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)
```

In [5]:

```
X_train.head(2)
```

Out[5]:

	Unnamed: 0	Unnamed: 0.1	id	teacher_id	te
43676	43676	152262	p203367	640c72932c7413d6be631a3d4ea627f1	
29172	29172	10691	p232099	b28a2c98d6bc923c6ad67045c4c70d35	

In [6]:

```
y_train
```

Out[6]:

```
array([1, 1, 0, ..., 1, 1, 1], dtype=int64)
```

In [7]:

```
X_test.head(2)
```

Out[7]:

	Unnamed: 0	Unnamed: 0.1	id	teacher_id	tea
33818	33818	105887	p186644	71409aabf3bacfdfff94ca510c5c8476	
6140	6140	101178	p020768	34b27c2f81c204041ca52eb54caf8b68	

In [8]:

```
y_test
```

Out[8]:

```
array([1, 1, 1, ..., 1, 0, 1], dtype=int64)
```

1.4 Encoding Categorical and Numerical features

1.4.1 encoding categorical features: clean_categories

In [9]:

```
vectorizer_cat = CountVectorizer()
vectorizer_cat.fit(X_train['clean_categories'].values) # fit has to happen or

X_train_cc_ohe = vectorizer_cat.transform(X_train['clean_categories'].values)
X_test_cc_ohe = vectorizer_cat.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_cc_ohe.shape, y_train.shape)
print(X_test_cc_ohe.shape, y_test.shape)
print(vectorizer_cat.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 [10]:

```
vectorizer_subcat = CountVectorizer()
vectorizer_subcat.fit(X_train['clean_subcategories'].values) # fit has to hap

X_train_csc_ohe = vectorizer_subcat.transform(X_train['clean_subcategories'].
X_test_csc_ohe = vectorizer_subcat.transform(X_test['clean_subcategories'].va

print("After vectorizations")
print(X_train_csc_ohe.shape, y_train.shape)
print(X_test_csc_ohe.shape, y_test.shape)
print(vectorizer_subcat.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 [11]:

```
vectorizer_school_state = CountVectorizer()
vectorizer_school_state.fit(X_train['school_state'].values)

X_train_state_ohe = vectorizer_school_state.transform(X_train['school_state'].
X_test_state_ohe = vectorizer_school_state.transform(X_test['school_state'].v

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer_school_state.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 [12]:

```
vectorizer_prefix = CountVectorizer()
vectorizer_prefix.fit(X_train['teacher_prefix'].values)

X_train_teacher_ohe = vectorizer_prefix.transform(X_train['teacher_prefix'].v
X_test_teacher_ohe = vectorizer_prefix.transform(X_test['teacher_prefix'].val

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer_prefix.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 [13]:

```
vectorizer_grade = CountVectorizer()
vectorizer_grade.fit(X_train['project_grade_category'].values)

X_train_grade_ohe = vectorizer_grade.transform(X_train['project_grade_categor
X_test_grade_ohe = vectorizer_grade.transform(X_test['project_grade_category'

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer_grade.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 [14]:

```
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,)
[[0.00537419]
 [0.005011  ]
 [0.00511608]
 ...
 [0.00206252]
 [0.00667761]
 [0.00377337]]
[[0.00188648]
 [0.00106282]
 [0.0093107 ]
 ...
 [0.00853019]
 [0.02930259]
 [0.00461425]]
```

1.4.7 encoding numerical features: teacher_number_of_previously_posted_projects

In [15]:

```
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,)
[[0.00124794]
 [0.00083196]
 [0.00041598]
 ...
 [0.00062397]
 [0.00166392]
 [0.        ]]
[[0.01365105]
 [0.00158733]
 [0.00031747]
 ...
 [0.        ]
 [0.00031747]
 [0.00031747]]
```

1.4.8 encoding numerical features: quantity

In [16]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(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,)
[[0.0056184 ]
 [0.00181772]
 [0.00033049]
 ...
 [0.00033049]
 [0.00644463]
 [0.00495741]]
[[0.00253953]
 [0.0013852 ]
 [0.00692598]
 ...
 [0.00092346]
 [0.00046173]
 [0.00023087]]
```

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

In [17]:

```
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
0.142
0.166
(33500,)
(16500,)
```

In [18]:

```
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,)
[[0.00164784]
 [0.00521317]
 [0.00368517]
 ...
 [0.00560266]
 [0.00590227]
 [0.00755011]]
[[0.00870842]
 [0.00700088]
 [0.01054401]
 ...
 [0.00751314]
 [0.00307356]
 [0.00426883]]
```

1.4.10 encoding numerical features: number of words in the title

In [19]:

```
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
3
3
(33500,)
(16500,)
```

In [20]:

```
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,)
[[0.00486319]
 [0.00680847]
 [0.00194528]
 ...
 [0.00291792]
 [0.00486319]
 [0.00389055]]
[[0.01387776]
 [0.00277555]
 [0.01110221]
 ...
 [0.00277555]
 [0.00971443]
 [0.00693888]]
```

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

In [21]:

```
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
199
276
(33500,)
(16500,)
```

In [22]:

```
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,)
[[0.00624041]
 [0.00973913]
 [0.00495141]
 ...
 [0.00497187]
 [0.00744757]
 [0.00368286]]
[[0.01184383]
 [0.00778893]
 [0.0070888 ]
 ...
 [0.00568854]
 [0.0129232 ]
 [0.00860574]]
```

1.5 Vectorizing Text features

1.5.1 Vectorizing using BOW

Essay

In [23]:

```
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_bow_essay = CountVectorizer(min_df=10,ngram_range=(1,4), max_featu
vectorizer_bow_essay.fit(X_train['essay'].values) # fit has to happen only or

X_train_essay_bow = vectorizer_bow_essay.transform(X_train['essay'].values)
X_test_essay_bow = vectorizer_bow_essay.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.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 [24]:

```
vectorizer_bow_title = CountVectorizer(min_df=10,ngram_range=(1,4), max_featu
vectorizer_bow_title.fit(X_train['project_title'].values) # fit has to happer


X_train_titles_bow = vectorizer_bow_title.transform(X_train['project_title'].
X_test_titles_bow = vectorizer_bow_title.transform(X_test['project_title'].va

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

```
After vectorizations
(33500, 4021) (33500,)
(16500, 4021) (16500,)
```

project_resource_summary

In [25]:

```
vectorizer_bow_rs = CountVectorizer(min_df=10,ngram_range=(1,4), max_features
vectorizer_bow_rs.fit(X_train['project_resource_summary'].values) # fit has t


X_train_psr_bow = vectorizer_bow_rs.transform(X_train['project_resource_summa
X_test_psr_bow = vectorizer_bow_rs.transform(X_test['project_resource_summary

print("After vectorizations")
print(X_train_psr_bow.shape, y_train.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 [26]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf_essay = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_fea
vectorizer_tfidf_essay.fit(X_train['essay'].values)

X_train_essay_tfidf = vectorizer_tfidf_essay.transform(X_train['essay'].value
X_test_essay_tfidf = vectorizer_tfidf_essay.transform(X_test['essay'].values)

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

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

project_title

In [27]:

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

X_train_titles_tfidf = vectorizer_tfidf_title.transform(X_train['project_titl
X_test_titles_tfidf = vectorizer_tfidf_title.transform(X_test['project_title'

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

```
After vectorizations
(33500, 4021) (33500,)
(16500, 4021) (16500,)
```

project_resource_summary

In [28]:

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

X_train_prs_tfidf = vectorizer_tfidf_rs.transform(X_train['project_resource_s
X_test_prs_tfidf = vectorizer_tfidf_rs.transform(X_test['project_resource_sum

print("After vectorizations")
print(X_train_prs_tfidf.shape, y_train.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 [29]:

```
# 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 [30]:

```
# 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 [00:11<00:00, 2936.78it/s]

33500
300
```

In [31]:

```
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 [00:05<00:00, 2921.46it/s]
```

project_title

In [32]:

```
# 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, 150303.57it/s]

33500
300
```

In [33]:

```
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, 149750.22it/s]
```

project_resource_summary

In [34]:

```
# 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, 32433.97it/s]

33500
300
```

In [35]:

```
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, 31332.09it/s]
```

1.5.4 Vectorizing using TFIDF W2V

project_title

In [36]:

```
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, 100878.69it/s]

33500
300
```

In [37]:

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

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

```
100%|██████████████████████████████████████████████████████████
██████████████| 16500/16500 [00:00<00:00, 100250.77it/s]

16500
300
```

essay

In [38]:

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

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

```
100%|████████████████████████████████████████████████████████████
████████████████| 33500/33500 [02:19<00:00, 240.81it/s]

33500
300
```

In [39]:

```
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 [01:07<00:00, 243.23it/s]

16500
300
```

project_resource_summary

In [40]:

```
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, 10774.89it/s]

33500
300
```

In [41]:

```
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, 10780.64it/s]

16500
300
```

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

In [42]:

```
# 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_ewc_norm, X_train_twc_norm, X_train_ss_n
                             X_train_essay_bow, X_train_titles_bow, X_train_p


X_test_bow_matrix = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_grade_ohe,
                            X_test_teacher_ohe, X_test_price_norm, X_test_ppp
                            X_test_ewc_norm, X_test_twc_norm, X_test_ss_norm,
                            X_test_essay_bow, X_test_titles_bow, X_test_psr_b

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

```
Final Data matrix
(33500, 14122) (33500,)
(16500, 14122) (16500,)
```

In [43]:

```
# 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_ewc_norm, X_train_twc_norm, X_train_ss_n
                               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_teacher_ohe, X_test_price_norm, X_test_ppp
                              X_test_ewc_norm, X_test_twc_norm, X_test_ss_norm,
                              X_test_titles_tfidf, X_test_essay_tfidf, X_test_p

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, 14122) (33500,)
(16500, 14122) (16500,)
```

In [44]:

```
# 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_train
                              X_train_ewc_norm, X_train_twc_norm, X_train_ss_n
                              avg_w2v_essay_train, avg_w2v_titles_train, avg_w

X_test_aw2v_matrix = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_grade_ohe,
                             X_test_teacher_ohe, X_test_price_norm, X_test_ppp
                             X_test_ewc_norm, X_test_twc_norm, X_test_ss_norm,
                             avg_w2v_essay_test, avg_w2v_titles_test, avg_w2v_

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

```
Final Data matrix
(33500, 1001) (33500,)
(16500, 1001) (16500,)
```

In [45]:

```
# 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_train
                              X_train_ewc_norm, X_train_twc_norm, X_train_ss_n
                              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_teacher_ohe, X_test_price_norm, X_test_ppp
                             X_test_ewc_norm, X_test_twc_norm, X_test_ss_norm,
                             tfidf_w2v_essay_test, tfidf_w2v_title_test, tfidf

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, 1001) (33500,)
(16500, 1001) (16500,)
```

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

In [46]:

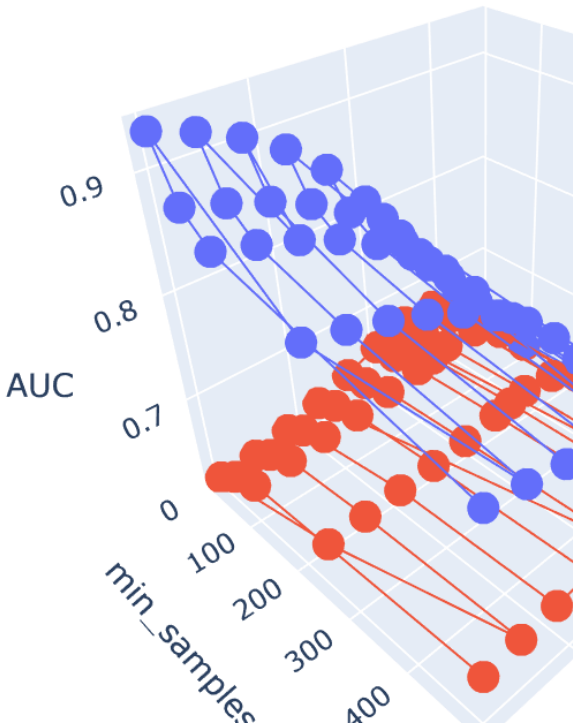
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.G
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier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

parameters = {'max_depth' : [2, 4, 6, 8, 9, 10, 12, 14, 16, 18, 20], 'min_sam
dtc = DecisionTreeClassifier(class_weight =  "balanced")
clf = GridSearchCV(dtc, parameters, cv=5, scoring='roc_auc', return_train_sco
clf.fit(X_train_bow_matrix, y_train)

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

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_max_depth']
M = results['param_min_samples_split']


results
```

Out[46]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	para
0	1.712000	0.047273	0.040116	0.008381	2	
1	1.692217	0.064606	0.039741	0.008219	2	
2	1.741809	0.033066	0.055598	0.005515	2	
3	1.804599	0.085000	0.039574	0.009951	2	

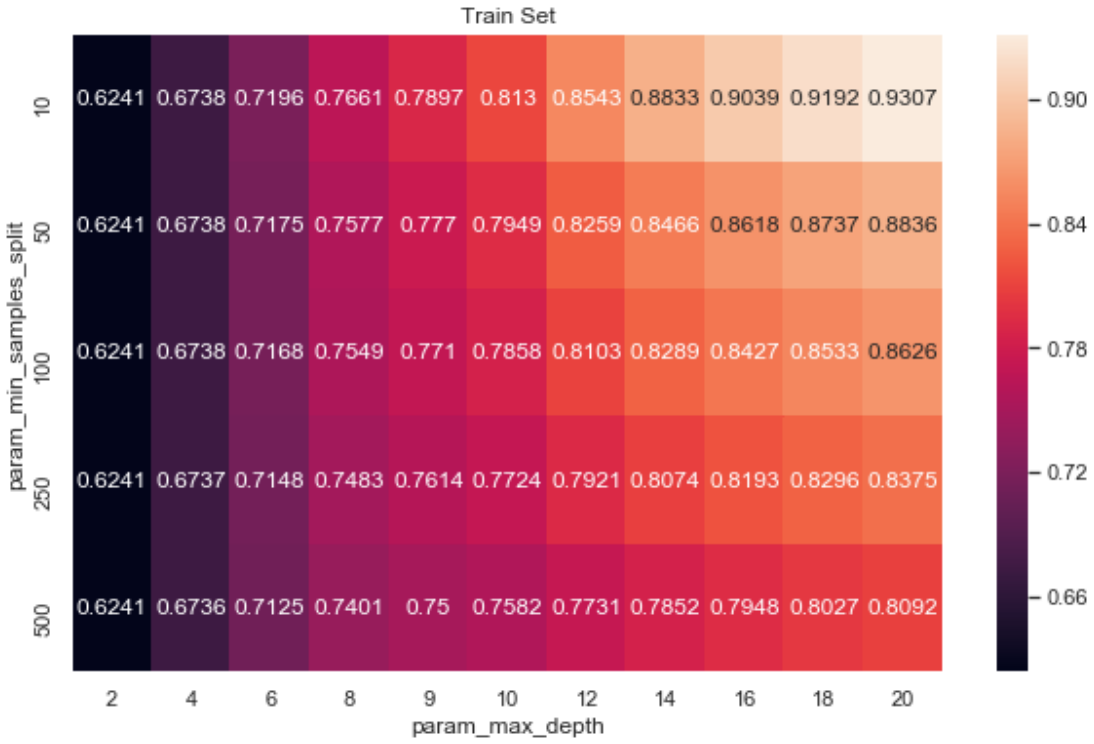
In [47]:

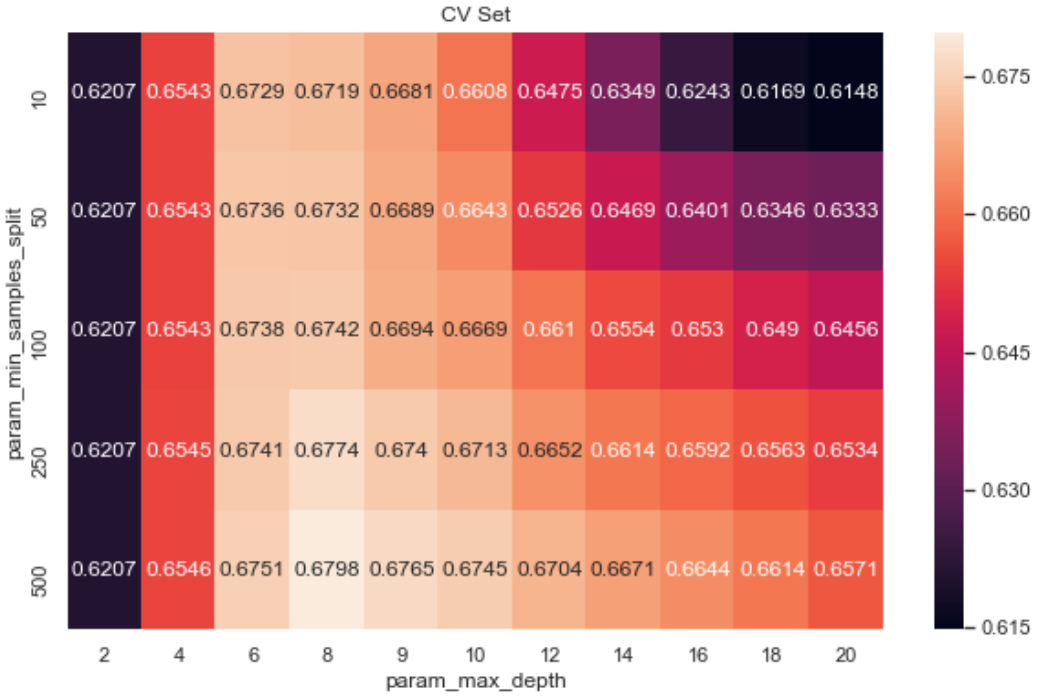
```
trace1 = go.Scatter3d(x = K, y = M, z = train_auc, name = 'Train')
trace2 = go.Scatter3d(x = K, y = M, z = cv_auc, name = 'Cross Validation')
data = [trace1, trace2]

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

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



In [48]:

```
import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(clf.cv_results_).groupby(['param_min_samples_split
plt.figure(figsize=(10,6))
plt.title('Train Set')
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g')
plt.show()


plt.figure(figsize=(10,6))
plt.title('CV Set')
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g')
plt.show()
```





In [49]:

```
best_max_depth_bow = clf.best_params_['max_depth']
best_min_samples_split_bow = clf.best_params_['min_samples_split']
print('best value for max depth is {} and best value for min samples split is
```

```
best value for max depth is 8 and best value for min samples s
plit is 500
```

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

In [50]:

```
parameters = {'max_depth' : [2, 4, 6, 8, 9, 10, 12, 14, 16, 18, 20], 'min_sam
dtc = DecisionTreeClassifier(class_weight =  "balanced")
clf = GridSearchCV(dtc, parameters, cv=5, scoring='roc_auc', return_train_sco
clf.fit(X_train_bow_matrix, y_train)

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

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_max_depth']
M = results['param_min_samples_split']
```

In [51]:

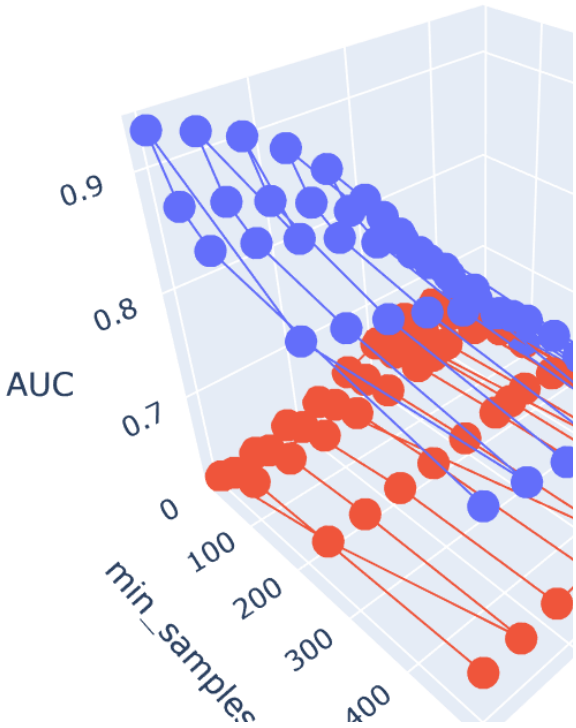
```
results
```

Out[51]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	para
0	1.629803	0.023339	0.032912	0.003337	2	
1	1.770487	0.166434	0.050278	0.015584	2	
2	1.769947	0.159036	0.050966	0.021474	2	
3	1.829452	0.177473	0.046276	0.025393	2	

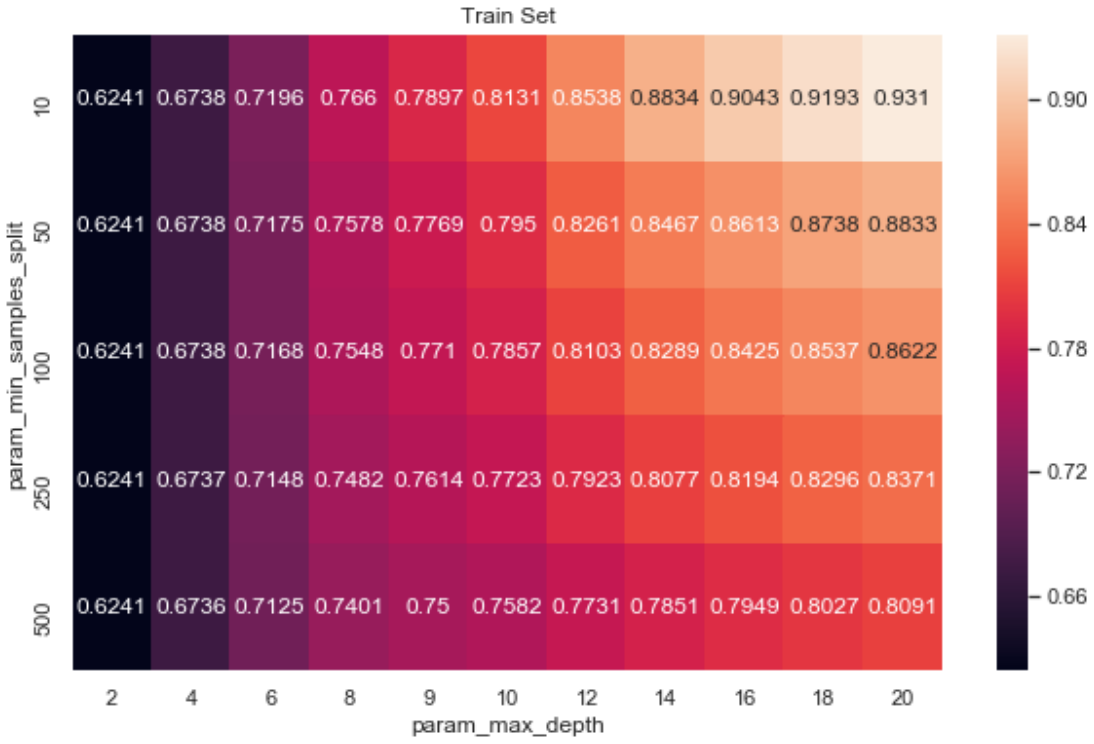
In [52]:

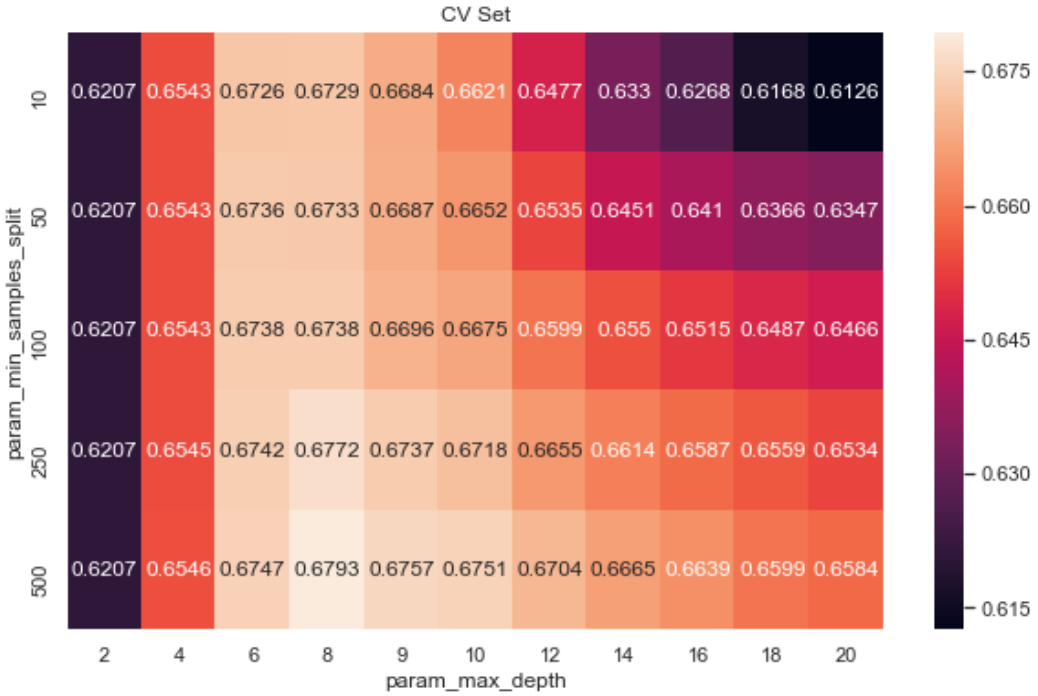
```
trace1 = go.Scatter3d(x = K, y = M, z = train_auc, name = 'Train')
trace2 = go.Scatter3d(x = K, y = M, z = cv_auc, name = 'Cross Validation')
data = [trace1, trace2]

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

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



In [53]:

```
import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(clf.cv_results_).groupby(['param_min_samples_split
plt.figure(figsize=(10,6))
plt.title('Train Set')
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g')
plt.show()


plt.figure(figsize=(10,6))
plt.title('CV Set')
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g')
plt.show()
```





In [54]:

```
best_max_depth_tfidf = clf.best_params_['max_depth']
best_min_samples_split_tfidf = clf.best_params_['min_samples_split']
print('best value for max depth is {} and best value for min samples split is
```

```
best value for max depth is 8 and best value for min samples s
plit is 500
```

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

In [55]:

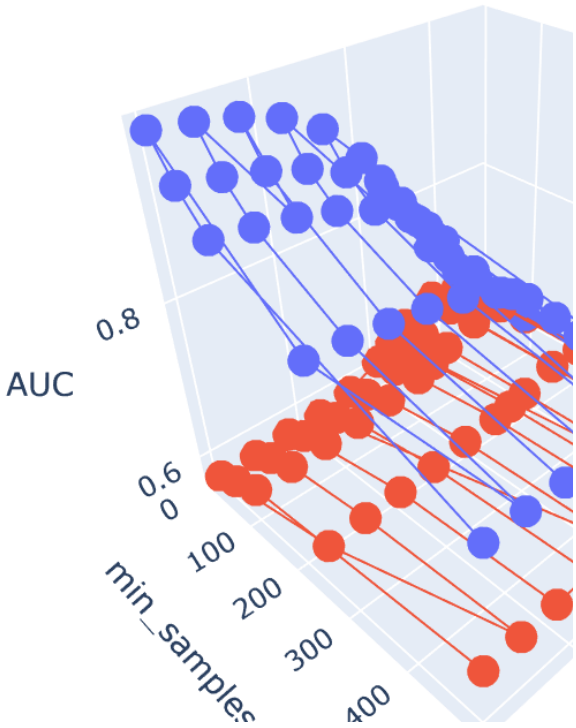
```
parameters = {'max_depth' : [2, 4, 6, 8, 9, 10, 12, 14, 16, 18, 20], 'min_sar
dtc = DecisionTreeClassifier(class_weight =  "balanced")
clf = GridSearchCV(dtc, parameters, cv=5, scoring='roc_auc', return_train_sc
clf.fit(X_train_aw2v_matrix, y_train)

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

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_max_depth']
M = results['param_min_samples_split']
results
```

Out[55]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	para
0	9.073709	0.395961	0.101467	0.049019	2	
1	8.991551	0.074362	0.081228	0.008158	2	
2	9.169541	0.292204	0.117928	0.024518	2	
3	9.075585	0.355840	0.068065	0.016601	2	

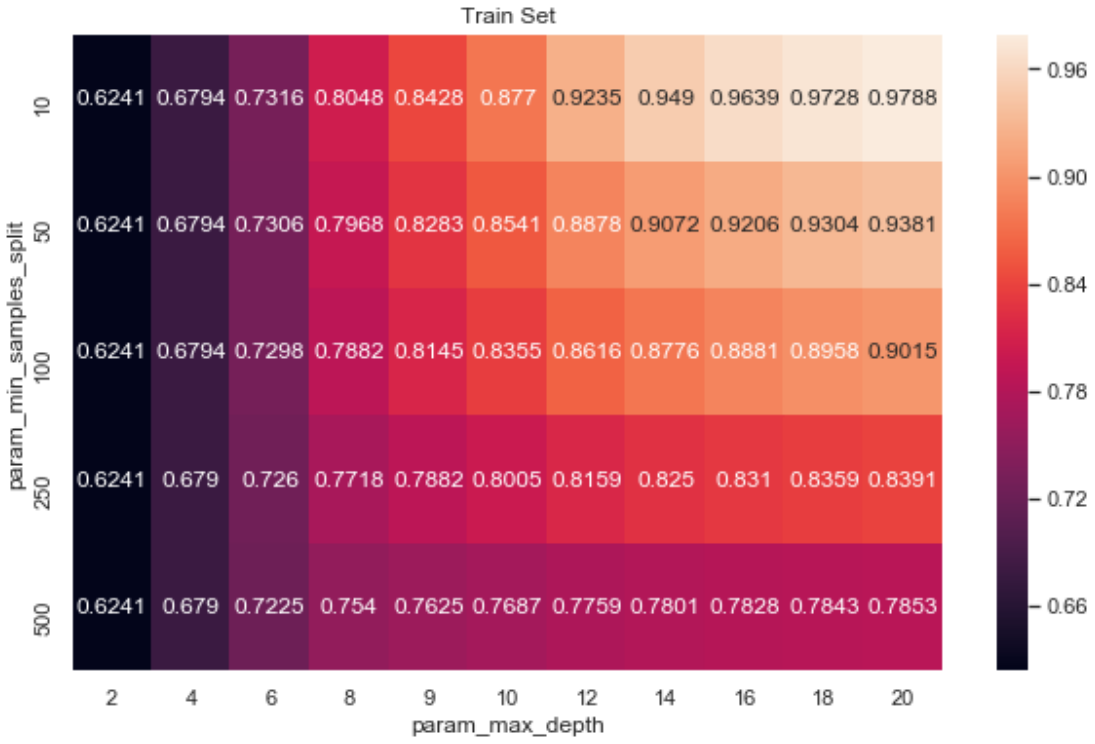
In [56]:

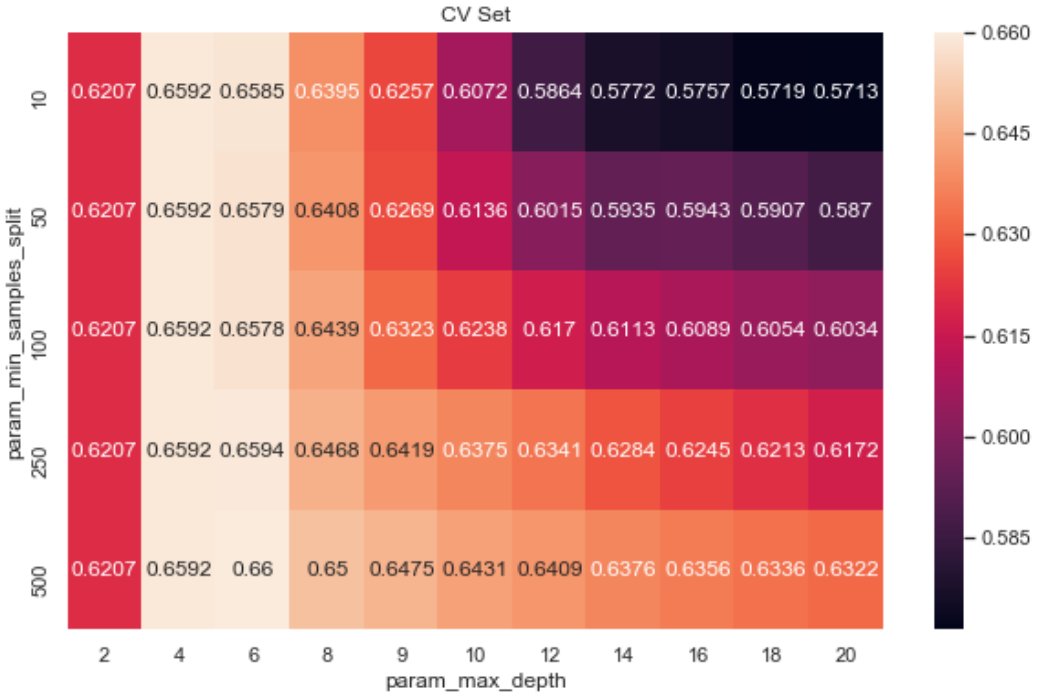
```
trace1 = go.Scatter3d(x = K, y = M, z = train_auc, name = 'Train')
trace2 = go.Scatter3d(x = K, y = M, z = cv_auc, name = 'Cross Validation')
data = [trace1, trace2]

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

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



In [57]:

```
import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(clf.cv_results_).groupby(['param_min_samples_split
plt.figure(figsize=(10,6))
plt.title('Train Set')
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g')
plt.show()


plt.figure(figsize=(10,6))
plt.title('CV Set')
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g')
plt.show()
```





In [58]:

```
best_max_depth_aw2v = clf.best_params_['max_depth']
best_min_samples_split_aw2v = clf.best_params_['min_samples_split']
print('best value for max depth is {} and best value for min samples split is
```

```
best value for max depth is 6 and best value for min samples s
plit is 500
```

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

In [59]:

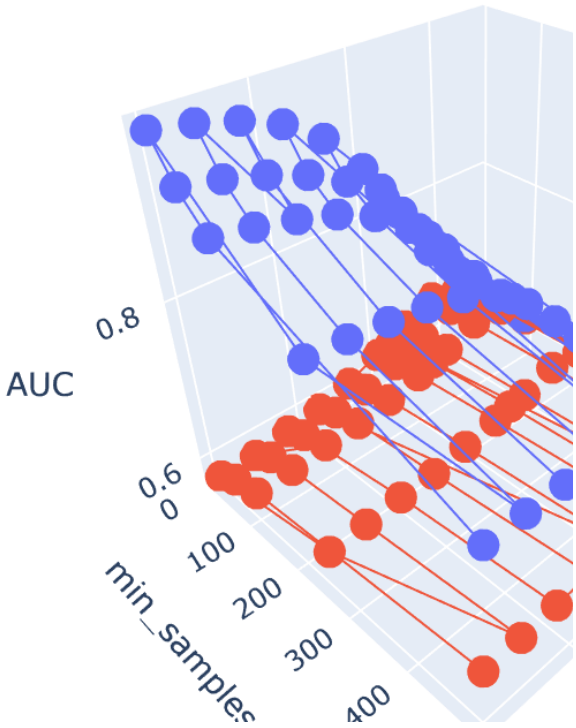
```
parameters = {'max_depth' : [2, 4, 6, 8, 9, 10, 12, 14, 16, 18, 20], 'min_sar
dtc = DecisionTreeClassifier(class_weight =  "balanced")
clf = GridSearchCV(dtc, parameters, cv=5, scoring='roc_auc', return_train_sc
clf.fit(X_train_tw2v_matrix, y_train)

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

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_max_depth']
M = results['param_min_samples_split']
results
```

Out[59]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	para
0	8.511832	0.244704	0.069805	0.020916	2	
1	8.912551	0.307221	0.120678	0.044723	2	
2	8.749189	0.324302	0.108908	0.040983	2	
3	8.896416	0.374482	0.066223	0.016101	2	

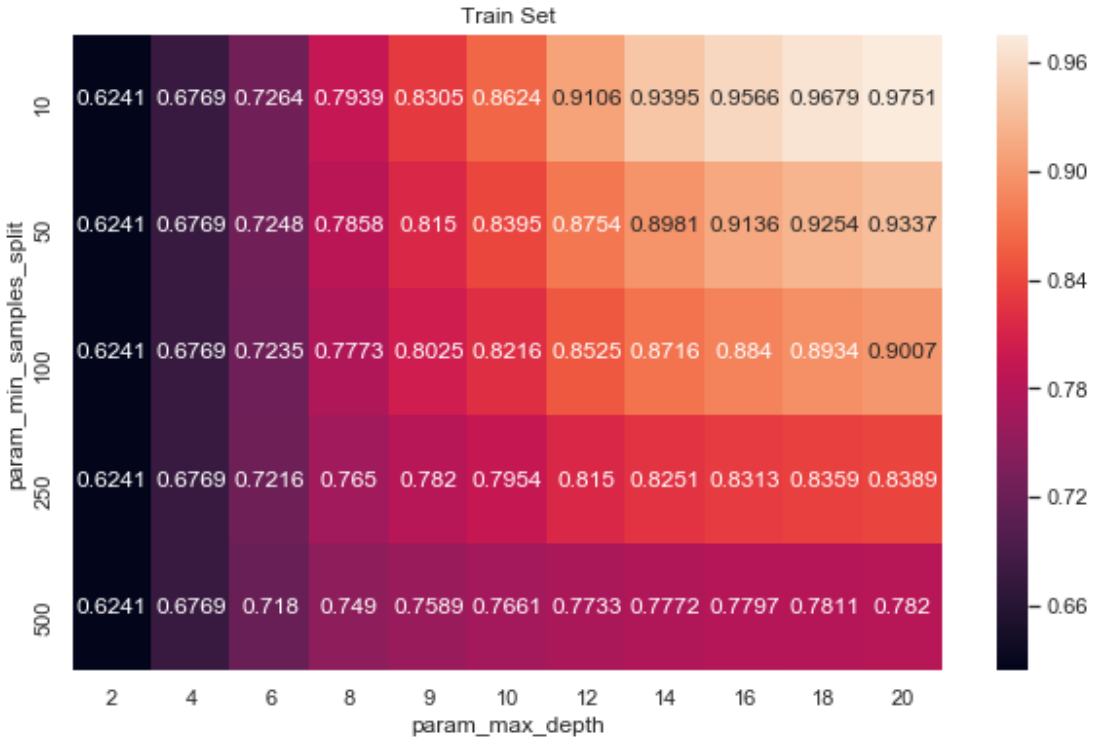
In [60]:

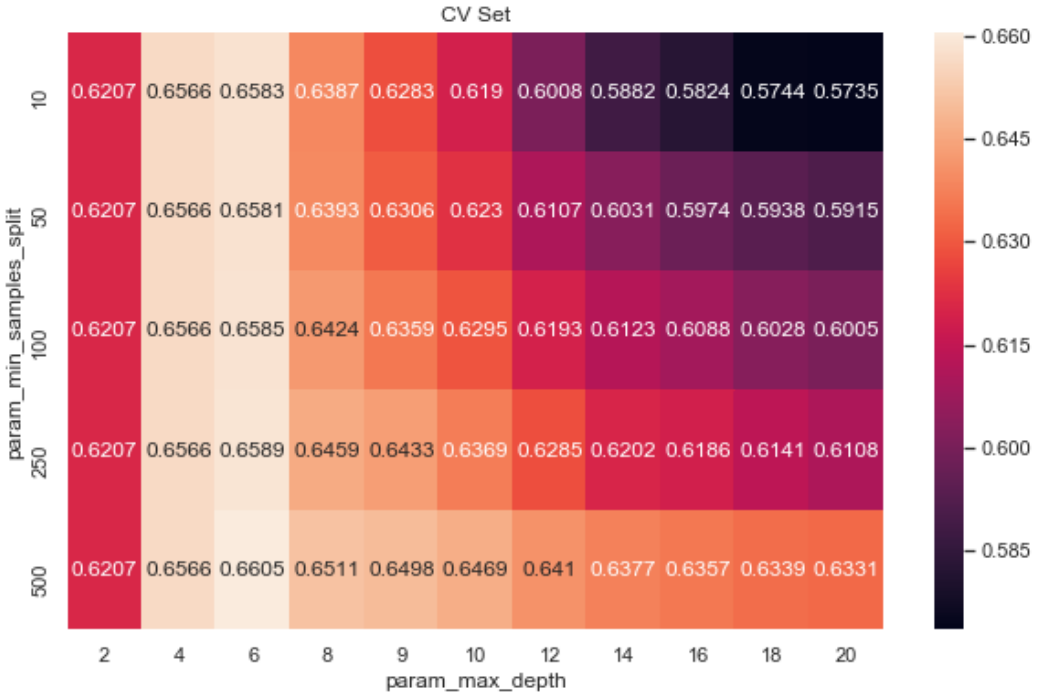
```
trace1 = go.Scatter3d(x = K, y = M, z = train_auc, name = 'Train')
trace2 = go.Scatter3d(x = K, y = M, z = cv_auc, name = 'Cross Validation')
data = [trace1, trace2]

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

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



In [61]:

```
import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(clf.cv_results_).groupby(['param_min_samples_split
plt.figure(figsize=(10,6))
plt.title('Train Set')
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g')
plt.show()


plt.figure(figsize=(10,6))
plt.title('CV Set')
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g')
plt.show()
```





In [62]:

```
best_max_depth_tw2v = clf.best_params_['max_depth']
best_min_samples_split_tw2v = clf.best_params_['min_samples_split']
print('best value for max depth is {} and best value for min samples split is
```

```
best value for max depth is 6 and best value for min samples s
plit is 500
```

In [63]:

```
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
```

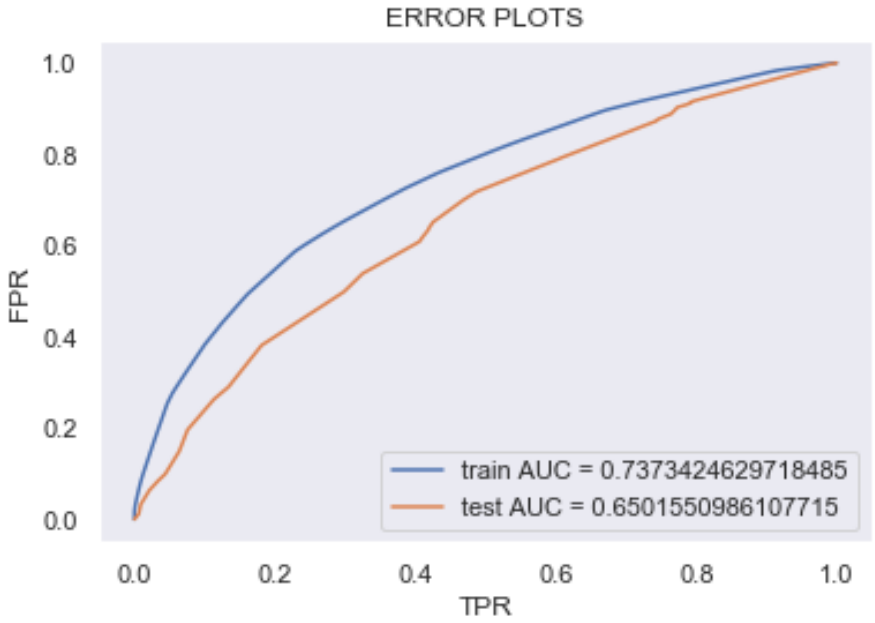
In [64]:

```
# 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 Decision Tree with obtained best Hyper parameter on BOW

In [65]:

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

dt = DecisionTreeClassifier(max_depth= best_max_depth_bow, min_samples_split=
dt.fit(X_train_bow_matrix, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability esti
# not the predicted outputs

y_train_pred = batch_predict(dt, X_train_bow_matrix)
y_test_pred = batch_predict(dt, 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()
```

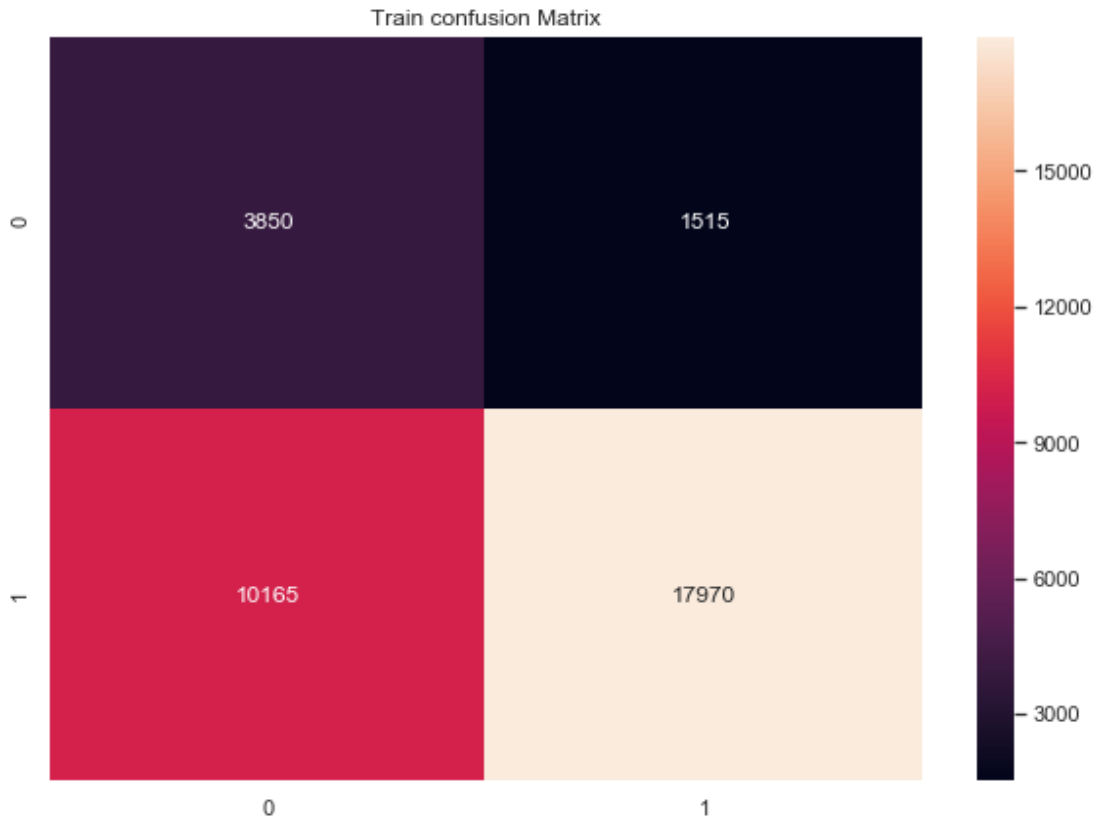


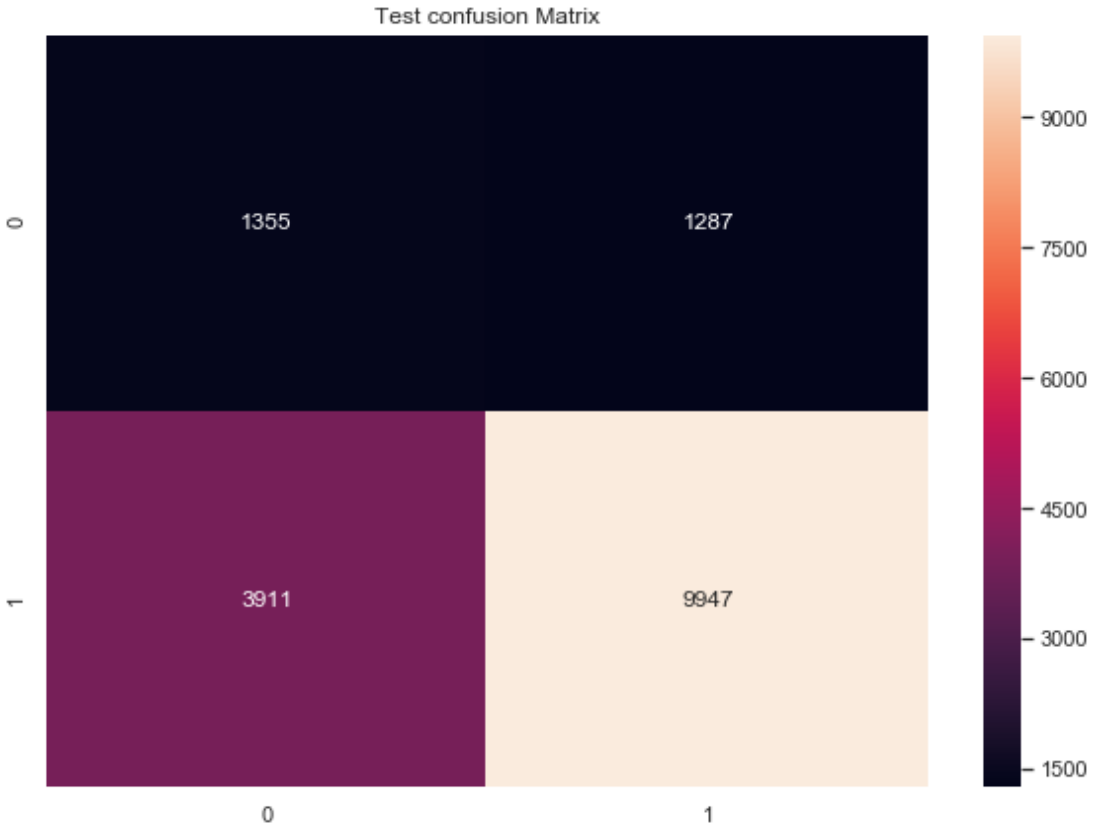
Confusion Matrix with predicted and original labels for BOW

In [66]:

```
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/35572247dt

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) 0.4583446440747752 for thresh
old 0.476
```




Test confusion Matrix
1355
1287
3911
9947
0
1
0
1
9000
7500
6000
4500
3000
1500

In [67]:

```
bow_features_names = []

for feature in vectorizer_cat.get_feature_names() :
    bow_features_names.append(feature)

for feature in vectorizer_subcat.get_feature_names() :
    bow_features_names.append(feature)

for feature in vectorizer_school_state.get_feature_names() :
    bow_features_names.append(feature)

for feature in vectorizer_grade.get_feature_names() :
    bow_features_names.append(feature)

for feature in vectorizer_prefix.get_feature_names() :
    bow_features_names.append(feature)

for feature in vectorizer_bow_title.get_feature_names() :
    bow_features_names.append(feature)

for feature in vectorizer_bow_essay.get_feature_names() :
    bow_features_names.append(feature)

for feature in vectorizer_bow_rs.get_feature_names() :
    bow_features_names.append(feature)

bow_features_names.append("price")
bow_features_names.append("quantity")
bow_features_names.append("teacher_number_of_previously_posted_projects")
bow_features_names.append('title_word_count')
bow_features_names.append('essay_word_count')
bow_features_names.append('essay_sentiment_score')

print('Number of feature are :',len(bow_features_names))
bow_features_names
```

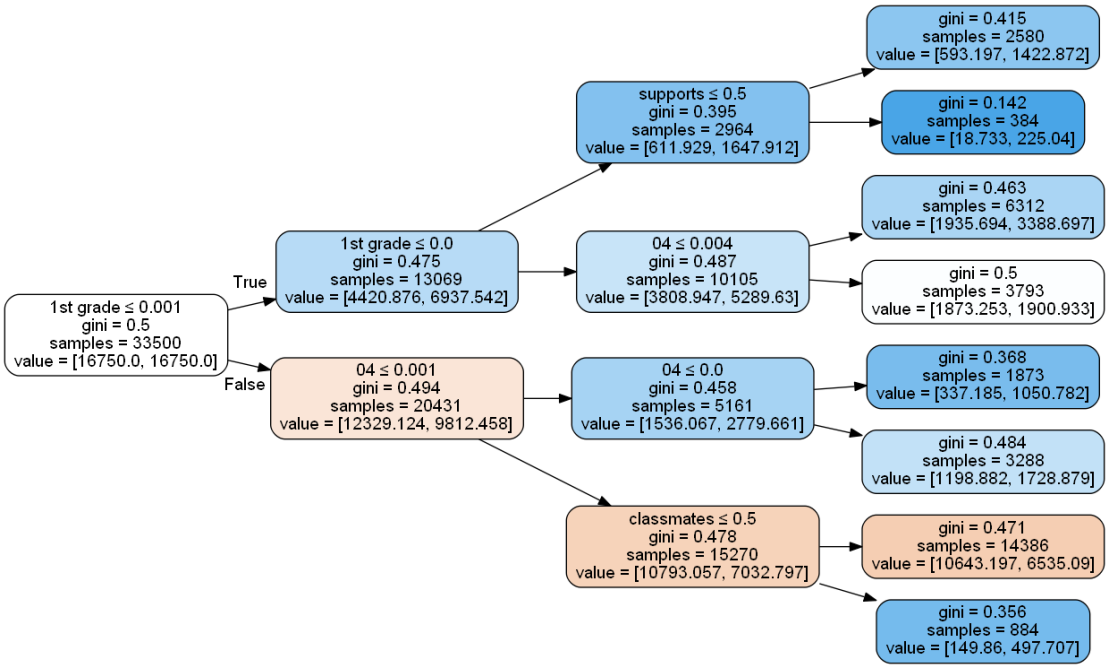
```
 'criss',
 'criss cross',
 'critical',
 'critical thinkers',
 'critical thinking',
 'critical thinking through',
 'cross',
 'cultivating',
 'culturally',
 'culture',
 'curiosity',
 'curious',
 'current',
 'current events',
 'curriculum',
```

```
 'cushions',
 'cut',
 'daily',
 'dance',
 'dancing',
```

In [68]:

```
import warnings
import graphviz
warnings.filterwarnings("ignore")
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
from graphviz import Source
import pydotplus

dt = DecisionTreeClassifier(max_depth= 3, min_samples_split=best_min_samples_
dt.fit(X_train_bow_matrix, y_train)

dot_data = StringIO()
export_graphviz(dt, out_file=dot_data, filled=True, rounded=True, special_cha
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[68]:



In [69]:

```
predictions = predict_with_best_t(y_test_pred,best_t)
len(predictions)
```

Out[69]:

16500

In [70]:

```
act_vs_predicted = pd.DataFrame({'index' : X_test.index, 'actual_label':y_tes
act_vs_predicted.head(20)
```

Out[70]:

	index	actual_label	predicted_label
0	33818	1	1
1	6140	1	1
2	11555	1	0
3	23477	1	1
4	31007	1	0
5	49771	1	1
6	20399	1	1
7	35350	1	0
8	44517	1	1
9	33698	1	1
10	6332	1	0
11	15025	1	0
12	44119	1	1
13	496	0	1
14	19356	0	1
15	11569	1	0
16	23147	0	1
17	4775	1	1
18	25411	1	1
19	2472	1	1

In [71]:

```
false_positives = []
for i in tqdm(range(len(act_vs_predicted))):
    if(act_vs_predicted['actual_label'][i]==0 and act_vs_predicted['predicted
        false_positives.append(i)

len(false_positives)
```

```
100%|██████████████████████████████████████████████████
████████████████| 16500/16500 [00:00<00:00, 72130.55it/s]
```

Out[71]:

```
1287
```

In [72]:

```
false_positives_essay = []
for i in false_positives :
    false_positives_essay.append(X_test['essay'].values[i])
```

In [73]:

```
false_positives_essay[0]
```

Out[73]:

```
'My students are eager, not only for musical knowledge, but fo
r playing musical instrument as well.Presently, I teach elemen
tary/middle school music (4-8) at a high needs public school i
n Illinois. Due to budget cuts, the school is unable to fund a
music program for the school. Music, being an interdisciplinar
y subject, encompasses all subjects students encounter during
their education. The students at this school are very enthusia
stic about music and should have a top-notch music program. Th
is can only be possible with your help!The students will learn
how to work together in an ensemble which encourages team wor
k. They will also receive the satisfaction of learning how to
play a musical instrument. Music is very entertaining and allo
ws for freedom of expression. My students deserve the opportun
ity to express themselves through music, and playing the trump
et will provide that foundation.Music is the universal languag
e, and provides an experience that will allow the students to
appreciate the creative process, and develop a commitment to e
xcellence. Your donations will help the students to experience
music on a greater level through their performance.'
```

In [74]:

```
from wordcloud import WordCloud, STOPWORDS
from os import path
from PIL import Image
comment_words = ' '
stopwords = set(STOPWORDS)
for val in false_positives_essay :
    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 + ' '
```

In [75]:

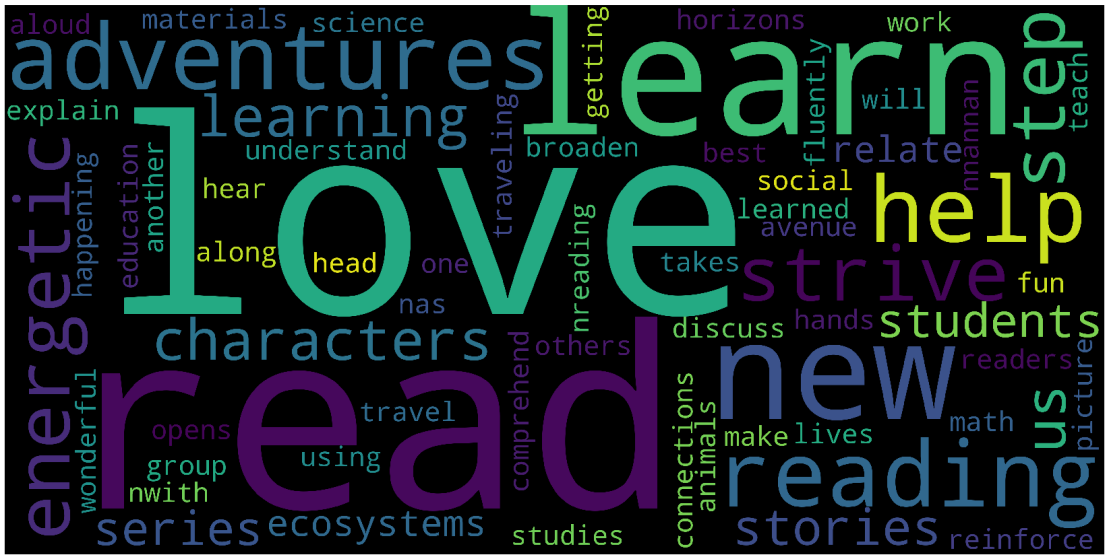
```
len(comment_words)
```

Out[75]:

1095

In [76]:

```
wc = WordCloud(width=5000,height=2500,background_color="black",stopwords=stop
wc.generate(comment_words)
plt.figure( figsize=(30,15))
plt.imshow(wc)
plt.axis("off")
plt.show()
```



In [77]:

```
test_price_df = pd.DataFrame(X_test['price'])
fp_price_df = test_price_df.iloc[false_positives,:]
fp_price_df.head(5)
```

Out[77]:

	price
496	199.98
19356	40.59
23147	37.04
19171	537.97
31151	549.00

In [78]:

```
len(fp_price_df)
```

Out[78]:

1287

In [79]:

```
sns.boxplot(y='price', data=fp_price_df)
plt.title('False Positives "Price" Box Plot', size =15, y = 1.07)
plt.show()
```

In [80]:

```
test_nppp_df = pd.DataFrame(X_test['teacher_number_of_previously_posted_proje
fp_nppp_df = test_nppp_df.iloc[false_positives,:]
fp_nppp_df.head(5)
```

Out[80]:

	teacher_number_of_previously_posted_projects
496	25
19356	16
23147	0
19171	6
31151	1

In [81]:

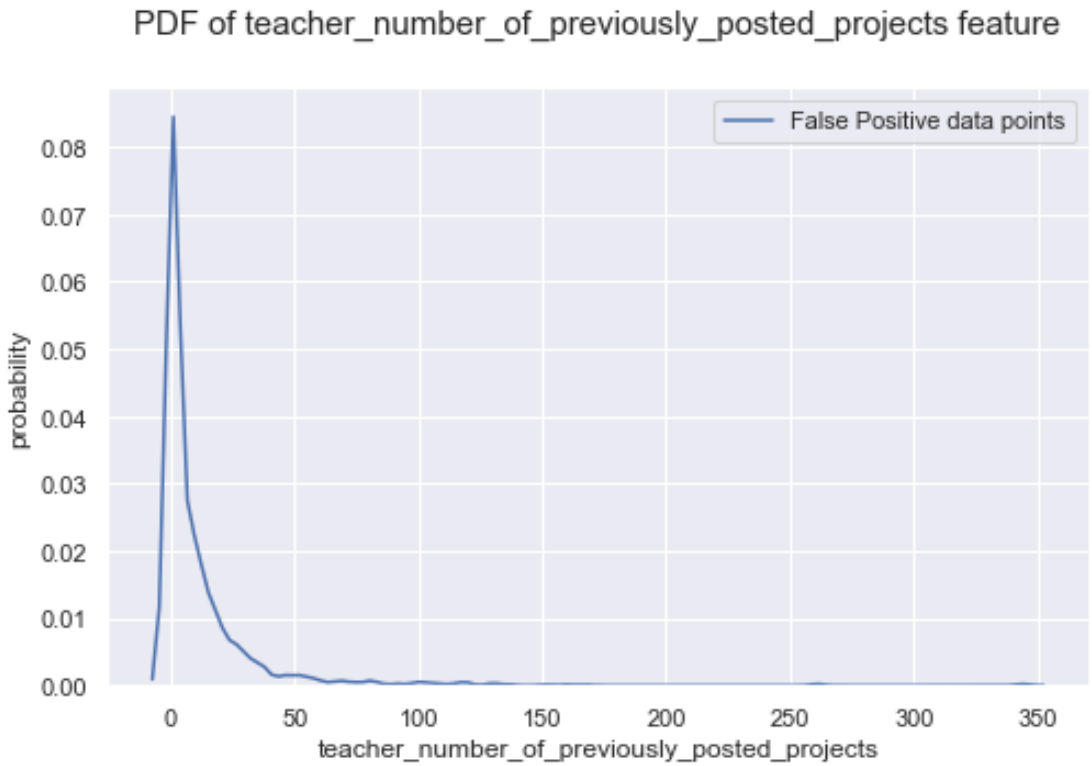
```
len(fp_nppp_df)
```

Out[81]:

1287

In [82]:

```
plt.figure(figsize=(8,5))
sns.distplot(fp_nppp_df.values, hist=False, label="False Positive data points
plt.title('PDF with the Teacher_number_of_previously_posted_projects for the
plt.xlabel('Teacher_number_of_previously_posted_projects')
plt.ylabel('probability')
plt.legend()
plt.xlabel('teacher_number_of_previously_posted_projects')
plt.title('PDF of teacher_number_of_previously_posted_projects feature', size
plt.show()
```



In [ ]:

```
```

Applying Decision Tree with obtained best Hyper parameter on TFIDF

In [83]:

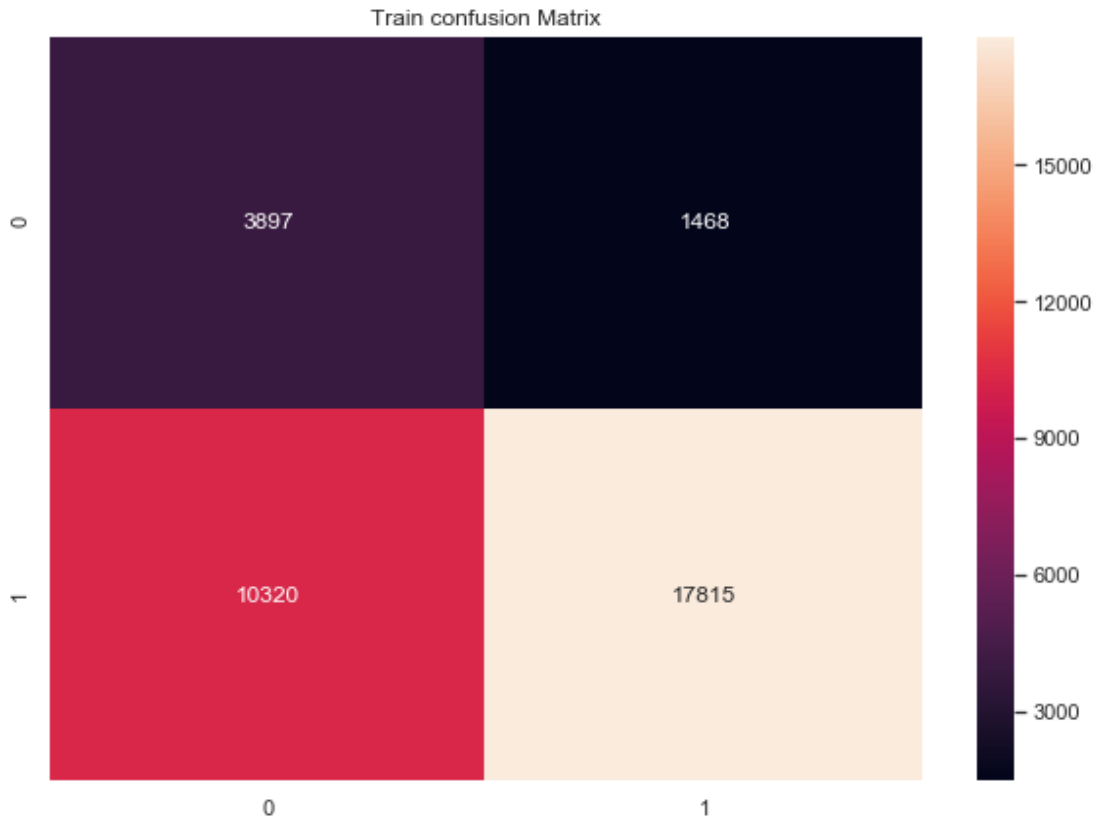
```
dt = DecisionTreeClassifier(max_depth= best_max_depth_tfidf, min_samples_spli
dt.fit(X_train_tfidf_matrix, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability esti
# not the predicted outputs

y_train_pred = batch_predict(dt, X_train_tfidf_matrix)
y_test_pred = batch_predict(dt, 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()
```

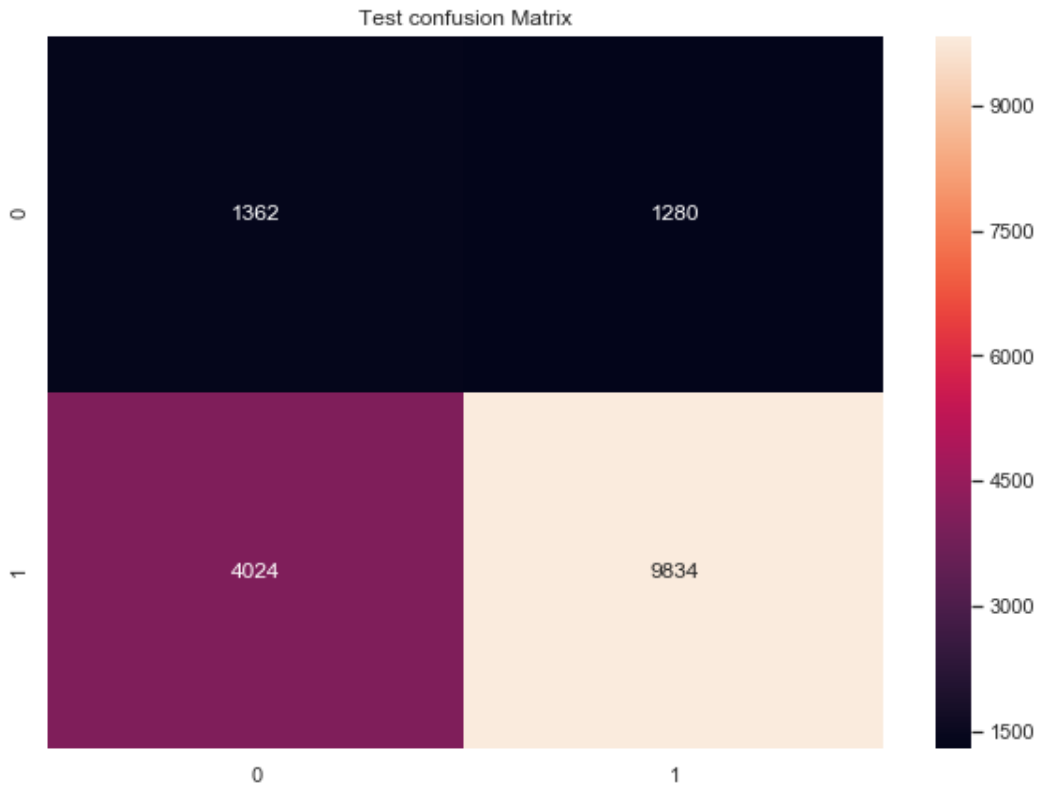
Confusion Matrix with predicted and original labels for TFIDF

In [84]:

```
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) 0.45993831167164173 for thres
hold 0.463
```




Test confusion Matrix
1362
1280
4024
9834
0
1
0
1
9000
7500
6000
4500
3000
1500

In [85]:

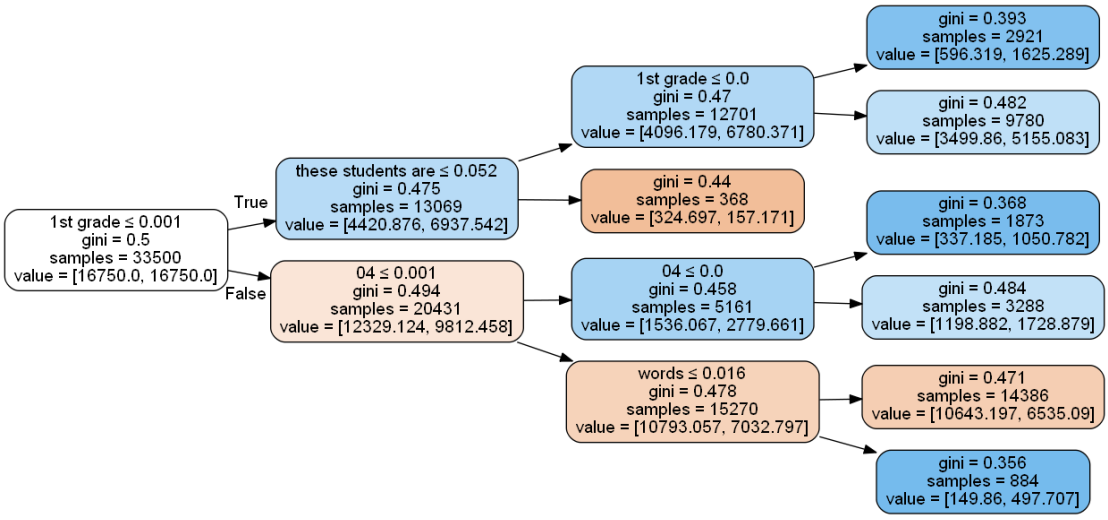
```
tfidf_features_names = []

for feature in vectorizer_cat.get_feature_names() :
    tfidf_features_names.append(feature)

for feature in vectorizer_subcat.get_feature_names() :
    tfidf_features_names.append(feature)

for feature in vectorizer_school_state.get_feature_names() :
    tfidf_features_names.append(feature)

for feature in vectorizer_grade.get_feature_names() :
    tfidf_features_names.append(feature)

for feature in vectorizer_prefix.get_feature_names() :
    tfidf_features_names.append(feature)

for feature in vectorizer_tfidf_title.get_feature_names() :
    tfidf_features_names.append(feature)

for feature in vectorizer_tfidf_essay.get_feature_names() :
    tfidf_features_names.append(feature)

for feature in vectorizer_tfidf_rs.get_feature_names() :
    tfidf_features_names.append(feature)

tfidf_features_names.append("price")
tfidf_features_names.append("quantity")
tfidf_features_names.append("teacher_number_of_previously_posted_projects")
tfidf_features_names.append('title_word_count')
tfidf_features_names.append('essay_word_count')
tfidf_features_names.append('essay_sentiment_score')

print('Number of feature are :',len(tfidf_features_names))
```

```
Number of feature are : 14122
```

In [88]:

```
import warnings
import graphviz
warnings.filterwarnings("ignore")
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
from graphviz import Source
import pydotplus

dt = DecisionTreeClassifier(max_depth= 3, min_samples_split=best_min_samples_
dt.fit(X_train_tfidf_matrix, y_train)

dot_data = StringIO()
export_graphviz(dt, out_file=dot_data, filled=True, rounded=True, special_cha
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[88]:



In [ ]:

```
```

In [89]:

```
predictions = predict_with_best_t(y_test_pred,best_t)
len(predictions)
```

Out[89]:

```
16500
```

In [90]:

```
act_vs_predicted = pd.DataFrame({'index' : X_test.index, 'actual_label':y_tes
act_vs_predicted.head(20)
```

Out[90]:

	index	actual_label	predicted_label
0	33818	1	1
1	6140	1	1
2	11555	1	0
3	23477	1	1
4	31007	1	0
5	49771	1	1
6	20399	1	1
7	35350	1	0
8	44517	1	1
9	33698	1	1
10	6332	1	1
11	15025	1	0
12	44119	1	1
13	496	0	1
14	19356	0	1
15	11569	1	0
16	23147	0	1
17	4775	1	1
18	25411	1	1
19	2472	1	1

In [91]:

```
false_positives = []
for i in tqdm(range(len(act_vs_predicted))):
    if(act_vs_predicted['actual_label'][i]==0 and act_vs_predicted['predicted
        false_positives.append(i)

len(false_positives)
```

```
100%|████████████████████████████████████████████████████████████
████████████████| 16500/16500 [00:00<00:00, 87535.29it/s]
```

Out[91]:

```
1280
```

In [92]:

```
false_positives_essay = []
for i in false_positives :
    false_positives_essay.append(X_test['essay'].values[i])
```

In [93]:

```
false_positives_essay[0]
```

Out[93]:

```
'My students are eager, not only for musical knowledge, but fo
r playing musical instrument as well.Presently, I teach elemen
tary/middle school music (4-8) at a high needs public school i
n Illinois. Due to budget cuts, the school is unable to fund a
music program for the school. Music, being an interdisciplinar
y subject, encompasses all subjects students encounter during
their education. The students at this school are very enthusia
stic about music and should have a top-notch music program. Th
is can only be possible with your help!The students will learn
how to work together in an ensemble which encourages team wor
k. They will also receive the satisfaction of learning how to
play a musical instrument. Music is very entertaining and allo
ws for freedom of expression. My students deserve the opportun
ity to express themselves through music, and playing the trump
et will provide that foundation.Music is the universal languag
e, and provides an experience that will allow the students to
appreciate the creative process, and develop a commitment to e
xcellence. Your donations will help the students to experience
music on a greater level through their performance.'
```

In [94]:

```
comment_words = ' '
stopwords = set(STOPWORDS)
for val in false_positives_essay :
    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 + ' '
```

In [95]:

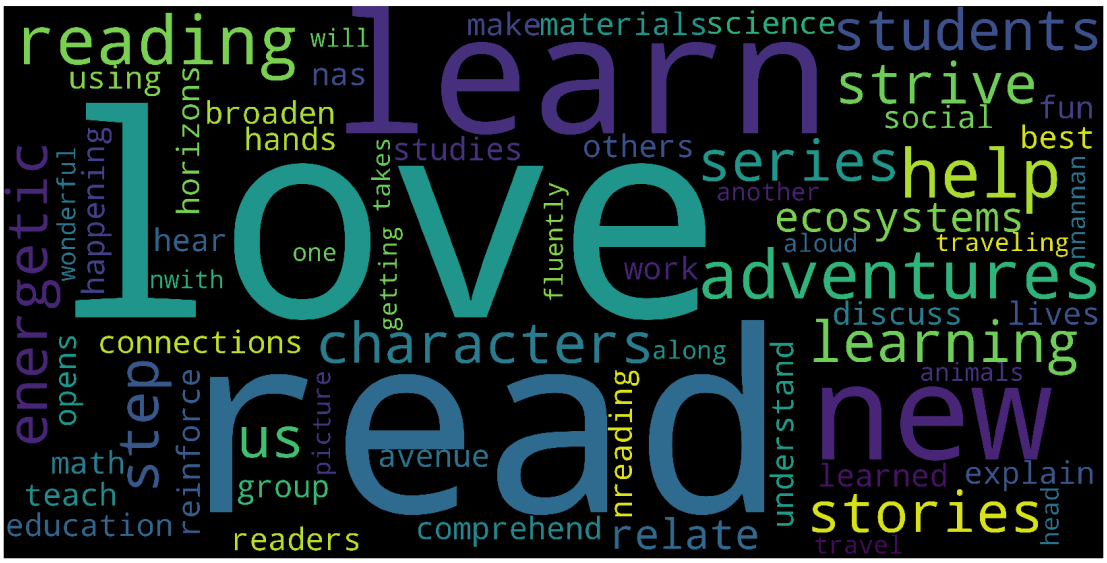
```
len(comment_words)
```

Out[95]:

1095

In [96]:

```
wc = WordCloud(width=5000,height=2500,background_color="black",stopwords=stop
wc.generate(comment_words)
plt.figure( figsize=(30,15))
plt.imshow(wc)
plt.axis("off")
plt.show()
```



In [97]:

```
test_price_df = pd.DataFrame(X_test['price'])
fp_price_df = test_price_df.iloc[false_positives,:]
fp_price_df.head(5)
```

Out[97]:

	price
496	199.98
19356	40.59
23147	37.04
19171	537.97
31151	549.00

In [98]:

```
len(fp_price_df)
```

Out[98]:

1280

In [99]:

```
sns.boxplot(y='price', data=fp_price_df)
plt.title('False Positives "Price" Box Plot', size =15, y = 1.07)
plt.show()
```

In [100]:

```
test_nppp_df = pd.DataFrame(X_test['teacher_number_of_previously_posted_proje
fp_nppp_df = test_nppp_df.iloc[false_positives,:]
fp_nppp_df.head(5)
```

Out[100]:

	teacher_number_of_previously_posted_projects
496	25
19356	16
23147	0
19171	6
31151	1

In [101]:

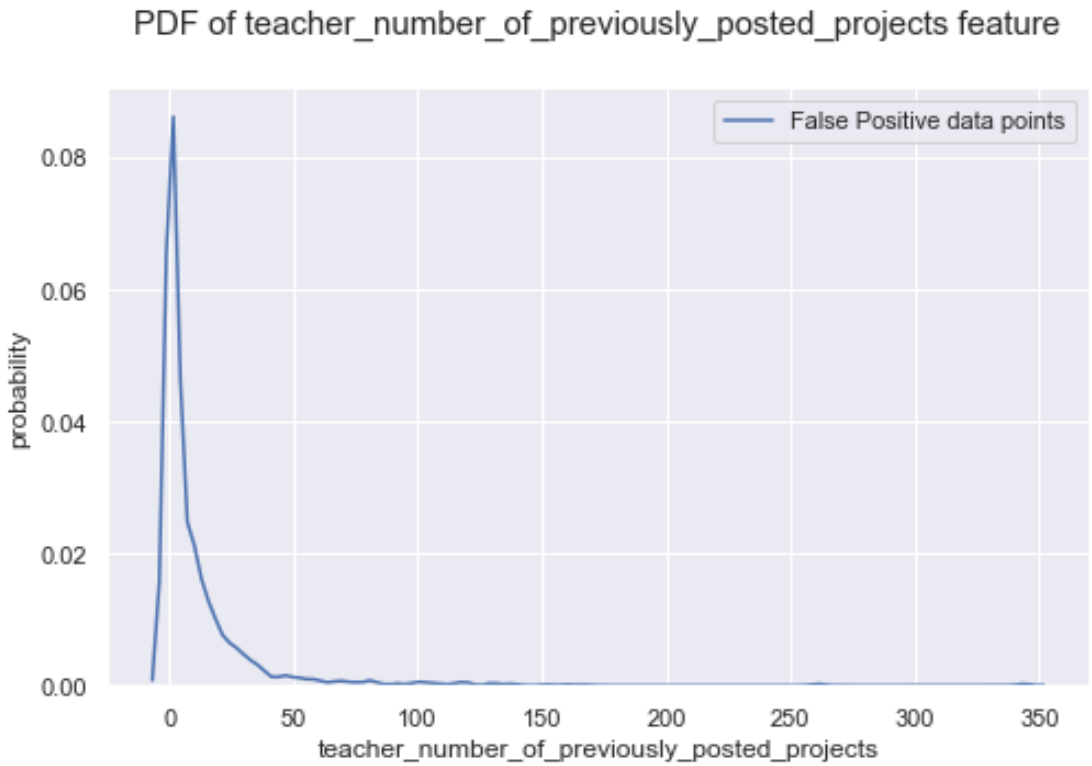
```
len(fp_nppp_df)
```

Out[101]:

```
1280
```

In [102]:

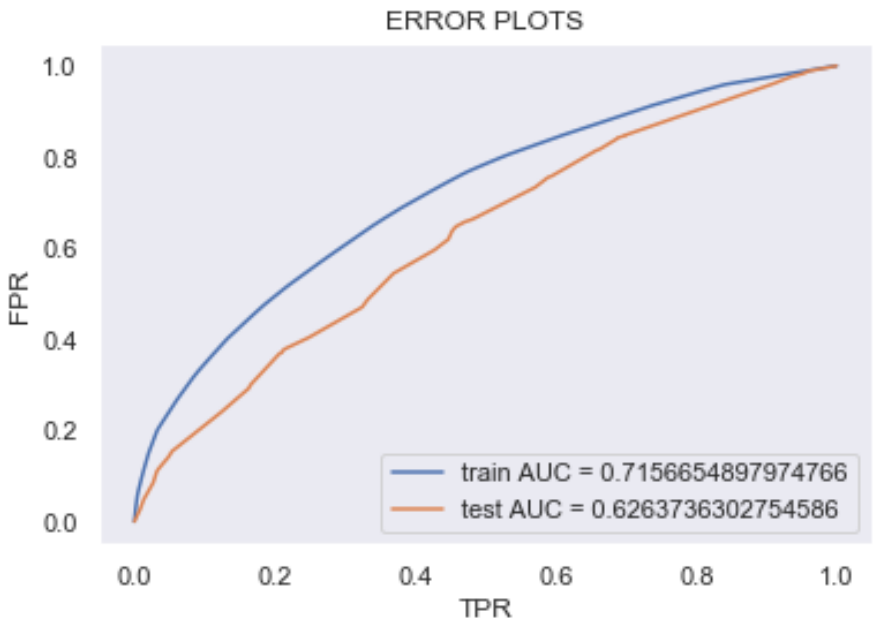
```
plt.figure(figsize=(8,5))
sns.distplot(fp_nppp_df.values, hist=False, label="False Positive data points
plt.title('PDF with the Teacher_number_of_previously_posted_projects for the
plt.xlabel('Teacher_number_of_previously_posted_projects')
plt.ylabel('probability')
plt.legend()
plt.xlabel('teacher_number_of_previously_posted_projects')
plt.title('PDF of teacher_number_of_previously_posted_projects feature', size
plt.show()
```



Applying Decision Tree with obtained best Hyper parameter on AVG W2V representation

In [103]:

```
dt = DecisionTreeClassifier(max_depth= best_max_depth_aw2v, min_samples_split
dt.fit(X_train_aw2v_matrix, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability esti
# not the predicted outputs

y_train_pred = batch_predict(dt, X_train_aw2v_matrix)
y_test_pred = batch_predict(dt, 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()
```

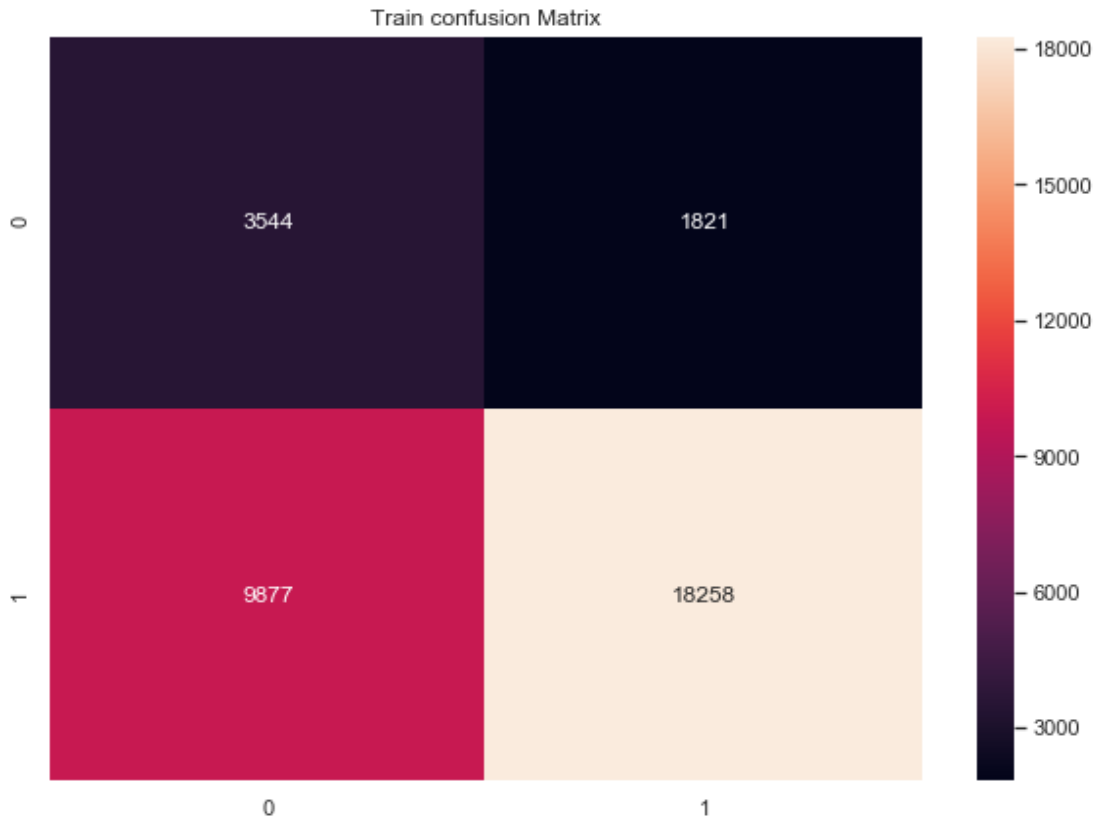


Confusion Matrix with predicted and original labels for AVG W2V

In [104]:

```
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) 0.42867708629558815 for thres
hold 0.516
```



In [105]:

```
predictions = predict_with_best_t(y_test_pred,best_t)
len(predictions)
```

Out[105]:

16500

In [106]:

```
act_vs_predicted = pd.DataFrame({'index' : X_test.index, 'actual_label':y_tes
act_vs_predicted.head(20)
```

Out[106]:

	index	actual_label	predicted_label
0	33818	1	1
1	6140	1	1
2	11555	1	0
3	23477	1	1
4	31007	1	1
5	49771	1	0
6	20399	1	0
7	35350	1	1
8	44517	1	1
9	33698	1	1
10	6332	1	1
11	15025	1	0
12	44119	1	0
13	496	0	1
14	19356	0	1
15	11569	1	0
16	23147	0	1
17	4775	1	1
18	25411	1	0
19	2472	1	1

In [107]:

```
false_positives = []
for i in tqdm(range(len(act_vs_predicted))):
    if(act_vs_predicted['actual_label'][i]==0 and act_vs_predicted['predicted
        false_positives.append(i)

len(false_positives)
```

```
100%|████████████████████████████████████████████████████████████
████████████████| 16500/16500 [00:00<00:00, 82721.07it/s]
```

Out[107]:

```
1265
```

In [108]:

```
false_positives_essay = []
for i in false_positives :
    false_positives_essay.append(X_test['essay'].values[i])
```

In [109]:

```
false_positives_essay[0]
```

Out[109]:

```
'My students are eager, not only for musical knowledge, but fo
r playing musical instrument as well.Presently, I teach elemen
tary/middle school music (4-8) at a high needs public school i
n Illinois. Due to budget cuts, the school is unable to fund a
music program for the school. Music, being an interdisciplinar
y subject, encompasses all subjects students encounter during
their education. The students at this school are very enthusia
stic about music and should have a top-notch music program. Th
is can only be possible with your help!The students will learn
how to work together in an ensemble which encourages team wor
k. They will also receive the satisfaction of learning how to
play a musical instrument. Music is very entertaining and allo
ws for freedom of expression. My students deserve the opportun
ity to express themselves through music, and playing the trump
et will provide that foundation.Music is the universal languag
e, and provides an experience that will allow the students to
appreciate the creative process, and develop a commitment to e
xcellence. Your donations will help the students to experience
music on a greater level through their performance.'
```

In [110]:

```
comment_words = ' '
stopwords = set(STOPWORDS)
for val in false_positives_essay :
    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 + ' '
```

In [111]:

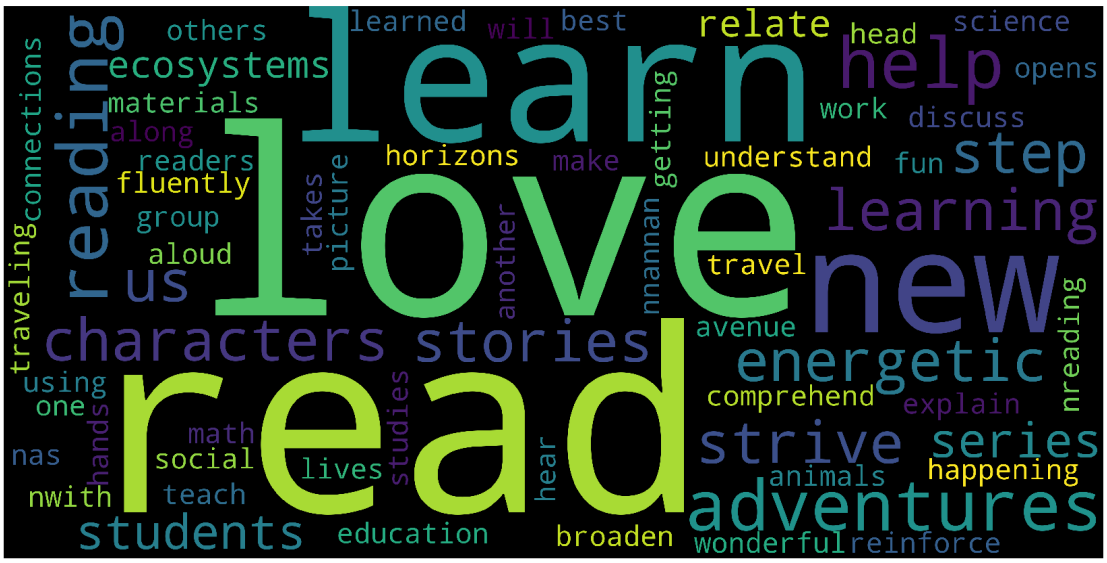
```
len(comment_words)
```

Out[111]:

1095

In [112]:

```
wc = WordCloud(width=5000,height=2500,background_color="black",stopwords=stop
wc.generate(comment_words)
plt.figure( figsize=(30,15))
plt.imshow(wc)
plt.axis("off")
plt.show()
```



In [113]:

```
test_price_df = pd.DataFrame(X_test['price'])
fp_price_df = test_price_df.iloc[false_positives,:]
fp_price_df.head(5)
```

Out[113]:

	price
496	199.98
19356	40.59
23147	37.04
26897	93.90
19171	537.97

In [114]:

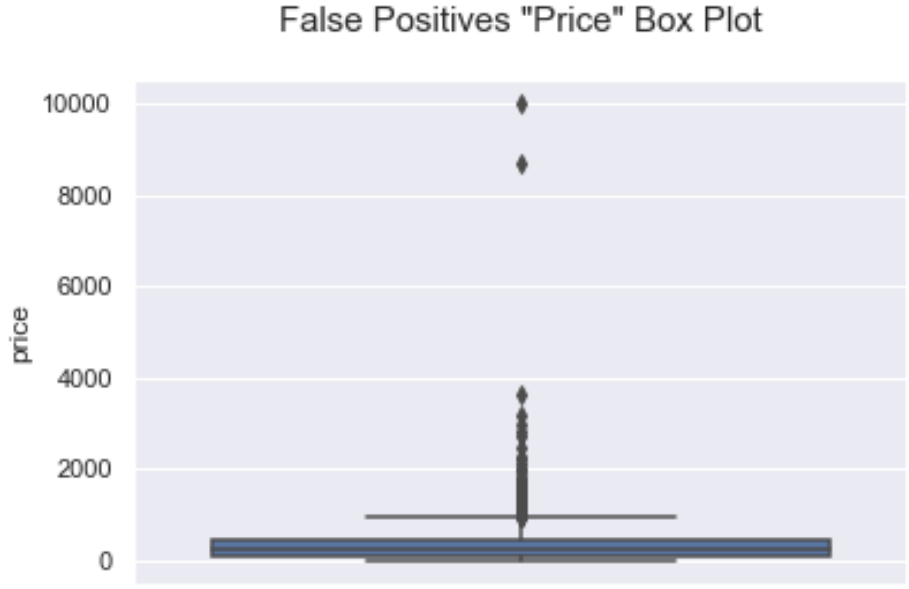
```
len(fp_price_df)
```

Out[114]:

1265

In [115]:

```
sns.boxplot(y='price', data=fp_price_df)
plt.title('False Positives "Price" Box Plot', size =15, y = 1.07)
plt.show()
```



In [116]:

```
test_nppp_df = pd.DataFrame(X_test['teacher_number_of_previously_posted_proje
fp_nppp_df = test_nppp_df.iloc[false_positives,:]
fp_nppp_df.head(5)
```

Out[116]:

	teacher_number_of_previously_posted_projects
496	25
19356	16
23147	0
26897	3
19171	6

In [117]:

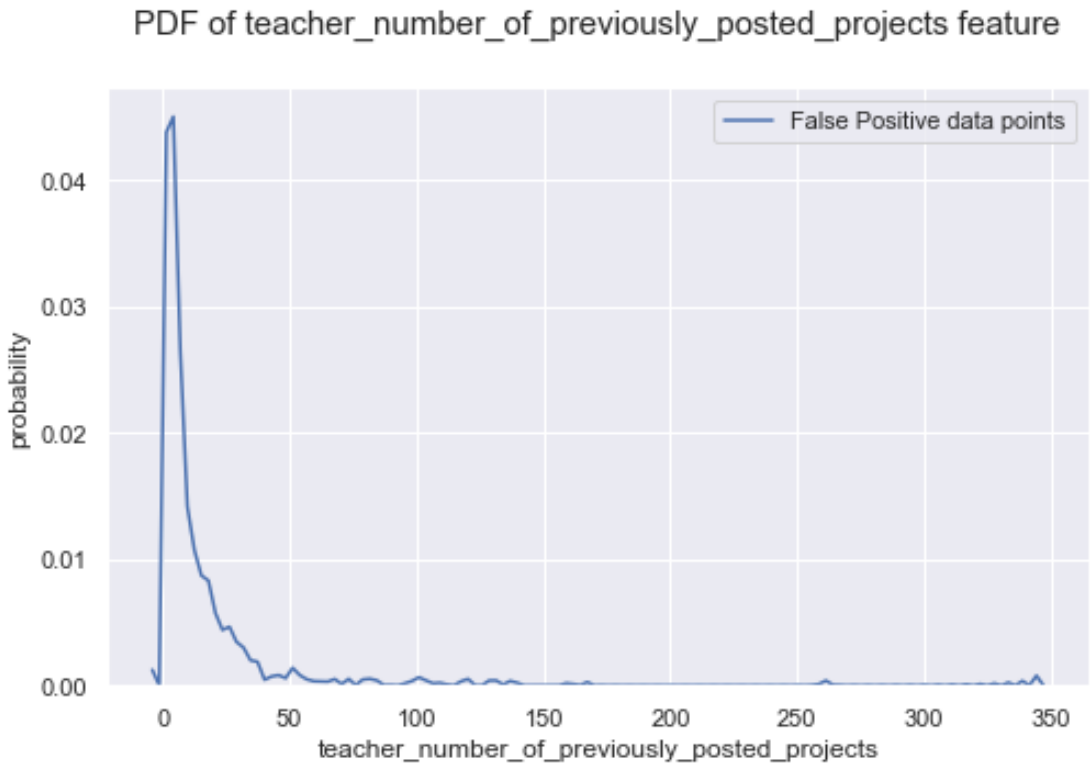
```
len(fp_nppp_df)
```

Out[117]:

1265

In [118]:

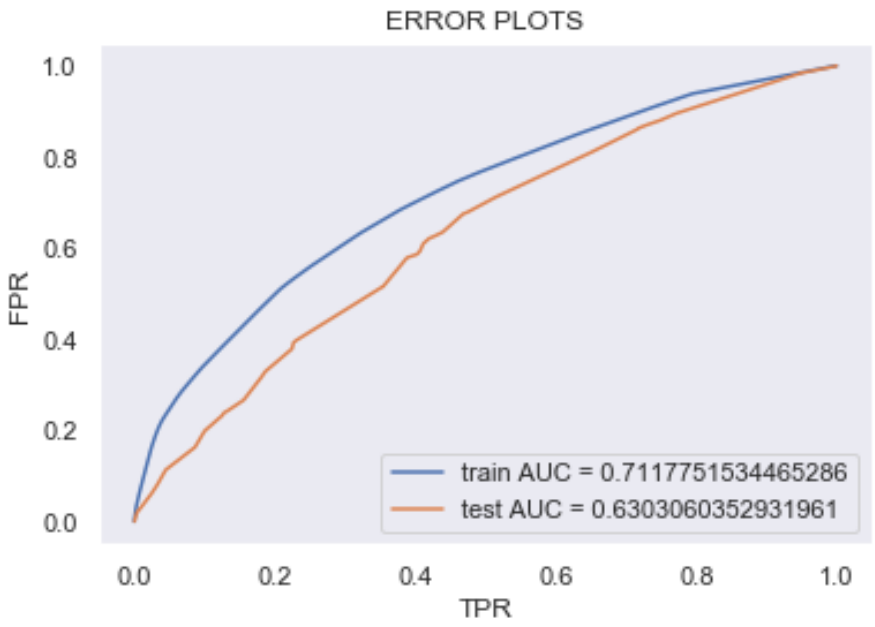
```
plt.figure(figsize=(8,5))
sns.distplot(fp_nppp_df.values, hist=False, label="False Positive data points
plt.title('PDF with the Teacher_number_of_previously_posted_projects for the
plt.xlabel('Teacher_number_of_previously_posted_projects')
plt.ylabel('probability')
plt.legend()
plt.xlabel('teacher_number_of_previously_posted_projects')
plt.title('PDF of teacher_number_of_previously_posted_projects feature', size
plt.show()
```



Applying Decision Tree with obtained best Hyper parameter on TFIDF W2V representation

In [119]:

```
dt = DecisionTreeClassifier(max_depth= best_max_depth_tw2v, min_samples_split
dt.fit(X_train_tw2v_matrix, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability esti
# not the predicted outputs

y_train_pred = batch_predict(dt, X_train_tw2v_matrix)
y_test_pred = batch_predict(dt, 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()
```

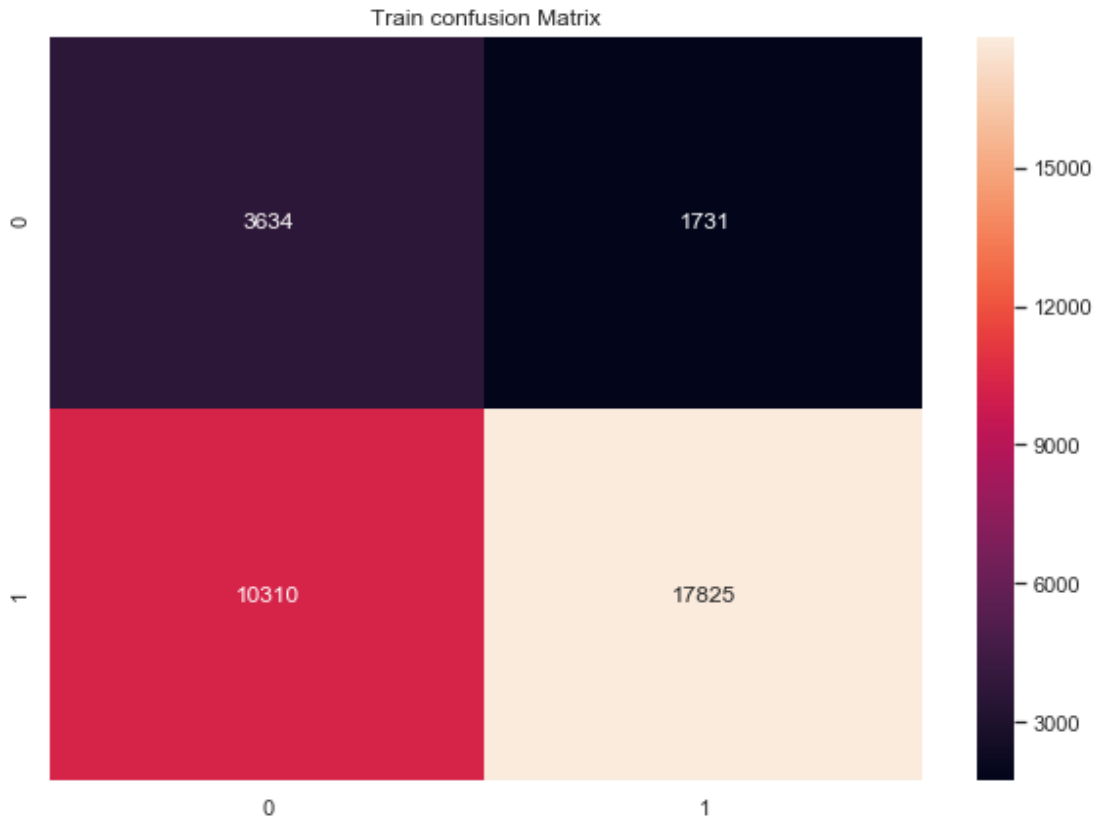


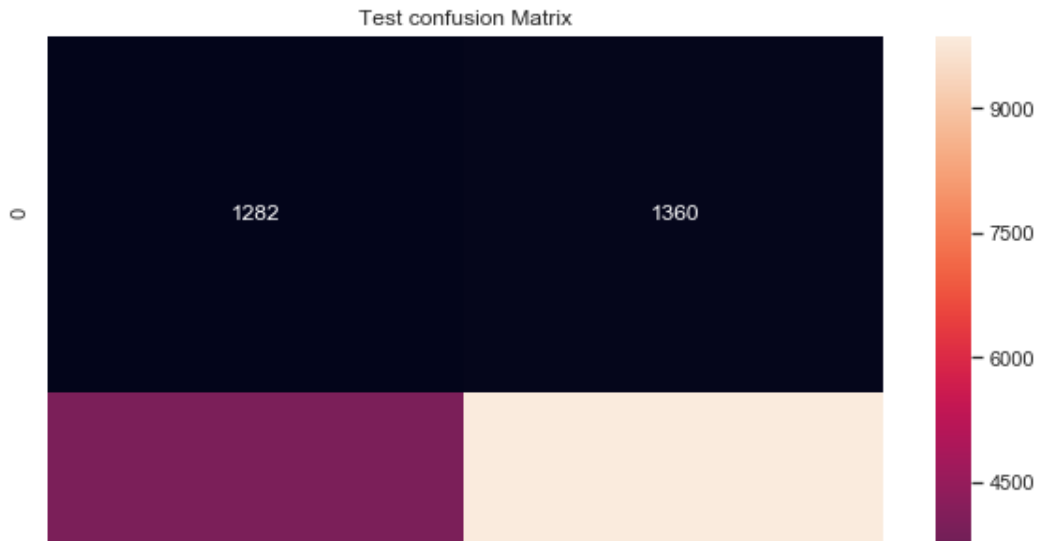
Confusion Matrix with predicted and original labels for TFIDF W2V

In [120]:

```
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) 0.42913883285735743 for thres
hold 0.505
```





In [121]:

```
predictions = predict_with_best_t(y_test_pred,best_t)
len(predictions)
```

Out[121]:

16500

In [122]:

```
act_vs_predicted = pd.DataFrame({'index' : X_test.index, 'actual_label':y_tes
act_vs_predicted.head(20)
```

Out[122]:

	index	actual_label	predicted_label
0	33818	1	1
1	6140	1	0
2	11555	1	0
3	23477	1	1
4	31007	1	0
5	49771	1	0
6	20399	1	1
7	35350	1	1
8	44517	1	1
9	33698	1	1
10	6332	1	1
11	15025	1	0
12	44119	1	1
13	496	0	1
14	19356	0	1
15	11569	1	0
16	23147	0	1
17	4775	1	1
18	25411	1	1
19	2472	1	1

In [123]:

```
false_positives = []
for i in tqdm(range(len(act_vs_predicted))):
    if(act_vs_predicted['actual_label'][i]==0 and act_vs_predicted['predicted
        false_positives.append(i)

len(false_positives)
```

```
100%|██████████████████████████████████████████████████████████████
██████████████| 16500/16500 [00:00<00:00, 86421.42it/s]
```

Out[123]:

```
1360
```

In [124]:

```
false_positives_essay = []
for i in false_positives :
    false_positives_essay.append(X_test['essay'].values[i])
```

In [125]:

```
false_positives_essay[0]
```

Out[125]:

```
'My students are eager, not only for musical knowledge, but fo
r playing musical instrument as well.Presently, I teach elemen
tary/middle school music (4-8) at a high needs public school i
n Illinois. Due to budget cuts, the school is unable to fund a
music program for the school. Music, being an interdisciplinar
y subject, encompasses all subjects students encounter during
their education. The students at this school are very enthusia
stic about music and should have a top-notch music program. Th
is can only be possible with your help!The students will learn
how to work together in an ensemble which encourages team wor
k. They will also receive the satisfaction of learning how to
play a musical instrument. Music is very entertaining and allo
ws for freedom of expression. My students deserve the opportun
ity to express themselves through music, and playing the trump
et will provide that foundation.Music is the universal languag
e, and provides an experience that will allow the students to
appreciate the creative process, and develop a commitment to e
xcellence. Your donations will help the students to experience
music on a greater level through their performance.'
```

In [126]:

```
comment_words = ' '
stopwords = set(STOPWORDS)
for val in false_positives_essay :
    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 + ' '
```

In [127]:

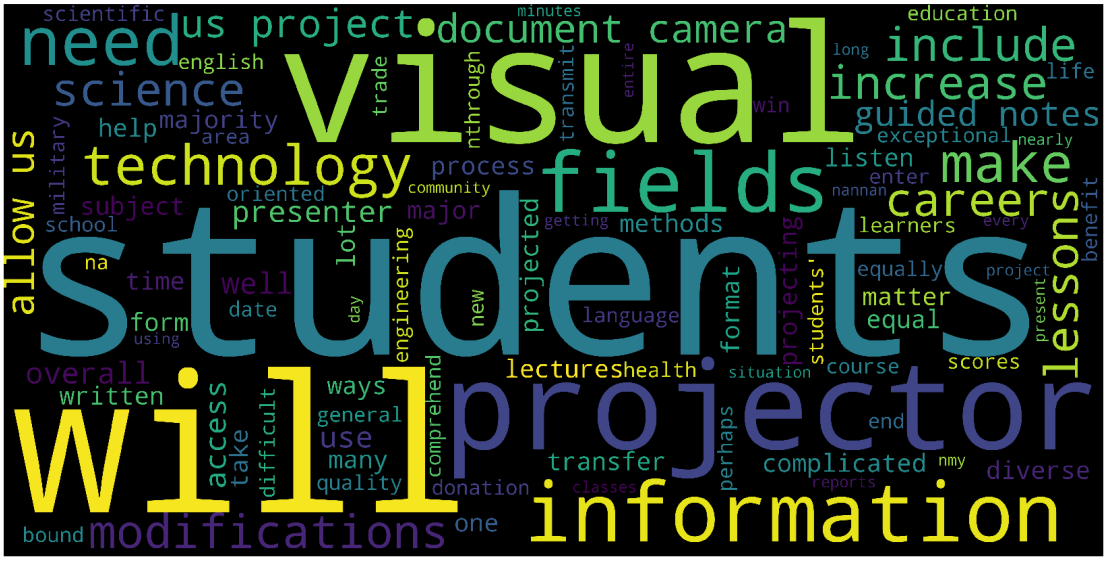
```
len(comment_words)
```

Out[127]:

2803

In [128]:

```
wc = WordCloud(width=5000,height=2500,background_color="black",stopwords=stop
wc.generate(comment_words)
plt.figure( figsize=(30,15))
plt.imshow(wc)
plt.axis("off")
plt.show()
```



In [129]:

```
test_price_df = pd.DataFrame(X_test['price'])
fp_price_df = test_price_df.iloc[false_positives,:]
fp_price_df.head(5)
```

Out[129]:

	price
496	199.98
19356	40.59
23147	37.04
19171	537.97
31151	549.00

In [130]:

```
len(fp_price_df)
```

Out[130]:

```
1360
```

In [131]:

```
sns.boxplot(y='price', data=fp_price_df)
plt.title('False Positives "Price" Box Plot', size =15, y = 1.07)
plt.show()
```

In [132]:

```
test_nppp_df = pd.DataFrame(X_test['teacher_number_of_previously_posted_proje
fp_nppp_df = test_nppp_df.iloc[false_positives,:]
fp_nppp_df.head(5)
```

Out[132]:

	teacher_number_of_previously_posted_projects
496	25
19356	16
23147	0
19171	6
31151	1

In [133]:

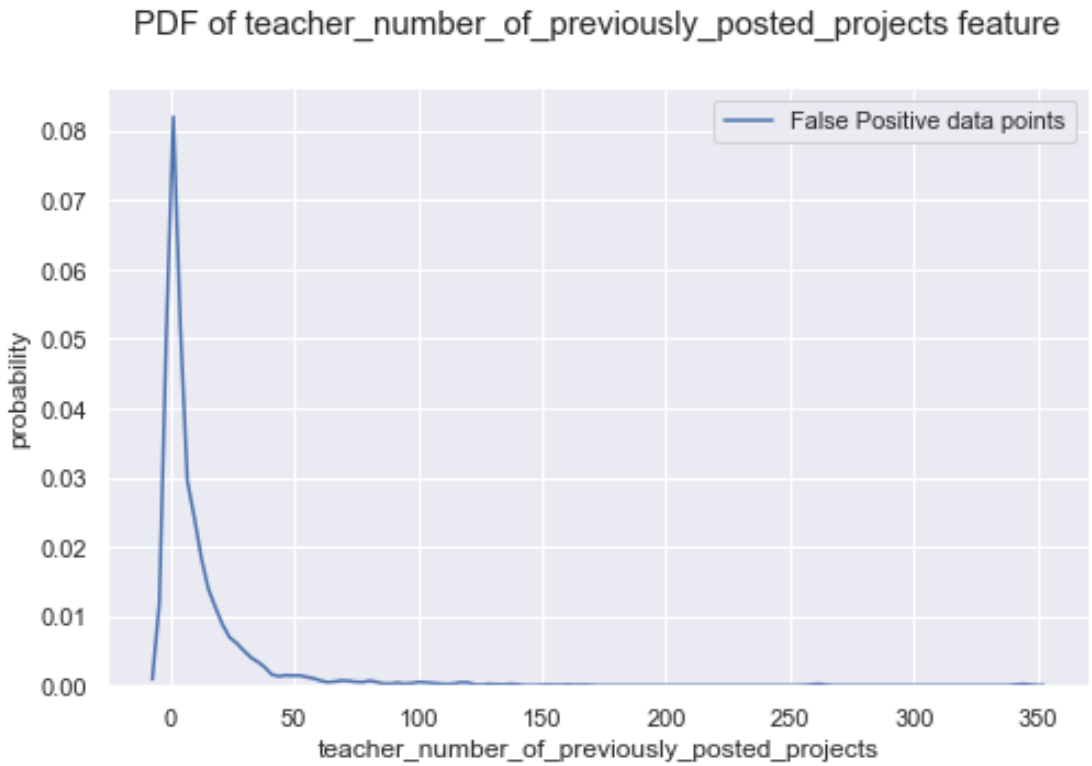
```
len(fp_nppp_df)
```

Out[133]:

```
1360
```

In [134]:

```
plt.figure(figsize=(8,5))
sns.distplot(fp_nppp_df.values, hist=False, label="False Positive data points
plt.title('PDF with the Teacher_number_of_previously_posted_projects for the
plt.xlabel('Teacher_number_of_previously_posted_projects')
plt.ylabel('probability')
plt.legend()
plt.xlabel('teacher_number_of_previously_posted_projects')
plt.title('PDF of teacher_number_of_previously_posted_projects feature', size
plt.show()
```



In [ ]:

```
```

Task-2

In [135]:

```
print(X_train_tfidf_matrix.shape)
print(X_test_tfidf_matrix.shape)
```

```
(33500, 14122)
(16500, 14122)
```

In [136]:

```
def selectKImportance(model, X, k):
     return X[:,model.feature_importances_.argsort()[::-1][:k]]
```

In [137]:

```
dt = DecisionTreeClassifier(max_depth= best_max_depth_tfidf, min_samples_spli
dt.fit(X_train_tfidf_matrix, y_train)
X_train_5k_features = selectKImportance(dt, X_train_tfidf_matrix, 5000)
X_test_5k_features = selectKImportance(dt, X_test_tfidf_matrix, 5000)
```

In [138]:

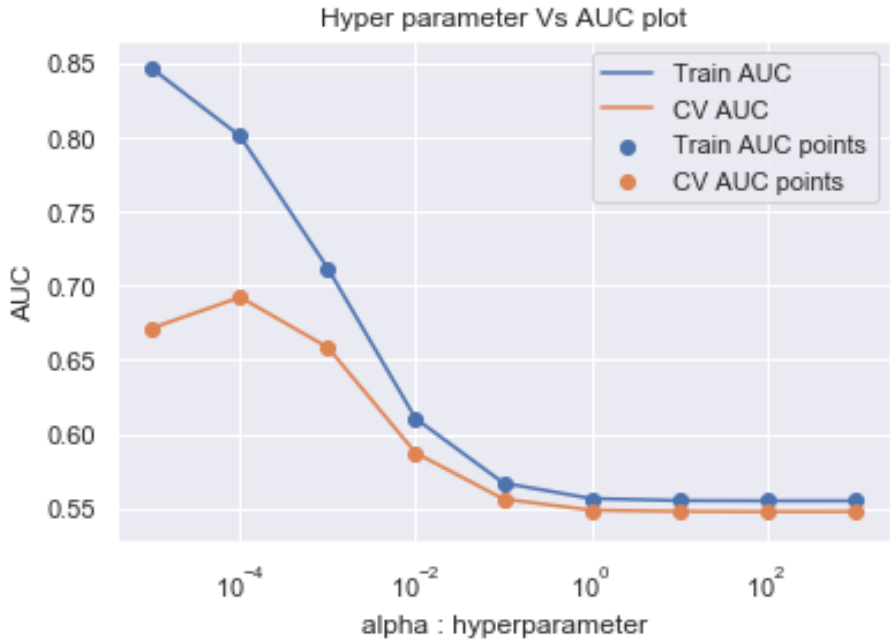
```
print('After selecting top 5K features, the shape of train matrix is',X_train
print('After selecting top 5K features, the shape of test matrix is',X_test_5
```

```
After selecting top 5K features, the shape of train matrix is
(33500, 5000)
After selecting top 5K features, the shape of test matrix is
(16500, 5000)
```

Applying Logistic Regression on top 5K features of TFIDF Matrix

In [139]:

```
from sklearn import linear_model

parameters = {'alpha':[0.00001,0.0001,0.001,0.01,0.1,1,10,10**2,10**3]}
lg = linear_model.SGDClassifier(loss='log', class_weight =  "balanced" )
clf = GridSearchCV(lg, parameters, cv=10, scoring='roc_auc', return_train_sc
clf.fit(X_train_5k_features, 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(True)
plt.show()
```



In [140]:

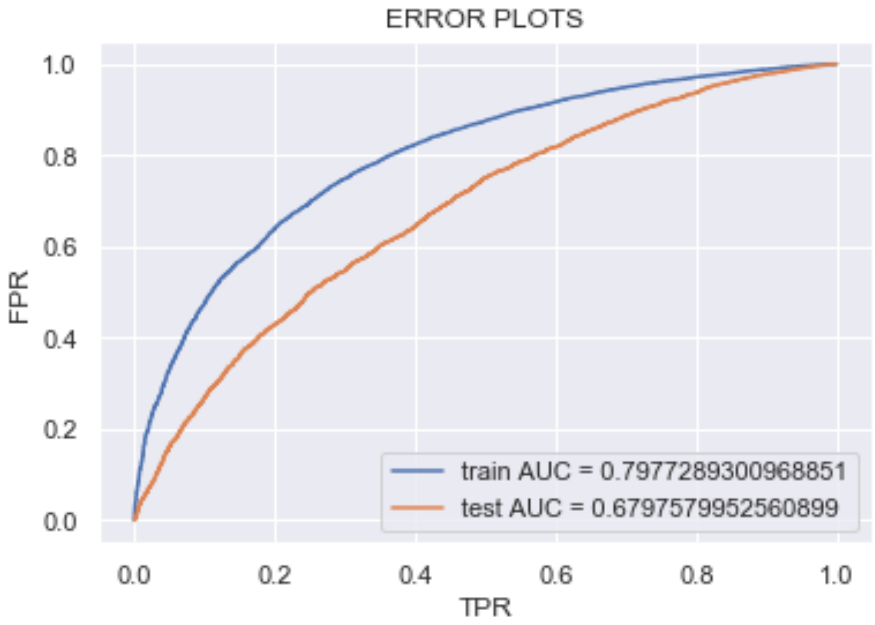
```
best_alpha = clf.best_params_['alpha']
best_alpha
```

Out[140]:

```
0.0001
```

In [141]:

```
clf = linear_model.SGDClassifier(loss = 'log', alpha = best_alpha, class_weig
clf.fit(X_train_5k_features, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability esti
# not the predicted outputs

y_train_pred = batch_predict(clf, X_train_5k_features)
y_test_pred = batch_predict(clf, X_test_5k_features)

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.show()
```

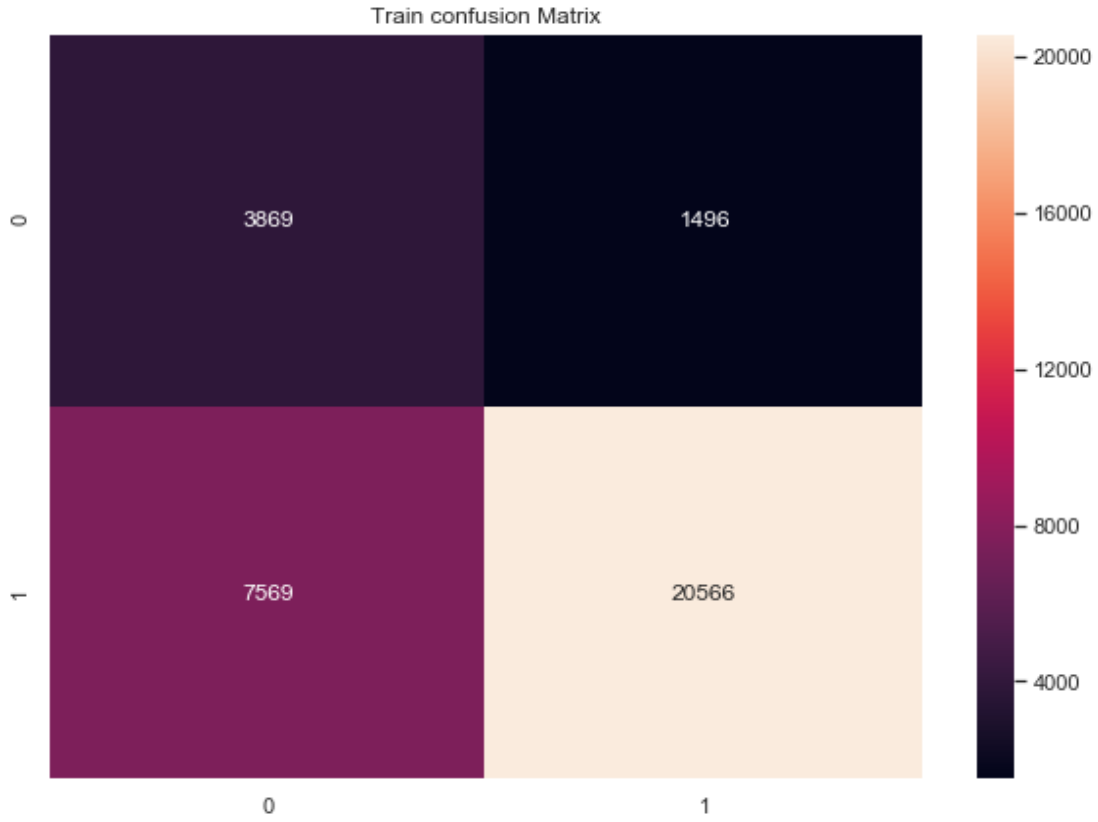


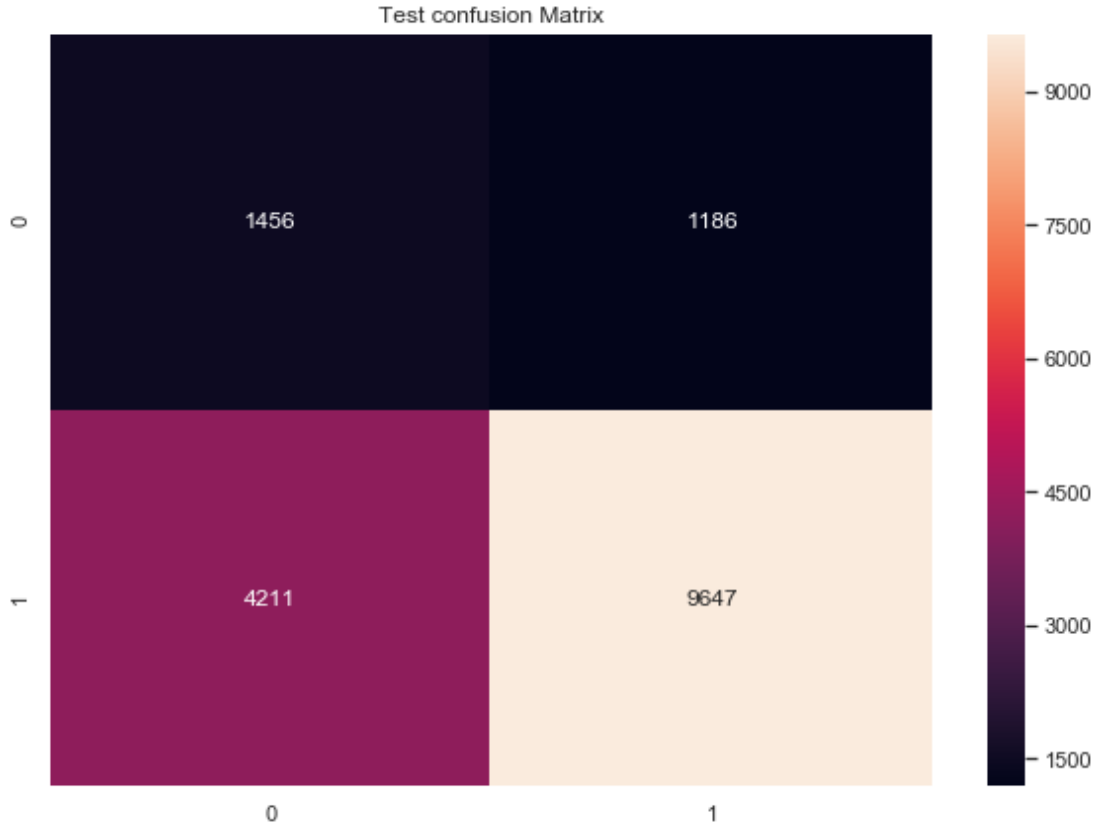
Confusion Matrix with predicted and original labels

In [142]:

```
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) 0.5271472137648148 for thresh
old 0.494
```




Test confusion Matrix
1456
1186
4211
9647
0
1
0
1
9000
7500
6000
4500
3000
1500

Conclusion

In [143]:

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

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

table.add_row(['BOW', 'Decision Trees', ('Max Depth = '+str(best_max_depth_bo
table.add_row(['TFIDF', 'Decision Trees', ('Max Depth = '+str(best_max_depth_
table.add_row(['AVG W2V', 'Decision Trees', ('Max Depth = '+str(best_max_dept
table.add_row(['TFIDF W2V', 'Decision Trees', ('Max Depth = '+str(best_max_de
table.add_row(['TFIDF with 5K Features', 'Logistic Regression', ('Alpha = '+s
print(table)
```

```
+------------------------+---------------------+-------------
--------------------+--------+
|       Vectorizer       |        Model        |         Hyper
Parameter          |  AUC   |
+------------------------+---------------------+-------------
--------------------+--------+
|          BOW           |    Decision Trees   | Max Depth =
8, Min Samples = 500 | 0.6501 |
|         TFIDF          |    Decision Trees   | Max Depth =
8, Min Samples = 500 | 0.6367 |
|        AVG W2V         |    Decision Trees   | Max Depth =
6, Min Samples = 500 | 0.6263 |
|       TFIDF W2V        |    Decision Trees   | Max Depth =
6, Min Samples = 500 | 0.6303 |
| TFIDF with 5K Features | Logistic Regression |          Alph
a = 0.0001          | 0.6797 |
+------------------------+---------------------+-------------
--------------------+--------+
```

Summary

- BOW vectorizer gave AUC 0.6501 with Max Depth = 8, Min Samples = 500
- TFIDF vectorizer gave AUC 0.6367 with Max Depth = 8, Min Samples = 500
- AVG W2V vectorizer gave AUC 0.6263 with Max Depth = 6, Min Samples = 500
- TFIDF W2V vectorizer gave AUC 0.6303 with Max Depth = 6, Min Samples = 500
- TFIDF with 5K Features vectorizer gave AUC 0.6797 with Logistic Regression and best hyper parameter Alpha = 0.0001
- BOW vectorizer has the best AUC
- TFIDF, TFIDF W2V and AVG W2V has the next best AUC respectively
- TFIDF AUC raised from 0.6367 to 0.6797 when Logistic Regression with 5K Features is used instead of Decision Trees