In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
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
```

In [3]:

```
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
```

In [4]:

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

In [5]:

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

In [6]:

```
from tqdm import tqdm
import os
```

In [7]:

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

In [8]:

```
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
```

1.1 Loading Data

In [9]:

```
#using the preprocessed data which was created in Logistic Regression assignment
data = pd.read_csv('preprocessed_data1.csv', nrows = 50000)
data.head(2)
```

Out[9]:

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcatego
0	0	mrs	in	grades_prek_2	literacy_language	esl_literacy
1	1	mr	fl	grades_6_8	history_civics_health_sports	civics_government_teamspo

In [10]:

```
print(data.columns.values)
print(data.shape)
```

```
['Unnamed: 0' 'teacher_prefix' 'school_state' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'teacher_number_of_previously_posted_projects' 'project_is_approved'
 'price' 'quantity' 'preprocessed_essays' 'preprocessed_titles']
(50000, 12)
```

In [11]:

```
y = data['project_is_approved'].values
x = data.drop(['project_is_approved'], axis = 1)
print(x.shape)
x.head(2)
```

```
(50000, 11)
```

Out[11]:

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcatego
0	0	mrs	in	grades_prek_2	literacy_language	esl_literacy
1	1	mr	fl	grades_6_8	history_civics_health_sports	civics_government_teamspo

1.2 Splitting data into Train and cross validation(or test): Stratified Sampling

In [12]:

```
#Train Test split (Using GridSearchCV)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size= 0.3, stratify = y)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(35000, 11)
(15000, 11)
(35000,)
(15000,)
```

1.3 Make Data Model Ready: encoding text features

Encoding Essay : TF-IDF

In [23]:

```
vectorizer = TfidfVectorizer(min_df = 10, ngram_range=(1,2), max_features=2000)
# fit should be done only on train data
vectorizer.fit(X_train['preprocessed_essays'].values)

X_train_essay_tfidf = vectorizer.transform(X_train['preprocessed_essays'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['preprocessed_essays'].values)

print("Shape after vectorization")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
```

```
Shape after vectorization
(35000, 2000) (35000,)
(15000, 2000) (15000,)
```

Encoding Essay : TFIDF W2V

In [16]:

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

In [14]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['preprocessed_essays'].values)
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [17]:

```
tfidf_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['preprocessed_essays'].values): # 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/review
    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
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_train.append(vector)

print(len(tfidf_w2v_vectors_train))
print(len(tfidf_w2v_vectors_train[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 35000/35000 [02:
30<00:00, 233.04it/s]
```

```
35000
300
```

In [18]:

```
tfidf_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_essays'].values): # 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/review
    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
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_test.append(vector)

print(len(tfidf_w2v_vectors_test))
print(len(tfidf_w2v_vectors_test[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 15000/15000 [01:
04<00:00, 233.74it/s]
```

```
15000
300
```

Encoding Project title : TFIDF

In [19]:

```
# check if we have any nan values are there
print(X_train['preprocessed_titles'].isnull().values.any())
print("number of nan values",X_train['preprocessed_titles'].isnull().values.sum())
print(X_test['preprocessed_titles'].isnull().values.any())
print("number of nan values",X_test['preprocessed_titles'].isnull().values.sum())
```

```
True
number of nan values 16
True
number of nan values 7
```

In [20]:

```
X_train['preprocessed_titles'].value_counts()
```

Out[20]:

```
wiggle work                           120
flexible seating                       94
like move move                         69
hear                                   64
wobble work                            48
                                     ...
miss 100 shots not take                 1
learning language modern technology     1
brain pop                               1
steam bins 5th graders                  1
different succeed                       1
Name: preprocessed_titles, Length: 31257, dtype: int64
```

In [21]:

```
# replacing missing values with literacy language project as title because
# this has most number of projects belong to this category
X_train['preprocessed_titles']=X_train['preprocessed_titles'].fillna('literacy language project')
print(X_train['preprocessed_titles'].isnull().values.any())
X_test['preprocessed_titles']=X_test['preprocessed_titles'].fillna('literacy language project')
print(X_test['preprocessed_titles'].isnull().values.any())
```

```
False
False
```

In [24]:

```
vectorizer = TfidfVectorizer(min_df = 10,ngram_range=(1,2), max_features=2000)
# fit should be done only on train data
vectorizer.fit(X_train['preprocessed_titles'].values)

X_train_titles_tfidf = vectorizer.transform(X_train['preprocessed_titles'].values)
X_test_titles_tfidf = vectorizer.transform(X_test['preprocessed_titles'].values)

print("Shape after vectorization")
print(X_train_titles_tfidf.shape, y_train.shape)
print(X_train_titles_tfidf.shape, y_test.shape)
```

```
Shape after vectorization
(35000, 2000) (35000,)
(35000, 2000) (15000,)
```

Encoding Project title : TFIDF W2V

In [25]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['preprocessed_titles'].values)
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [26]:

```
tfidf_w2v_vectors_train_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['preprocessed_titles'].values): # 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/review
    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
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_train_titles.append(vector)

print(len(tfidf_w2v_vectors_train_titles))
print(len(tfidf_w2v_vectors_train_titles[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████████| 35000/35000
[00:01<00:00, 18557.22it/s]
```

```
35000
300
```

In [27]:

```
tfidf_w2v_vectors_test_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_titles'].values): # 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/review
    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
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_test_titles.append(vector)

print(len(tfidf_w2v_vectors_test_titles))
print(len(tfidf_w2v_vectors_test_titles[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 15000/15000
[00:00<00:00, 17862.98it/s]
```

```
15000
300
```

1.4 Make Data Model Ready: encoding numerical, categorical features

encoding categorical features: School State

In [30]:

```
#onehot encoding
vectorizer = CountVectorizer()
#fit has to only happen on train data
vectorizer.fit(X_train['school_state'].values)
print(vectorizer.get_feature_names())

#we use the fitted CountVectorizer to convert the text to a vector
X_train_state_ohe = vectorizer.transform(X_train['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_test_state_ohe.shape, y_test.shape)
```

```
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm',
'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv
', 'wy']
After Vectorizations:
(35000, 51) (35000,)
(15000, 51) (15000,)
```

encoding categorical features: teacher_prefix

In [31]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer.transform(X_train['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_test_teacher_ohe.shape, y_test.shape)
```

```
['dr', 'mr', 'mrs', 'ms', 'teacher']
After vectorizations
(35000, 5) (35000,)
(15000, 5) (15000,)
```

encoding categorical features: project_grade_category

In [32]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only on train data
print(vectorizer.get_feature_names())

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


print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_test_grade_ohe.shape, y_test.shape)
```

```
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
After vectorizations
(35000, 4) (35000,)
(15000, 4) (15000,)
```

encoding categorical features: project_subject_categories

In [36]:

```
vectorizer = CountVectorizer(binary=True)
vectorizer.fit(X_train['project_subject_categories'].values) # fit has to happen only on train
data

# we use the fitted CountVectorizer to convert the text to vector
X_train_project_categories_ohe = vectorizer.transform(X_train['project_subject_categories'].values
)
X_test_project_categories_ohe = vectorizer.transform(X_test['project_subject_categories'].values)

print("After vectorizations")
print(X_train_project_categories_ohe.shape, y_train.shape)
print(X_test_project_categories_ohe.shape, y_test.shape)
```

```
After vectorizations
(35000, 49) (35000,)
(15000, 49) (15000,)
```

encoding categorical features: project_subject_subcategories

In [37]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_project_subcategories_ohe = vectorizer.transform(X_train['project_subject_subcategories'].
values)
X_test_project_subcategories_ohe = vectorizer.transform(X_test['project_subject_subcategories'].va
lues)

print("After vectorizations")
print(X_train_project_subcategories_ohe.shape, y_train.shape)
```

```
print(X_test_project_subcategories_ohe.shape, y_test.shape)
```

```
After vectorizations
(35000, 365) (35000,)
(15000, 365) (15000,)
```

encoding numerical features: price

In [38]:

```
print(X_train['price'].value_counts())
```

```
149.99    166
479.00    163
399.99    161
269.99    138
49.99     126
         ...
51.00       1
75.42       1
213.34      1
407.71      1
278.53      1
Name: price, Length: 22659, dtype: int64
```

In [39]:

```
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,-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)
```

```
After vectorizations
(1, 35000) (35000,)
(1, 15000) (15000,)
```

In [40]:

```
#print(X_train_price_norm)
X_train_price_norm= X_train_price_norm.reshape(-1, 1)
X_test_price_norm = X_test_price_norm.reshape(-1, 1)
print(X_train_price_norm.shape)
print(X_test_price_norm.shape)
```

```
(35000, 1)
(15000, 1)
```

teacher_number_of_previously_posted_projects

In [41]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))

X_train_teacher_number_of_previously_posted_projects_norm =
normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
X_test_teacher_number_of_previously_posted_projects_norm =
normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))

print("After vectorizations")
```

```
print(X_train_teacher_number_of_previously_posted_projects_norm.shape, y_train.shape)
print(X_test_teacher_number_of_previously_posted_projects_norm.shape, y_test.shape)
```

```
After vectorizations
(1, 35000) (35000,)
(1, 15000) (15000,)
```

In [42]:

```
print(X_train_teacher_number_of_previously_posted_projects_norm)
```

```
[[0.00017447 0.         0.00017447 ... 0.         0.         0.00034893]]
```

In [43]:

```
#reshape
X_train_teacher_number_of_previously_posted_projects_norm=
X_train_teacher_number_of_previously_posted_projects_norm.reshape(-1, 1)
X_test_teacher_number_of_previously_posted_projects_norm =
X_test_teacher_number_of_previously_posted_projects_norm.reshape(-1, 1)
print(X_train_teacher_number_of_previously_posted_projects_norm.shape)
print(X_test_teacher_number_of_previously_posted_projects_norm.shape)
```

```
(35000, 1)
(15000, 1)
```

Assignment 8: DT

Set 1: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)

In [44]:

```
from scipy.sparse import hstack
X_tr = hstack((X_train_titles_tfidf, X_train_essay_tfidf,
X_train_teacher_number_of_previously_posted_projects_norm, X_train_price_norm,
X_train_project_categories_ohe, X_train_project_subcategories_ohe, X_train_grade_ohe,
X_train_teacher_ohe, X_train_state_ohe)).tocsr()
X_te = hstack((X_test_titles_tfidf, X_test_essay_tfidf,
X_test_teacher_number_of_previously_posted_projects_norm, X_test_price_norm,
X_test_project_categories_ohe, X_test_project_subcategories_ohe, X_test_grade_ohe,
X_test_teacher_ohe, X_test_state_ohe)).tocsr()

print("Final Data Matrix:")
print(X_tr.shape, y_train.shape)
print(X_te.shape, y_test.shape)
```

```
Final Data Matrix:
(35000, 4476) (35000,)
(15000, 4476) (15000,)
```

Hyperparameter tuning using GridSearchCV

In [47]:

```
import math
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(class_weight = 'balanced')
parameters = {'max_depth':[1,5,10,50], 'min_samples_split':[5,10,100,500]}
clf = GridSearchCV(dt, parameters, cv=5, scoring='roc_auc', return_train_score = True)
clf.fit(X_tr, y_train)
```

Out[47]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0,
                                              class_weight='balanced',
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=None,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [1, 5, 10, 50],
                         'min_samples_split': [5, 10, 100, 500]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [55]:

```
parameters['max_depth']
```

Out[55]:

```
[1, 5, 10, 50]
```

In [56]:

```
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
# hyperparamters =  clf.cv_results_['get_params']
# print(hyperparamters)

# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=parameters['min_samples_split'], y=parameters['max_depth'], z=train_auc, na
me = 'train')
trace2 = go.Scatter3d(x=parameters['min_samples_split'], y=parameters['max_depth'], z=cv_auc, name
= 'Cross validation')
data = [trace1, trace2]

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

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

In [60]:

```
print(clf.best_estimator_)
print(clf.best_score_)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', criterion='gini',
                       max_depth=10, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=500,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
0.6381536223723724
```

* Best max_depth = 10 and the best min_samples_split = 500

Testing the performance of the model on test data, plotting ROC Curves

In [69]:

```
dt = DecisionTreeClassifier(class_weight = 'balanced', max_depth = 10, min_samples_split = 500)
dt.fit(X_tr, y_train)

y_train_pred = dt.predict_proba(X_tr)[:,1]
y_test_pred = dt.predict_proba(X_te)[:,1]
```

In [70]:

```
print(y_train_pred.shape)
y_train_pred
```

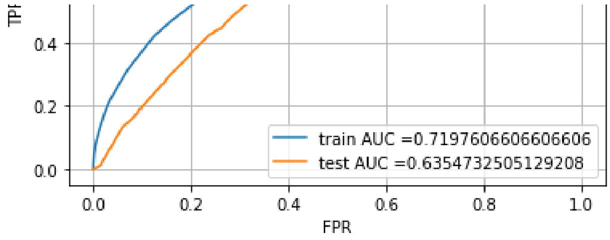
```
(35000,)
```

Out[70]:

```
array([0.39249947, 0.85648982, 0.96084138, ..., 0.32274137, 0.72219615,
       0.39249947])
```

In [72]:

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



- Using Decision Tree algo with max_depth of 10 and min_samples_split of 500, the test-auc is 0.635.

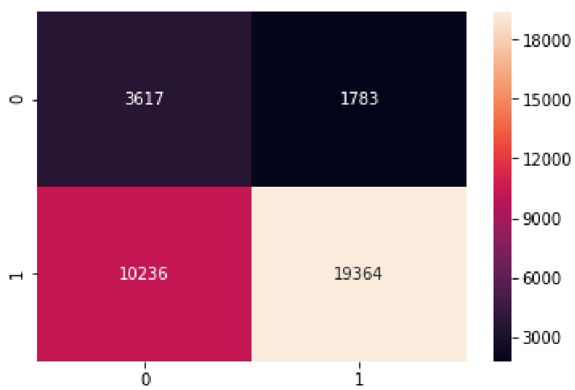
In [73]:

```
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 high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    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
```

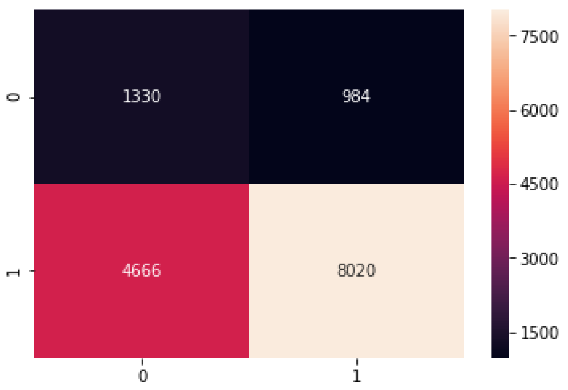
In [83]:

```
from sklearn.metrics import confusion_matrix
fig = plt.figure()
ax = fig.add_subplot(111)
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print('-'*75)
print("Train confusion matrix")
predictions = predict_with_best_t(y_test_pred, best_t)
cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax)


fig = plt.figure()
ax1 = fig.add_subplot(111)
print('-'*75)
print("Test confusion matrix")
cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax1)
```

```
the maximum value of tpr*(1-fpr) 0.4381856106106107 for threshold 0.499
---------------------------------------------------------------------------
Train confusion matrix
```



```
--------------------------------------------------------------------------------
Test confusion matrix
```



In [75]:

```python
#https://towardsdatascience.com/demystifying-confusion-matrix-confusion-9e82201592fd
tn, fp, fn, tp = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)).ravel()
print("True Negatives: ",tn)
print("False Positives: ",fp)
print("False Negatives: ",fn)
print("True Positives: ",tp)
```

```
True Negatives:  1330
False Positives:  984
False Negatives:  4666
True Positives:  8020
```

Word Cloud with with the words of essay text of these false positive data points

In [85]:

```python
#getting the indices of the False positives
fpi = []
for i in range(len(y_test)):
    if (y_test[i]==0) & (predictions[i]==1):
        fpi.append(i)

print(len(fpi))
```

```
984
```

In [95]:

```python
print(X_test.columns.values)
```

```
['Unnamed: 0' 'teacher_prefix' 'school_state' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'teacher_number_of_previously_posted_projects' 'price' 'quantity'
 'preprocessed_essays' 'preprocessed_titles']
```

In [96]:

```python
#getting the data from the indices
essay_fp = []
for i in fpi:
    essay_fp.append(X_test['preprocessed_essays'].values[i])

print(len(essay_fp))
```

```
984
```

In [101]:

```
#https://www.geeksforgeeks.org/generating-word-cloud-python/
from wordcloud import WordCloud, STOPWORDS
comment_words = ' '
stopwords = set(STOPWORDS)
for val in essay_fp:
    val = str(val)
    tokens = val.split()
for i in range(len(tokens)):
    tokens[i] = tokens[i].lower()
for words in tokens:
    comment_words = comment_words + words + ' '
wordcloud = WordCloud(width=800, height = 800, background_color='white', stopwords = stopwords, min
_font_size = 10).generate(comment_words)

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

plt.show()
```

box plot with the price of these false positive data points

In [109]:

```
X_test_fp = pd.DataFrame(X_test['price'])
X_test_fp.shape
```

Out[109]:

```
(15000, 1)
```

In [111]:

```
X_test_fp = X_test_fp.iloc[fpi,:]
X_test_fp.shape
```

Out[111]:

```
(984, 1)
```

In [112]:

```
X_test_fp.head(2)
```

Out[112]:

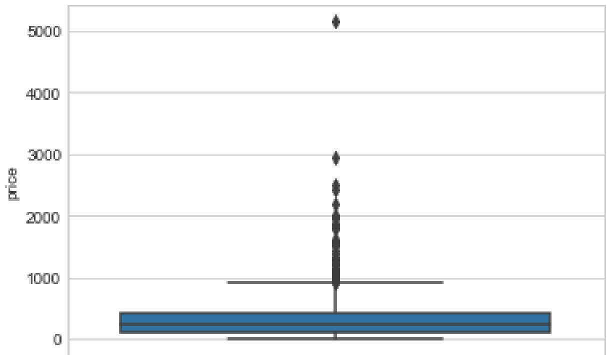
	price
490	410.43
20693	443.16

In [115]:

```
import seaborn as sns
sns.set_style('whitegrid')
sns.boxplot(x='price', data = X_test_fp, orient='v')
```

Out[115]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x27febe5dd68>
```



Most of the incorrectly accepted projects were of the price-range<1000

pdf with the teacher_number_of_previously_posted_projects of these false positive data points

In [118]:

```
X_test_fp_tnpp = pd.DataFrame(X_test['teacher_number_of_previously_posted_projects'])
print(X_test_fp_tnpp.shape)
X_test_fp_tnpp = X_test_fp_tnpp.iloc[fpi,:]
print(X_test_fp_tnpp.shape)
```

```
(15000, 1)
(984, 1)
```

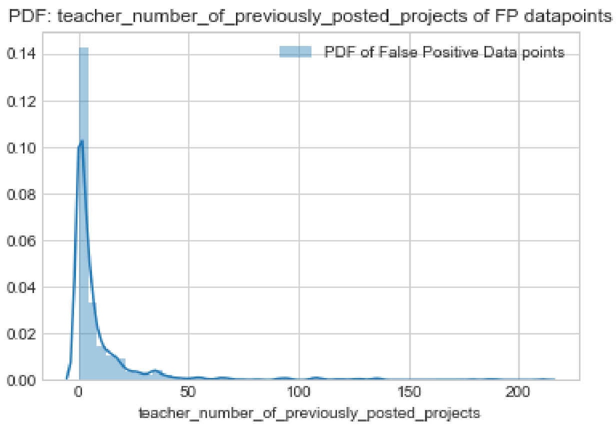
In [119]:

```
sns.set_style('whitegrid')
sns.distplot(X_test_fp_tnpp.values, label='PDF of False Positive Data points')
plt.title('PDF: teacher_number_of_previously_posted_projects of FP datapoints')
plt.xlabel('teacher_number_of_previously_posted_projects')
plt.legend()
plt.show
```

```
C:\Users\syeda\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning:
The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
```

Out[119]:

```
<function matplotlib.pyplot.show(*args, **kw)>
```



From the above pdf, it looks like most of the incorrectly accepted projects(FP data) had teacher_number_of_previously_posted_projects value less than 15

Set 2: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

In [125]:

```python
from scipy.sparse import hstack
X_tr2 = hstack((tfidf_w2v_vectors_train_titles, tfidf_w2v_vectors_train,
X_train_teacher_number_of_previously_posted_projects_norm, X_train_price_norm,
X_train_project_categories_ohe, X_train_project_subcategories_ohe, X_train_grade_ohe,
X_train_teacher_ohe, X_train_state_ohe)).tocsr()
X_te2 = hstack((tfidf_w2v_vectors_test_titles, tfidf_w2v_vectors_test,
X_test_teacher_number_of_previously_posted_projects_norm, X_test_price_norm,
X_test_project_categories_ohe, X_test_project_subcategories_ohe, X_test_grade_ohe,
X_test_teacher_ohe, X_test_state_ohe)).tocsr()

print("Final Data Matrix:")
print(X_tr2.shape, y_train.shape)
print(X_te2.shape, y_test.shape)
```

```
Final Data Matrix:
(35000, 1076) (35000,)
(15000, 1076) (15000,)
```

In [122]:

```python
import math
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(class_weight = 'balanced')
parameters = {'max_depth':[1,5,10,50], 'min_samples_split':[5,10,100,500]}
clf = GridSearchCV(dt, parameters, cv=5, scoring='roc_auc', return_train_score = True)
clf.fit(X_tr2, y_train)
```

Out[122]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0,
                                              class_weight='balanced',
                                              criterion='gini', max_depth=None,
```

```
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=None,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [1, 5, 10, 50],
                         'min_samples_split': [5, 10, 100, 500]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [123]:

```
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
# hyperparamters =  clf.cv_results_['get_params']
# print(hyperparamters)

# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=parameters['min_samples_split'], y=parameters['max_depth'], z=train_auc, na
me = 'train')
trace2 = go.Scatter3d(x=parameters['min_samples_split'], y=parameters['max_depth'], z=cv_auc, name
= 'Cross validation')
data = [trace1, trace2]

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

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

In [124]:

```
print(clf.best_estimator_)
```

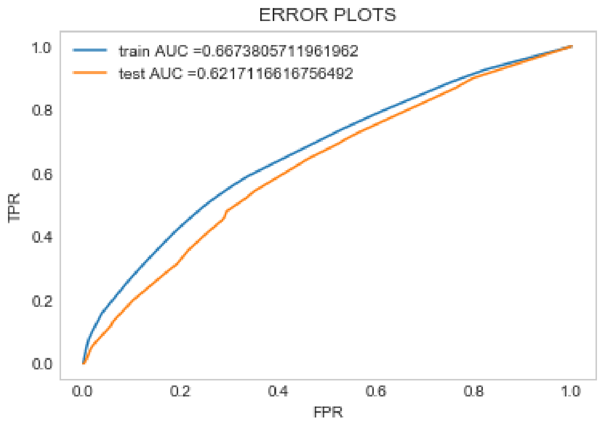
```
print(clf.best_score_)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', criterion='gini',
                       max_depth=5, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=500,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
0.622131131131131
```

* Best Hyperparameter values: max_depth=5, min_samples_split=500

In [126]:

```
dt = DecisionTreeClassifier(class_weight = 'balanced', max_depth = 5, min_samples_split = 500)
dt.fit(X_tr2, y_train)

y_train_pred = dt.predict_proba(X_tr2)[:,1]
y_test_pred = dt.predict_proba(X_te2)[:,1]

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

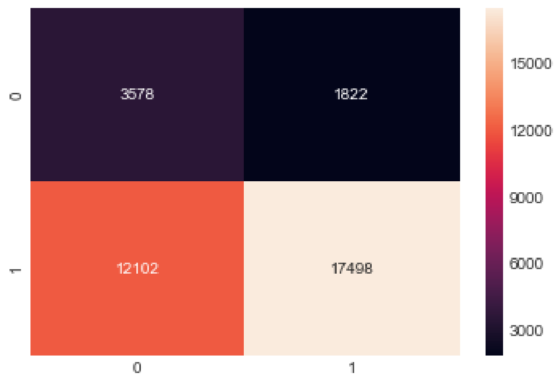


- using decision tree algo with max_depth=5 and min_samples_split=500, the test-auc is 0.6217

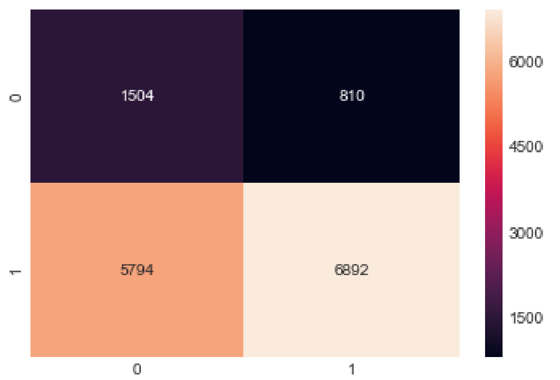
In [127]:

```
from sklearn.metrics import confusion_matrix
fig = plt.figure()
ax = fig.add_subplot(111)
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print('-'*75)
print("Train confusion matrix")
predictions = predict_with_best_t(y_test_pred, best_t)
cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax)


fig = plt.figure()
ax1 = fig.add_subplot(111)
print('-'*75)
print("Test confusion matrix")
cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax1)
```

```
the maximum value of tpr*(1-fpr) 0.3916907157157158 for threshold 0.49
------------------------------------------------------------------------------
Train confusion matrix
```




```
------------------------------------------------------------------------------
Test confusion matrix
```



In [128]:

```
tn, fp, fn, tp = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)).ravel()
print("True Negatives: ",tn)
print("False Positives: ",fp)
print("False Negatives: ",fn)
print("True Positives: ",tp)
```

```
True Negatives:  1504
False Positives:  810
False Negatives:  5794
True Positives:  6892
```

In [129]:

```
#getting the indices of the False positives
fpi = []
for i in range(len(y_test)):
    if (y_test[i]==0) & (predictions[i]==1):
        fpi.append(i)


print(len(fpi))
```

```
810
```

In [130]:

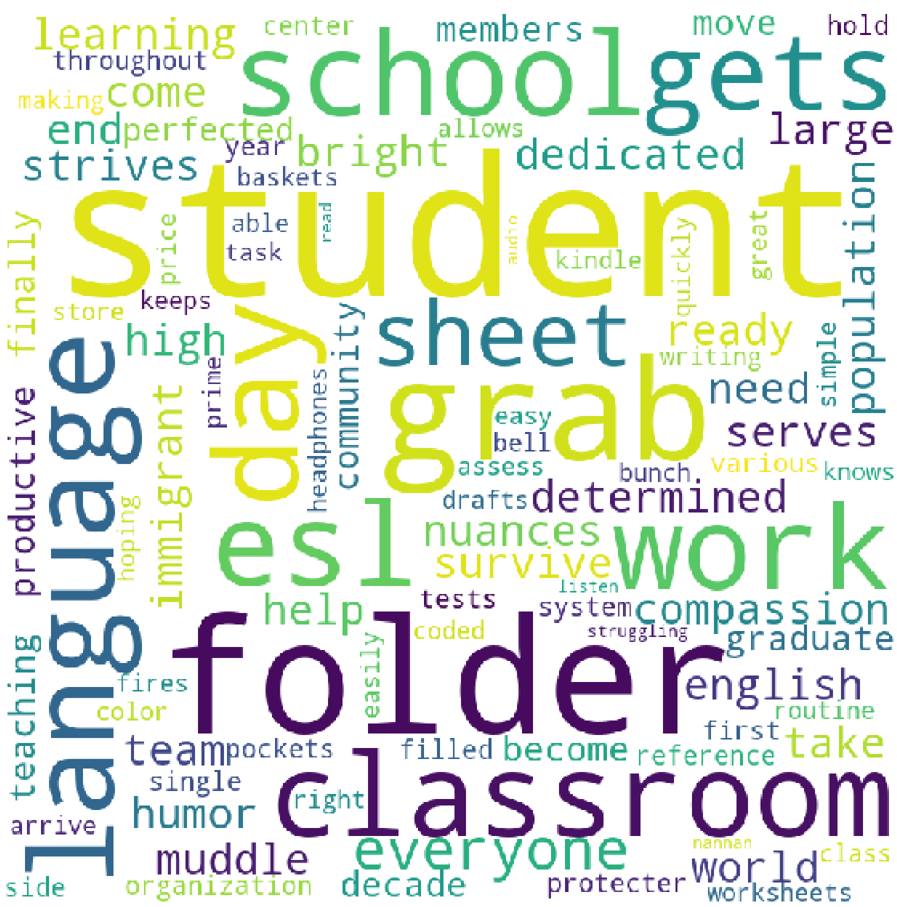
```
#getting the data from the indices
essay_fp = []
for i in fpi:
    essay_fp.append(X_test['preprocessed_essays'].values[i])
```

```
print(len(essay_fp))
```

810

In [131]:

```
#https://www.geeksforgeeks.org/generating-word-cloud-python/
from wordcloud import WordCloud, STOPWORDS
comment_words = ' '
stopwords = set(STOPWORDS)
for val in essay_fp:
    val = str(val)
    tokens = val.split()
for i in range(len(tokens)):
    tokens[i] = tokens[i].lower()
for words in tokens:
    comment_words = comment_words + words + ' '
wordcloud = WordCloud(width=800, height = 800, background_color='white', stopwords = stopwords, min
_font_size = 10).generate(comment_words)

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

plt.show()
```



box plot with the price of these false positive data points

In [133]:

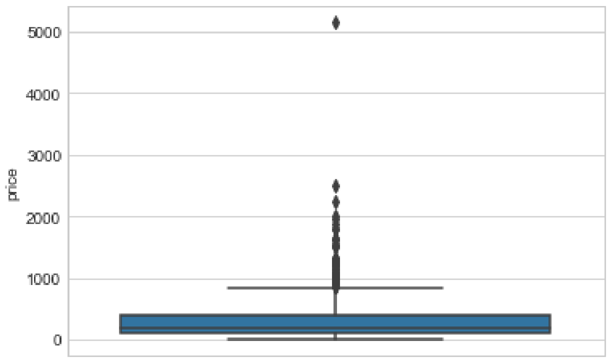
```
X_test_fp = pd.DataFrame(X_test['price'])
print(X_test_fp.shape)
X_test_fp = X_test_fp.iloc[fpi,:]
print(X_test_fp.shape)
```

```
import seaborn as sns
sns.set_style('whitegrid')
sns.boxplot(x='price', data = X_test_fp, orient='v')
```

```
(15000, 1)
(810, 1)
```

Out[133]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x27fe6df47b8>
```



Most of the incorrectly accepted projects were of the price-range 0-900

pdf with the teacher_number_of_previously_posted_projects of these false positive data points

In [134]:

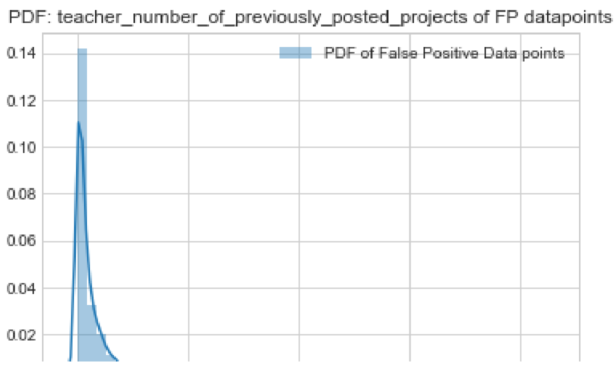
```
X_test_fp_tnpp = pd.DataFrame(X_test['teacher_number_of_previously_posted_projects'])
print(X_test_fp_tnpp.shape)
X_test_fp_tnpp = X_test_fp_tnpp.iloc[fpi,:]
print(X_test_fp_tnpp.shape)

sns.set_style('whitegrid')
sns.distplot(X_test_fp_tnpp.values, label='PDF of False Positive Data points')
plt.title('PDF: teacher_number_of_previously_posted_projects of FP datapoints')
plt.xlabel('teacher_number_of_previously_posted_projects')
plt.legend()
plt.show
```

```
(15000, 1)
(810, 1)
```

```
C:\Users\syeda\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning:

The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
```

Out[134]:

```
<function matplotlib.pyplot.show(*args, **kw)>
```



From the above pdf, it looks like most of the incorrectly accepted projects(FP data) had teacher_number_of_previously_posted_projects value less than 10

Task 2:

In [136]:

```
from scipy.sparse import hstack
X_tr3 = hstack((X_train_titles_tfidf, X_train_essay_tfidf,
X_train_teacher_number_of_previously_posted_projects_norm, X_train_price_norm,
X_train_project_categories_ohe, X_train_project_subcategories_ohe, X_train_grade_ohe,
X_train_teacher_ohe, X_train_state_ohe)).tocsr()
X_te3 = hstack((X_test_titles_tfidf, X_test_essay_tfidf,
X_test_teacher_number_of_previously_posted_projects_norm, X_test_price_norm,
X_test_project_categories_ohe, X_test_project_subcategories_ohe, X_test_grade_ohe,
X_test_teacher_ohe, X_test_state_ohe)).tocsr()

print("Final Data Matrix:")
print(X_tr3.shape, y_train.shape)
print(X_te3.shape, y_test.shape)
```

```
Final Data Matrix:
(35000, 4476) (35000,)
(15000, 4476) (15000,)
```

In [141]:

```
clf = DecisionTreeClassifier()
clf.fit(X_tr3, y_train)
```

Out[141]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [178]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectFromModel.html
sfm = SelectFromModel(clf, threshold=1e-5)

# Train the selector
sfm.fit(X_tr3, y_train)
```

Out[178]:

```
SelectFromModel(estimator=DecisionTreeClassifier(ccp_alpha=0.0,
                                                 class_weight=None,
                                                 criterion='gini',
                                                 max_depth=None,
                                                 max_features=None,
                                                 max_leaf_nodes=None,
                                                 min_impurity_decrease=0.0,
                                                 min_impurity_split=None,
                                                 min_samples_leaf=1,
                                                 min_samples_split=2,
                                                 min_weight_fraction_leaf=0.0,
                                                 presort='deprecated',
                                                 random_state=None,
                                                 splitter='best'),
                max_features=None, norm_order=1, prefit=False, threshold=1e-05)
```

In [182]:

In [182]:

```
x= sfm.estimator_.feature_importances_
```

In [183]:

```
x.shape
```

Out[183]:

```
(4476,)
```

In [185]:

```
sfm.get_support()
```

Out[185]:

```
array([False, False, False, ..., False, False, False])
```

In [186]:

```
X_tr3_fi = sfm.transform(X_tr3)
X_te3_fi = sfm.transform(X_te3)
```

In [187]:

```
X_tr3_fi.shape
```

Out[187]:

```
(35000, 1745)
```

In [188]:

```
X_te3_fi.shape
```

Out[188]:

```
(15000, 1745)
```

Applying LinearSVM

Hyperparameter tuning using GridSearchCV

In [190]:

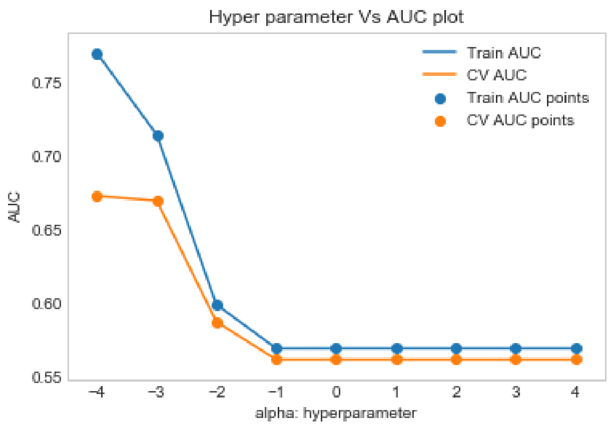
```
import math
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model

svm = linear_model.SGDClassifier(loss='hinge', penalty='l2', class_weight = 'balanced')
parameters = {'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]}
clf = GridSearchCV(svm, parameters, cv=5, scoring='roc_auc', return_train_score = True)
clf.fit(X_tr3_fi, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
alpha =  clf.cv_results_['param_alpha']
print(alpha)
```

```
[0.0001 0.001 0.01 0.1 1 10 100 1000 10000]
```

In [191]:

```
#use log10(alpha) on X axis while plotting ROC vs hyperparam plot, as it allows us to understand w
hat's happening in a better way
log_alpha = []
for a in tqdm(alpha):
    b=math.log10(a)
    log_alpha.append(b)

plt.plot(log_alpha, train_auc, label='Train AUC')
plt.plot(log_alpha, cv_auc, label='CV AUC')

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

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

```
100%|████████████████████████████████████████| 9/9
[00:00<00:00, 9470.33it/s]
```

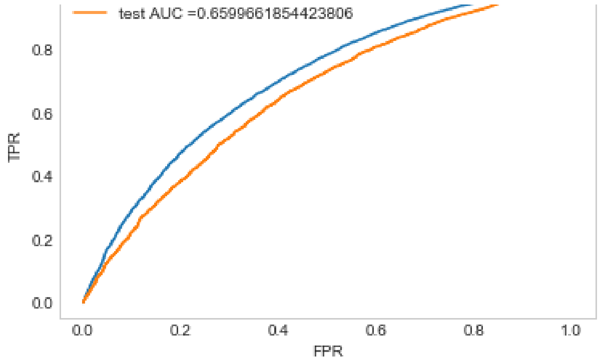


Best value of alpha: 0.001

Testing the performance of the model on test data, plotting ROC Curves

In [193]:

```
from sklearn.metrics import roc_curve, auc

svm = linear_model.SGDClassifier(loss='hinge', penalty='l2', alpha=0.001, class_weight = 'balanced'
)
svm.fit(X_tr3_fi, y_train)

y_train_pred = svm.decision_function(X_tr3_fi)
y_test_pred = svm.decision_function(X_te3_fi)

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

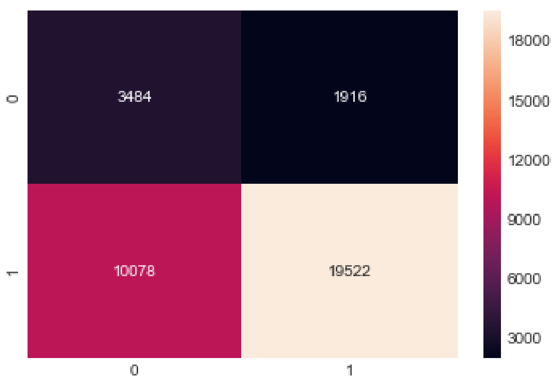


Summary: From the above plot, we observe that at alpha=0.001 and using L2 regularizer we get the train-AUC of 0.7059 and test-AUC of 0.6599.

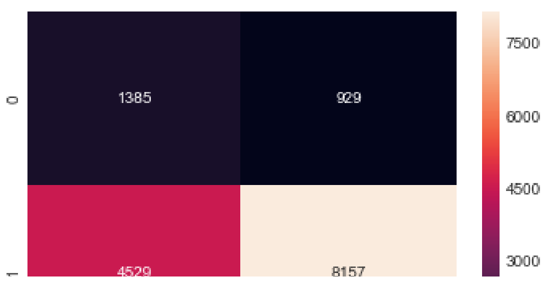
In [194]:

```
from sklearn.metrics import confusion_matrix
fig = plt.figure()
ax = fig.add_subplot(111)
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print('-'*75)
print("Train confusion matrix")
cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax)


fig = plt.figure()
ax1 = fig.add_subplot(111)
print('-'*75)
print("Test confusion matrix")
cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax1)
```

```
the maximum value of tpr*(1-fpr) 0.4255170670670671 for threshold 0.007
---------------------------------------------------------------------------
Train confusion matrix
```




```
---------------------------------------------------------------------------
Test confusion matrix
```



In [195]:

```
#https://towardsdatascience.com/demystifying-confusion-matrix-confusion-9e82201592fd
tn, fp, fn, tp = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)).ravel()
print("True Negatives: ",tn)
print("False Positives: ",fp)
print("False Negatives: ",fn)
print("True Positives: ",tp)
```

```
True Negatives:  1385
False Positives:  929
False Negatives:  4529
True Positives:  8157
```

Summary

In [196]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Model", "Hyper Parameter", "Test-AUC"]

x.add_row(["Decision Tree with TFIDF", (10,500), 0.6354])
x.add_row(["Decision Tree with TFIDF-W2V", (5,500), 0.6217])
x.add_row(["LinearSVM on Non-zero FI", 0.001, 0.6599])

print(x)
```

```
+------------------------------+-----------------+----------+
|            Model             | Hyper Parameter | Test-AUC |
+------------------------------+-----------------+----------+
|   Decision Tree with TFIDF   |    (10, 500)    |  0.6354  |
| Decision Tree with TFIDF-W2V |     (5, 500)    |  0.6217  |
|   LinearSVM on Non-zero FI   |      0.001      |  0.6599  |
+------------------------------+-----------------+----------+
```