0.001
- TFIDF W2V vectorizer gave AUC 0.6627 with the best hyper parameter 0.001 and L1 Regularization

- TFIDF with 3K Features vectorizer gave AUC 0.6823 with the best hyper parameter 0.001
- TFIDF with 3K Features vectorizer gave AUC 0.6515 with the best hyper parameter 0.001 and L1 Regularization
- BOW vectorizer has the best AUC
- TFIDF, AVG W2V and TFIDF W2V has the next best AUC respectively
- TFIDF W2V's AUC dropped from 0.6886 to 0.6627 when L1 Regularization is used instead of L2 Regularization
- TFIDF with 3K Features AUC dropped from 0.6823 to 0.6515 when L1 Regularization is used instead of L2 Regularization