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]:

```
#Concatinating essay and title texts
prepeocessed_data["titles_essays"] = prepeocessed_data["essay"]+' '+ prepeoce
```

In [4]:

```
prepeocessed_data.head(3)
```

Out[4]:

	Unnamed: 0	Unnamed: 0.1	id	teacher_id	teache
0	0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	
1	1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	
2	2	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	

3 rows × 22 columns

In [5]:

```
import nltk
nltk.download('stopwords')
stopWords = stopwords.words('english')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Addu\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [6]:

```
#removing stopwords from concatinated text
import string
print('before stopwords removal\n',prepeocessed_data["titles_essays"][0])

for index, sentence in enumerate(prepeocessed_data["titles_essays"]):
    clean_sentence = []
    for word in sentence.lower().split():
        word = word.translate(str.maketrans(dict.fromkeys(string.punctuation)
        if (word not in stopWords) and (word.isalpha()) :
            clean_sentence.append(word.strip())
    prepeocessed_data["titles_essays"][index] = ' '.join(clean_sentence)

print('\nafter stopwords removal\n',prepeocessed_data["titles_essays"][0])
```

```
before stopwords removal
 I have been fortunate enough to use the Fairy Tale STEM kits
in my classroom as well as the STEM journals, which my student
s really enjoyed.  I would love to implement more of the Lakes
hore STEM kits in my classroom for the next school year as the
y provide excellent and engaging STEM lessons.My students come
from a variety of backgrounds, including language and socioeco
nomic status.  Many of them don't have a lot of experience in
science and engineering and these kits give me the materials t
o provide these exciting opportunities for my students.Each mo
nth I try to do several science or STEM/STEAM projects.  I wou
ld use the kits and robot to help guide my science instruction
in engaging and meaningful ways.  I can adapt the kits to my c
urrent language arts pacing guide where we already teach some
of the material in the kits like tall tales (Paul Bunyan) or J
ohnny Appleseed.  The following units will be taught in the ne
xt school year where I will implement these kits: magnets, mot
ion, sink vs. float, robots.  I often get to these units and d
on't know If I am teaching the right way or using the right ma
terials.    The kits will give me additional ideas, strategie
s, and lessons to prepare my students in science.It is challen
ging to develop high quality science activities.  These kits g
ive me the materials I need to provide my students with scienc
e activities that will go along with the curriculum in my clas
sroom.  Although I have some things (like magnets) in my class
room, I don't know how to use them effectively.  The kits will
provide me with the right amount of materials and show me how
to use them in an appropriate way. Engineering STEAM into the
Primary Classroom

after stopwords removal
 fortunate enough use fairy tale stem kits classroom well stem
journals students really enjoyed would love implement lakeshor
e stem kits classroom next school year provide excellent engag
ing stem lessonsmy students come variety backgrounds including
language socioeconomic status many dont lot experience science
```

engineering kits give materials provide exciting opportunities
studentseach month try several science stemsteam projects woul
d use kits robot help guide science instruction engaging meani
ngful ways adapt kits current language arts pacing guide alrea
dy teach material kits like tall tales paul bunyan johnny appl
eseed following units taught next school year implement kits m
agnets motion sink vs float robots often get units dont know t
eaching right way using right materials kits give additional i
deas strategies lessons prepare students scienceit challenging
develop high quality science activities kits give materials ne
ed provide students science activities go along curriculum cla
ssroom although things like magnets classroom dont know use ef
fectively kits provide right amount materials show use appropr
iate way engineering steam primary classroom

In [7]:

```
for i in prepeocessed_data["titles_essays"][:5]:
    print(i,'\n')
```

```
fortunate enough use fairy tale stem kits classroom well stem
journals students really enjoyed would love implement lakeshor
e stem kits classroom next school year provide excellent engag
ing stem lessonsmy students come variety backgrounds including
language socioeconomic status many dont lot experience science
engineering kits give materials provide exciting opportunities
studentseach month try several science stemsteam projects woul
d use kits robot help guide science instruction engaging meani
ngful ways adapt kits current language arts pacing guide alrea
dy teach material kits like tall tales paul bunyan johnny appl
eseed following units taught next school year implement kits m
agnets motion sink vs float robots often get units dont know t
eaching right way using right materials kits give additional i
deas strategies lessons prepare students scienceit challenging
develop high quality science activities kits give materials ne
ed provide students science activities go along curriculum cla
ssroom although things like magnets classroom dont know use ef
fectively kits provide right amount materials show use appropr
iate way engineering steam primary classroom

imagine years old youre third grade classroom see bright light
s kid next chewing gum birds making noise street outside buzzi
ng cars hot teacher asking focus learning ack need break stude
ntsmost students autism anxiety another disability tough focus
school due sensory overload emotions students lot deal school
think makes incredible kids planet kind caring sympathetic kno
w like overwhelmed understand someone else struggling openmind
ed compassionate kids someday change worldit tough one thing t
ime sensory overload gets way hardest thing world focus learni
ng students need many breaks throughout day one best items wev
e used boogie board classroom students could take break exactl
y need one regardless rooms school occupied many students need
something hands order focus task hand putty give sensory input
need order focus calm overloaded help improve motor skills mak
e school funwhen students able calm ready learn able focus lea
rn retain get sensory input need prevent meltdowns scary every
one room lead better happier classroom community able learn be
st way possible sensory tools focus

class students comes diverse learners students learn best audi
tory meansi class twentyfour kindergarten studentsrnmy student
s attend title school great majority english language learners
students come lowincome homes students receive free breakfast
lunch students enthusiastic learners often faced many types ha
rdships home school often safe themby mobile listening storage
center students able reinforce enhance learning able listen st
ories using mobile listening center help reinforce high freque
```

```
ncy words introduced addition able listen stories reinforce re
ading comprehension skills strategies amongst auditory experie
ncesa mobile listening center help keep equipment neat organiz
edready use help reinforce enhance literacy skills numerous st
udents able use center help increase student learning mobile l
earning mobile listening center

recently read article giving students choice learn already set
goals let choose sit give options sit oni teach lowincome titl
e school every year class range abilities yet age learn differ
ently different interests adhd fast learners yet eager active 
learners want need able move around room yet place comfortable
complete workwe need classroom rug use class reading time stud
ents use learning times also requested four kore kids wobble c
hairs four back jack padded portable chairs students still mov
e whole group lessons without disrupting class areas provide l
ittle ones way wiggle workingbenjamin franklin said tell forge
t teach may remember involve learn want children involved lear
ning choice sit learn giving options comfortable flexible seat
ing flexible seating flexible learning

students crave challenge eat obstacles breakfast new texts hel
p ensure materials keep challenged thinkingwe urban public ele
mentary school class comprised girls boys incorporate hands ex
periences make learning meaningful students eager curious crea
tive learners heart social justice delight teachwith new commo
n core standards adopted district students need understand aut
hors craft structure analyze framework impacts readers interac
tion text characters texts also readalouds classroom rich inne
r thinking students delve deep examine characters motives chan
ge course storythese remarkable gifts provide students complex
texts take analytical skills cull ponder would extravagant rem
arkable gift would add depth library thank considering classro
om donation going deep art inner thinking
```

In [8]:

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

Out[8]:

```
(50000, 21)
```

In [9]:

```
# 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[9]:

```
(16500, 21)
```

In [10]:

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

Out[10]:

	Unnamed: 0	Unnamed: 0.1	id	teacher_id	te
4088	4088	61295	p143458	6068ae9233bd3a1ca626fd8a0ade0092	
43590	43590	148020	p002965	aa044ece066833c95cabfd8343d5854e	

2 rows × 21 columns

In [11]:

```
y_train
```

Out[11]:

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

In [12]:

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

Out[12]:

	Unnamed: 0	Unnamed: 0.1	id	teacher_id	te
40742	40742	45131	p225833	de186e8a293facc34a2d4e41c576bb79	
41060	41060	49807	p124290	770cac9504412b020a48a7e5981e714e	

2 rows × 21 columns

In [13]:

```
y_test
```

Out[13]:

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

Selecting top 2000 words

In [14]:

```
tfidf_vect = TfidfVectorizer(max_features = 2000)
tfidf_train = tfidf_vect.fit_transform (X_train['titles_essays'])
```

In [15]:

```
top_2000 = tfidf_vect.get_feature_names()
print(len(top_2000))
print(top_2000[0:5])
```

```
2000
['abilities', 'ability', 'able', 'absolutely', 'abstract']
```

Computing Co-occurance matrix

In [16]:

```
data = ["abc def ijk pqr","pqr klm opq","lmn pqr xyz abc def pqr abc"]
df = pd.DataFrame()
df['data'] = data
df
```

Out[16]:

	data
0	abc def ijk pqr
1	pqr klm opq
2	lmn pqr xyz abc def pqr abc

In [17]:

```
top_words = ["abc", "pqr", "def"]
```

In [18]:

```
n_neighbor = 2
occ_matrix = np.zeros((3,3))
for row in (df['data'].values):
    words_in_row = row.split()
    for index,word in enumerate(words_in_row):
        if word in top_words:
            for j in range(max(index-n_neighbor,0),min(index+n_neighbor,len(w
                if words_in_row[j] in top_words:
                    occ_matrix[top_words.index(word),top_words.index(words_ir
                else:
                    pass
        else:
            pass
np.fill_diagonal( occ_matrix, 0 )
```

In [19]:

```
matrix = pd.DataFrame(occ_matrix, index=top_words, columns=top_words)
matrix
```

Out[19]:

	abc	pqr	def
abc	0.0	3.0	3.0
pqr	3.0	0.0	2.0
def	3.0	2.0	0.0

In [20]:

```
n_neighbor = 5
occ_matrix_2000 = np.zeros((2000,2000))
for row in tqdm(X_train["titles_essays"].values):
    words_in_row = row.split()
    for index,word in enumerate(words_in_row):
        if word in top_2000:
            for j in range(max(index-n_neighbor,0),min(index+n_neighbor,len(w
                if words_in_row[j] in top_2000:
                    occ_matrix_2000[top_2000.index(word),top_2000.index(words
                else:
                    pass
        else:
            pass
np.fill_diagonal( occ_matrix_2000, 0 )
```

```
100%|████████████████████████████████████████
██████████████| 33500/33500 [31:38<00:00, 17.65it/s]
```

In [21]:

```
#Covarience matrix of top 2000 words
matrix_2000 = pd.DataFrame(occ_matrix_2000, index=top_2000, columns=top_2000)
matrix_2000.head(5)
```

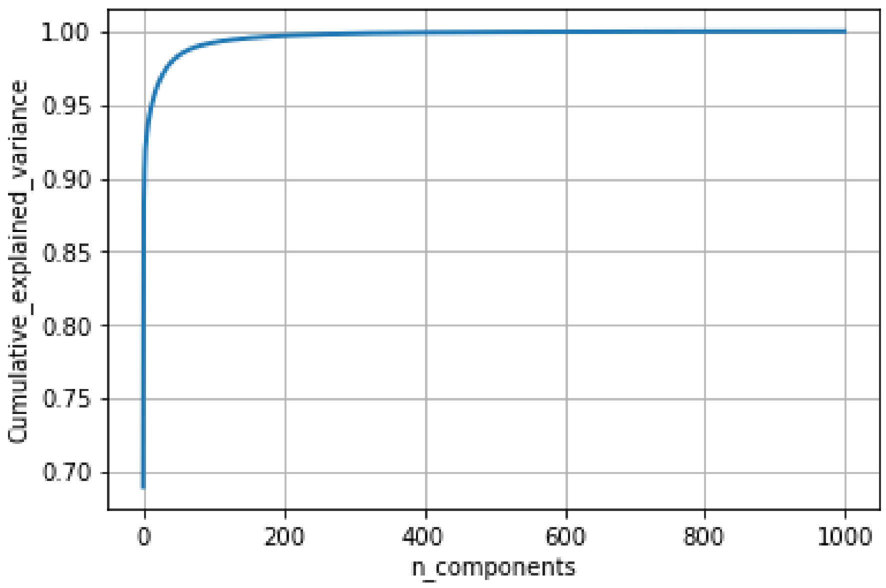
Out[21]:

```
            abilities  ability   able  absolutely  abstract  academic  academical
abilities         0.0     17.0   92.0         0.0       0.0     139.0          16
ability          17.0      0.0  160.0         6.0       4.0      98.0          21
able             92.0    160.0    0.0        14.0      17.0     228.0          58
absolutely        0.0      6.0   14.0         0.0       0.0       1.0           0
abstract          0.0      4.0   17.0         0.0       0.0       0.0           0

5 rows × 2000 columns
```

Applying TruncatedSVD and Calculating Vectors for `essay` and `project_title`

In [22]:

```
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import StandardScaler
svd = TruncatedSVD(n_components = 1000)
svd_2000 = svd.fit_transform(matrix_2000)

percentage_var_explained = svd.explained_variance_ / np.sum(svd.explained_var
cum_var_explained = np.cumsum(percentage_var_explained)
plt.figure(figsize=(6, 4))

plt.clf()
plt.plot(cum_var_explained, linewidth=2)
plt.axis('tight')
plt.grid()
plt.xlabel('n_components')
plt.ylabel('Cumulative_explained_variance')
plt.show()
```



In [23]:

```
svd = TruncatedSVD(n_components = 100)
tr_svd_2000 = svd.fit_transform(matrix_2000)
```

In [24]:

```
tr_svd_2000.shape
```

Out[24]:

```
(2000, 100)
```

```
In [25]:
```

```
(tr_svd_2000[0])
```

```
Out[25]:
```

```
array([ 1.51960729e+03, -6.22675973e+02,  2.43428247e+00, -1.8
4161820e+01,
         4.45205180e+01,  1.69899659e+02, -1.64001430e+02,  2.5
0101734e+02,
        -8.29778308e+01,  5.29525138e+00,  1.40106435e+02,  1.0
2425327e+02,
         1.37542196e+02,  1.46708079e+01, -4.96975963e+00, -5.3
6824124e+01,
        -4.81636643e+01,  3.64295726e+00, -1.94939067e+01, -2.1
9306683e+02,
        -2.39891809e+01,  2.60522606e+01, -2.84814697e+00,  1.7
5254208e+02,
        -9.95572976e+01, -1.66728478e+01, -1.96248061e+02,  4.5
3770744e+01,
         2.26899980e+01,  1.52925093e+01,  2.74738540e+01,  2.0
4875701e+01,
        -5.46004147e+01, -2.35272169e+01,  9.34089701e+00, -7.4
6717278e+01,
         1.30745557e+02,  7.69519791e+00,  1.79421589e+01, -4.5
1908949e-02,
        -1.40157395e+01,  2.05005596e+01,  7.38803599e+01, -1.8
5038648e+01,
        -2.91930976e+01, -4.46072424e+01, -6.91473917e+01, -6.9
1344363e+01,
        -5.58567933e+00,  4.86200710e+00,  1.59306807e+01, -1.0
8585916e+01,
         1.86611236e+01,  5.49395526e+01,  1.09318210e+01, -2.3
5378008e+01,
        -8.77134716e+00, -2.87890445e+01, -2.77533429e+01, -4.0
2616553e+01,
         4.84193361e+01, -2.19385190e+01, -4.84894523e+00,  3.0
6044635e+01,
        -5.47891442e+01,  1.31539720e+01,  4.01076615e+01, -7.6
5034125e+00,
         1.62029609e+01,  2.90872544e+01, -1.53090686e+01, -1.8
8494210e+01,
         3.19334672e+01, -3.34644423e+01, -3.28847498e+00,  1.0
1991796e+01,
        -8.94983206e+01, -1.55006732e+01, -3.07352786e+01, -9.4
6783638e+00,
        -1.39484467e+01, -1.22338231e+01,  1.16726217e+01,  5.4
2388745e+01,
         1.31324671e+01,  9.14362844e+00,  2.07804782e+01,  6.5
1249172e+00,
        -7.03547360e+00, -7.77281151e+00, -4.33260798e+01,  2.9
0366527e+00,
       -3.80435431e+00,  6.09480627e+01, -7.28281952e+01,  5.6
8774808e-01,
        2.49620709e+01,  2.32469514e+01,  1.93984748e+01,  3.9
2500865e+01])
```

In [26]:

```
tr_df = pd.DataFrame( tr_svd_2000, index = top_2000)
tr_df
```

Out[26]:

	0	1	2	3	4
abilities	1519.607287	-622.675973	2.434282	-18.416182	44.52051
ability	2588.318599	-1011.290726	-391.197685	-16.452346	61.78218
able	16082.547879	-6511.187083	-3312.599394	-198.184533	1030.64329
absolutely	384.362949	-113.658560	-13.405283	-8.165757	-46.59005
abstract	149.887491	-55.144631	-42.889127	7.924854	19.12975
...	...	...	...	...	..
yet	1273.511613	-287.763927	198.655412	-125.278725	-107.51933
yoga	886.756552	-294.596709	-219.003149	-12.424436	-35.40295
york	271.836280	-36.492685	144.162523	-61.907653	-32.25818
young	2248.572816	-596.033031	-93.831105	-34.866419	-213.79111
younger	351.325815	-198.054328	7.630201	-40.351303	6.82817

2000 rows × 100 columns

Creating Dictionary with top words as keys and 100D vector as values

In [27]:

```
dictionary = {name : tr_svd_2000[i] for i, name in enumerate(top_2000)}
```

In [29]:

```
dictionary['young']
```

Out[29]:

```
array([ 2.24857282e+03, -5.96033031e+02, -9.38311052e+01, -
3.48664186e+01,
       -2.13791118e+02,  9.38414438e+01, -9.24727675e+01,
9.20573928e+01,
        1.31784887e+02,  3.53551906e+01,  2.55423184e+01, -
1.93261499e+00,
        1.53084964e+02, -1.19453971e+02,  9.49010003e+00,
2.42760250e+01,
        2.85476054e+00, -3.14914577e+01, -1.40765202e+02,
4.33174042e+01,
       -2.95915936e+01,  1.07410515e+01,  4.49254098e+00, -
1.23613777e+02,
       -1.08485359e+01,  1.67411628e+01, -2.52786412e+01, -
1.40330436e+00,
        1.21589421e+01, -2.39625267e+01,  7.50733538e+01, -
3.05108646e+01,
       -9.15065877e+01, -7.47826528e+00,  4.68539118e+01,
3.33217234e+01,
        1.47363467e+02, -2.25750015e+01,  4.42770865e+01, -
3.46372881e-01,
        2.28646392e+01,  1.46077407e+00,  8.24371338e+01,
3.42481733e+01,
        1.10074630e+02,  1.53383377e+02, -4.16418854e+01,
1.24323194e+02,
       -6.64057340e+01,  6.11736363e+01, -2.58182097e+01,
1.55576471e+02,
       -1.69145064e+02, -4.21537215e+01, -1.32271029e+02,
4.46854486e+01,
        1.20404288e+01,  8.45844559e+01,  7.62790452e+01, -
1.53852593e+01,
        6.48738787e+01,  1.93316111e+01,  4.27691617e+01,
7.86013247e+01,
        1.39671337e+00,  5.62458543e+01, -1.13318636e+02,
6.74019497e+01,
        5.79618690e+01,  1.68980141e+02,  8.19535423e-01, -
3.23907789e+01,
       -5.96332500e+01,  3.00623695e+01, -5.90651228e+00, -
2.10451987e+01,
        9.77397858e+01, -2.25716177e+01,  5.01770464e+00,
3.55271236e+01,
        8.72216154e+01, -4.17330927e+01,  3.85235697e+00,
8.03387913e+01,
       -8.10427778e+00,  9.79627287e+00,  2.46327669e+01,
2.46946304e+01,
        5.69630282e+01,  1.35315285e+01, -1.13466462e+02, -
3.01776197e+01,
       -3.41722961e+01,  3.46573871e+01, -1.01434457e+01, -
5.06811704e+01,
        1.30601060e+01, -1.95285415e+01,  7.10938489e+01, -
5.13963798e+01])
```

In [30]:

```
#Building train average word 2 vec
train_vec = []
for row in tqdm(X_train['titles_essays']):
    vec = np.zeros(100)
    word_count = 0
    for word in row.split():
        if (word in top_2000):
            vec += dictionary[word]
            word_count += 1
    vec /= word_count
    train_vec.append(vec)
```

```
100%|█████████████████████████████████████████████████████████
██████████████████| 33500/33500 [01:15<00:00, 444.47it/s]
```

In [31]:

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

```
33500
100
```

In [32]:

```
train_vec[33499]
```

Out[32]:

```
array([ 1.29273051e+04,  3.53429486e+03, -2.21920707e+02,
2.95219769e+01,
         1.08350237e+02,  1.16326465e+02, -2.56147830e+02,
1.87572982e+01,
         5.88087010e+00, -8.39972607e+01,  2.55666489e+01,
5.83230598e+01,
         5.13416107e+01,  1.50975862e+01, -9.79285584e+00, -
5.75748113e+01,
        -3.85140576e+01,  9.56852586e+01, -6.30861258e+01,
6.98571772e+01,
        -2.54293795e+01,  5.43244582e+01,  5.59021961e+01, -
5.29557990e+01,
         2.22664814e+01,  1.65128873e+00, -4.41725881e+01, -
4.13408495e+01,
         5.46962102e-01,  1.05650076e+01,  5.69480031e+01,
2.36114475e+01,
         2.84036097e+01, -1.16574831e+01,  3.17628404e+01,
2.15660323e+01,
        -2.20694051e+01, -7.84301168e-02, -1.40305876e+01, -
3.14863664e+00,
        -1.93412827e+01,  1.38855052e+01, -2.30700562e+00, -
4.49070697e+01,
        -3.42443502e+01, -2.73199129e+00,  6.57735575e+00, -
2.99391342e+01,
         1.55311327e+01, -3.74598480e+01,  1.86315125e+01,
1.34452847e+01,
        -7.16404080e+00, -2.85825897e+01, -5.02228934e+00, -
4.54732618e+01,
        -8.50881036e+00,  9.00516218e+00, -1.22250070e+01,
3.71498966e+00,
        -2.17337804e+01,  3.30375482e+01,  6.72146420e-01, -
3.18488267e+00,
        -1.50449583e+01,  8.05100753e+00,  6.88379463e+00,
1.67077018e+01,
         4.05608422e+00,  1.79484124e+01, -1.13013185e+00, -
1.74016386e+01,
        -2.71956850e+01,  2.86511907e+01, -7.95173418e+00,
6.21324193e+00,
         6.48116237e+00,  1.22577079e+01, -9.45627735e+00, -
1.93332573e+01,
        -6.23245961e+00,  4.01137438e+00, -4.94513450e+00,
8.97461452e+00,
         1.24604307e+01,  4.53849222e+00, -2.10753351e+01, -
5.05330003e+00,
        -1.33870726e+01, -9.35310903e+00,  1.87618744e+00,
2.39146078e-01,
        -1.10746618e+01, -7.04136821e+00, -5.41000251e+00, -
4.16960541e+00,
         1.79879592e+00, -1.06883147e+01, -4.46811365e+00, -
```

In [33]:

```python
#Building test average word 2 vec
test_vec = []
for row in tqdm(X_test['titles_essays']):
    vec = np.zeros(100)
    word_count = 0
    for word in row.split():
        if (word in top_2000):
            vec += dictionary[word]
            word_count += 1
    vec /= word_count
    test_vec.append(vec)
```

```
100%|██████████████████████████████████████████████████████████
██████████████████| 16500/16500 [00:37<00:00, 444.34it/s]
```

In [34]:

```python
print(len(test_vec))
print(len(test_vec[0]))
```

```
16500
100
```

In [35]:

```
test_vec[0]
```

Out[35]:

```
array([ 1.18267979e+04,  1.88408025e+03, -4.21825550e+02,
2.03286172e+02,
         1.50924544e+02, -2.66424885e+01,  1.41717411e+02,
1.09426346e+02,
        -5.06360238e+01, -9.56393808e+01, -1.59659417e+02, -
4.76120399e+01,
        -4.26919591e+01,  2.26817708e+01, -3.29140155e+00,
1.28779843e+01,
         3.37061562e+00, -2.53042506e+01,  1.41928119e+02,
1.23598428e+01,
         3.40093823e+00,  4.84071038e+01, -1.62113301e+01,
2.74168890e+00,
        -2.43924276e+01,  3.36201426e+01,  1.55674176e+01, -
1.65408012e+01,
         1.21900726e+00, -5.87831319e-01, -4.14834613e+01,
6.11800146e+00,
        -3.61256818e+01, -8.73453968e+00,  2.47015741e+01, -
1.86148216e+01,
        -1.01897585e+01, -8.96588798e+00,  9.68883501e+00,
5.91585842e+00,
         3.23319199e+01, -1.05351719e+00, -1.13557447e+01,
3.87561145e+01,
         2.91777918e+01, -1.65034939e+01, -2.45574819e+01, -
1.77889407e+01,
        -1.74209437e+01, -1.24897094e+01, -1.43794232e+01, -
1.11253137e+01,
         7.75775932e+00, -4.56220967e+01, -4.03481276e+00, -
2.61792384e+01,
        -1.98278344e+01, -1.03493069e+01, -1.03671188e+01, -
2.13683379e+00,
         2.65805291e+01, -2.44616809e+01, -6.85432939e+00, -
1.52792428e+01,
         4.13681940e+00, -2.68647714e+00, -1.53768005e+01, -
1.38507916e+01,
         3.99183217e+00,  1.11292575e+01,  7.10257149e+00, -
6.10104789e+00,
        -9.38827922e+00, -6.19947296e+00, -1.17066980e+00,
1.17897759e+01,
        -1.34151746e+01,  6.92180895e+00,  1.39539140e+01, -
1.61668909e+01,
         9.13967470e+00, -6.30809427e-02,  3.87569345e+00,
6.90783097e-01,
         6.47569870e+00, -1.80683584e+00, -5.80312823e+00,
2.36733240e+00,
         2.81592595e-01,  9.15001051e+00,  1.42690201e+01,
2.26312103e+00,
       -3.67170396e+00, -6.09193854e+00, -5.74692762e+00,
6.74361246e+00,
       -8.30452939e+00, -8.69890817e+00, -1.58587643e+01,
9.74042811e-01])
```

1.4 Encoding Categorical and Numerical features

1.4.1 encoding categorical features: clean_categories

In [36]:

```
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, 9) (33500,)
(16500, 9) (16500,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_c
ivics', 'literacy_language', 'math_science', 'music_arts', 'sp
ecialneeds', 'warmth']
```

1.4.2 encoding categorical features: clean_subcategories

In [37]:

```
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, 30) (33500,)
(16500, 30) (16500,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civi
cs_government', 'college_careerprep', 'communityservice', 'ear
lydevelopment', 'economics', 'environmentalscience', 'esl', 'e
xtracurricular', 'financialliteracy', 'foreignlanguages', 'gym
_fitness', 'health_lifescience', 'health_wellness', 'history_g
eography', 'literacy', 'literature_writing', 'mathematics', 'm
usic', 'nutritioneducation', 'other', 'parentinvolvement', 'pe
rformingarts', 'socialsciences', 'specialneeds', 'teamsports',
'visualarts', 'warmth']
```

1.4.3 encoding categorical features: school_state

In [38]:

```
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 [39]:

```
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 [40]:

```
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 [41]:

```
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,)
[[0.0033889 ]
 [0.00672089]
 [0.00168384]
 ...
 [0.000152  ]
 [0.00405855]
 [0.0019169 ]]
[[0.00252062]
 [0.00522853]
 [0.00360656]
 ...
 [0.01107748]
 [0.00350797]
 [0.00239283]]
```

1.4.7 encoding numerical features: teacher_number_of_previously_posted_projects

In [42]:

```
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,)
[[0.0002097 ]
 [0.0035649 ]
 [0.        ]
 ...
 [0.0031455 ]
 [0.0014679 ]
 [0.01342082]]
[[0.00093488]
 [0.00467438]
 [0.002493  ]
 ...
 [0.00436276]
 [0.        ]
 [0.        ]]
```

1.4.8 encoding numerical features: quantity

In [43]:

```
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,)
[[0.00165646]
 [0.00182211]
 [0.00165646]
 ...
 [0.00182211]
 [0.00513503]
 [0.00314727]]
[[0.00137874]
 [0.00091916]
 [0.01034052]
 ...
 [0.00160853]
 [0.00229789]
 [0.00114895]]
```

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

In [44]:

```python
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
0.181
0.15
(33500,)
(16500,)
```

In [45]:

```
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(-

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,)
[[0.00089821]
 [0.00709585]
 [0.00323355]
 ...
 [0.00664675]
 [0.00604794]
 [0.00622758]]
[[0.00470225]
 [0.00320608]
 [0.00679688]
 ...
 [0.00803657]
 [0.00803657]
 [0.00829305]]
```

1.4.10 encoding numerical features: number of words in the title

In [46]:

```
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
9
9
(33500,)
(16500,)
```

In [47]:

```
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,)
[[0.00486006]
 [0.00388805]
 [0.00291604]
 ...
 [0.00291604]
 [0.00583207]
 [0.00486006]]
[[0.0083376]
 [0.006948 ]
 [0.0055584]
 ...
 [0.0055584]
 [0.0083376]
 [0.0083376]]
```

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

In [48]:

```
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
207
207
(33500,)
(16500,)
```

In [49]:

```
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,)
[[0.00723098]
 [0.00548981]
 [0.00368719]
 ...
 [0.00501867]
 [0.00374864]
 [0.00469092]]
[[0.00782864]
 [0.00584965]
 [0.01085533]
 ...
 [0.00663543]
 [0.00616978]
 [0.00561683]]
```

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

In [50]:

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

X_train_matrix = hstack((X_train_cc_ohe, X_train_csc_ohe, X_train_grade_ohe, 
                             X_train_teacher_ohe, X_train_price_norm, X_trair
                             X_train_ewc_norm, X_train_twc_norm, X_train_ss_r
                             train_vec)).tocsr()

X_test_matrix = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_grade_ohe, X_te
                            X_test_teacher_ohe, X_test_price_norm, X_test_ppp
                            X_test_ewc_norm, X_test_twc_norm, X_test_ss_norm,
                            test_vec)).tocsr()

print("Final Data matrix")
print(X_train_matrix.shape, y_train.shape)
print(X_test_matrix.shape, y_test.shape)
```

```
Final Data matrix
(33500, 205) (33500,)
(16500, 205) (16500,)
```

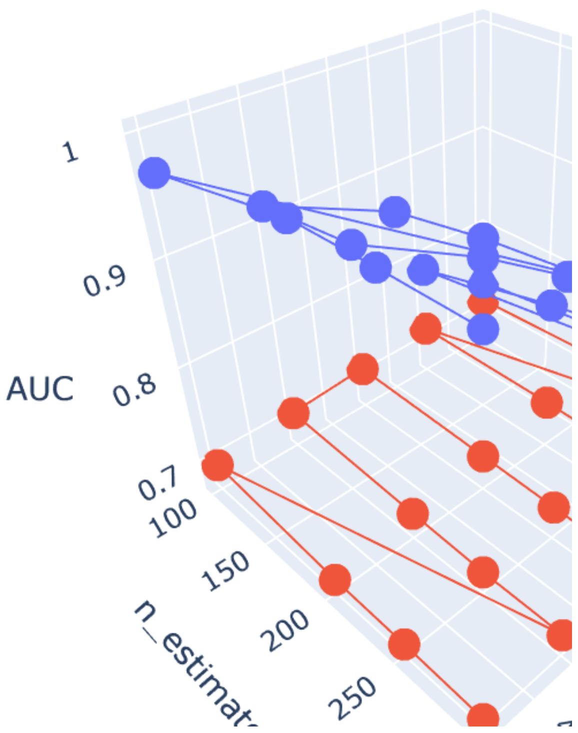
Finding Best Hyper parameters using K-Fold CV

In [51]:

```
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
parameters = {'max_depth' : [1, 2, 3, 4, 5], 'n_estimators' : [100, 200, 250,
xgbdt = xgb.XGBClassifier()
clf = GridSearchCV(xgbdt, parameters, cv=5, scoring='roc_auc', return_train_s
clf.fit(X_train_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_n_estimators']
```

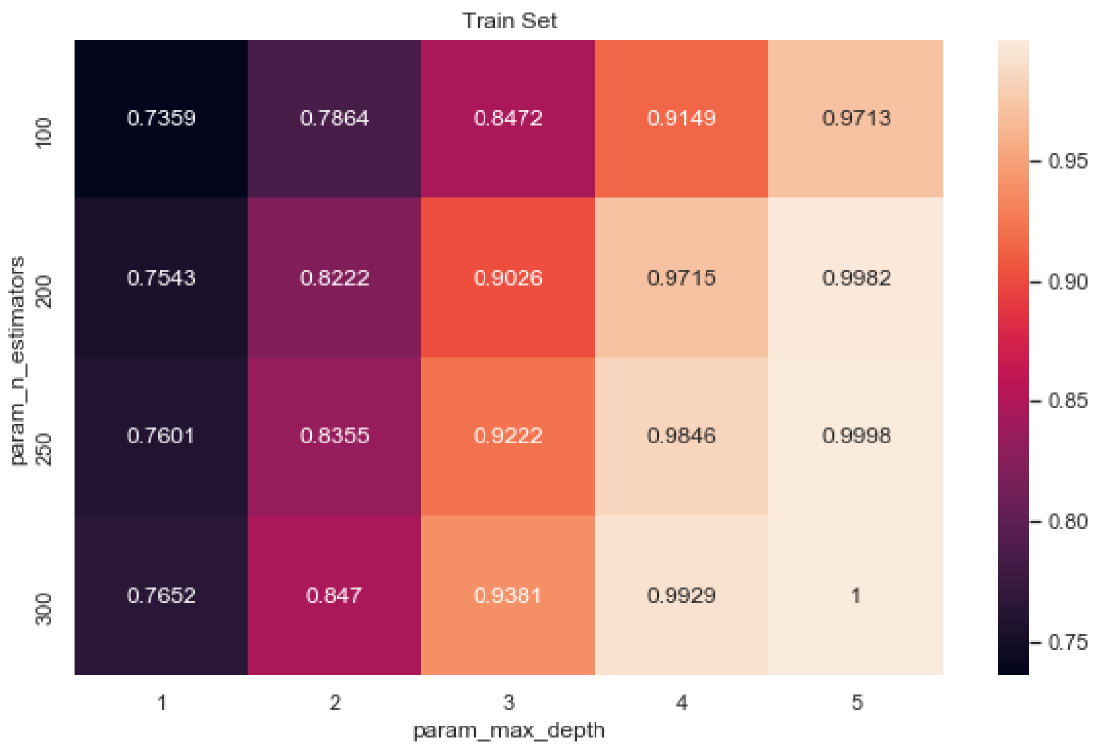
In [52]:

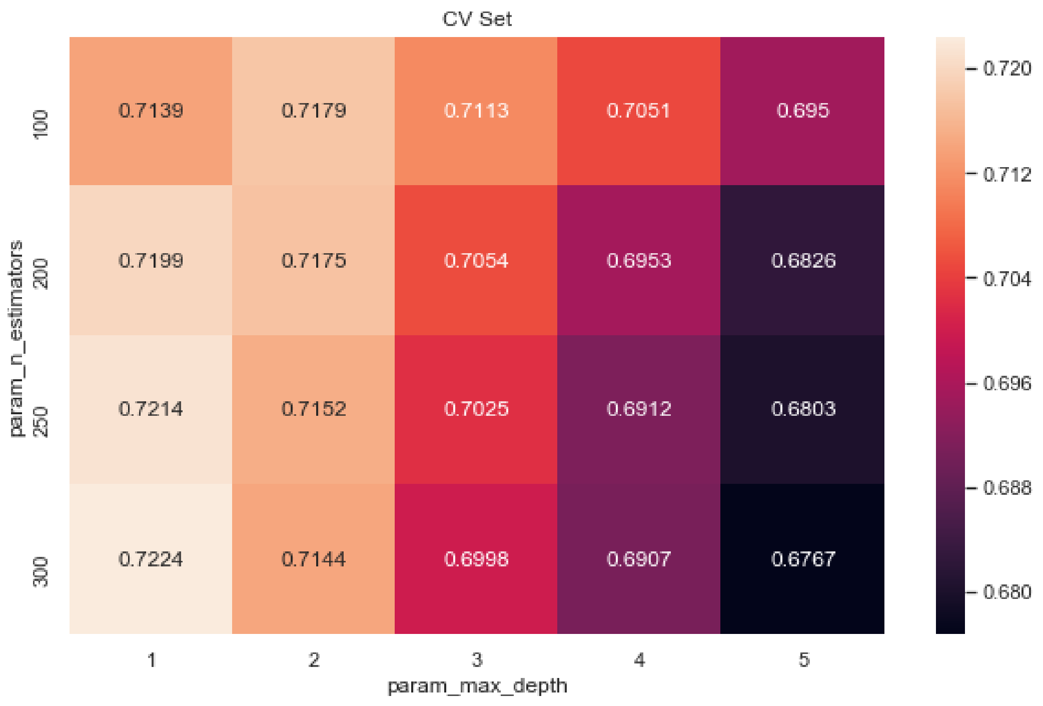
```
trace1 = go.Scatter3d(x = K, y = M, z = train_auc, name = 'Train')
trace2 = go.Scatter3d(x = K, y = M, z = cv_auc, name = 'Cross Validation')
data = [trace1, trace2]

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_n_estimators', 'p
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 = clf.best_params_['max_depth']
best_n_estimators = clf.best_params_['n_estimators']
print('best value for max depth is {} and best value for n_estimators is {}'.
```

```
best value for max depth is 1 and best value for n_estimators
is 300
```

In [ ]:

In [55]:

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

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

    return y_data_pred
```

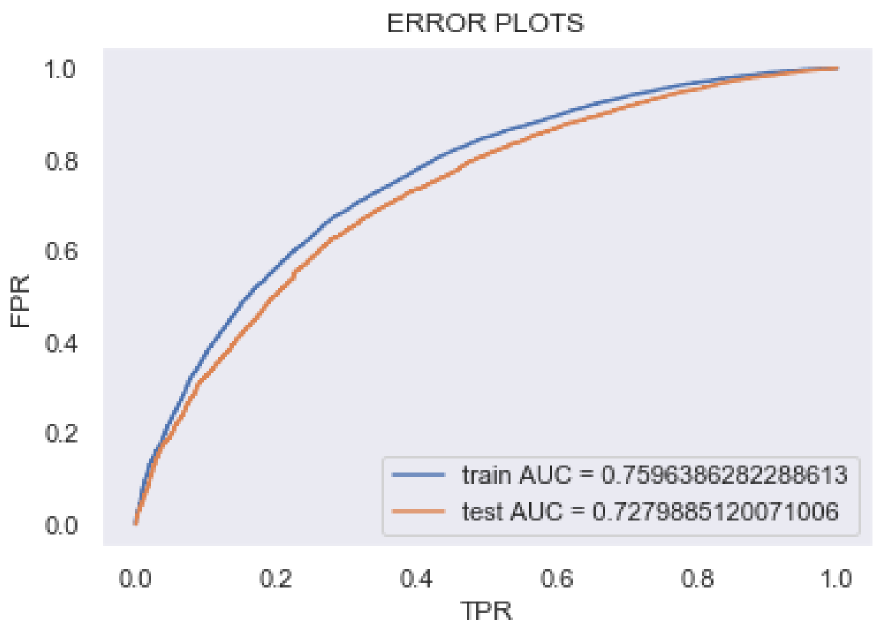
In [56]:

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

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

Applying GBDT with obtained best Hyper parameters

In [57]:

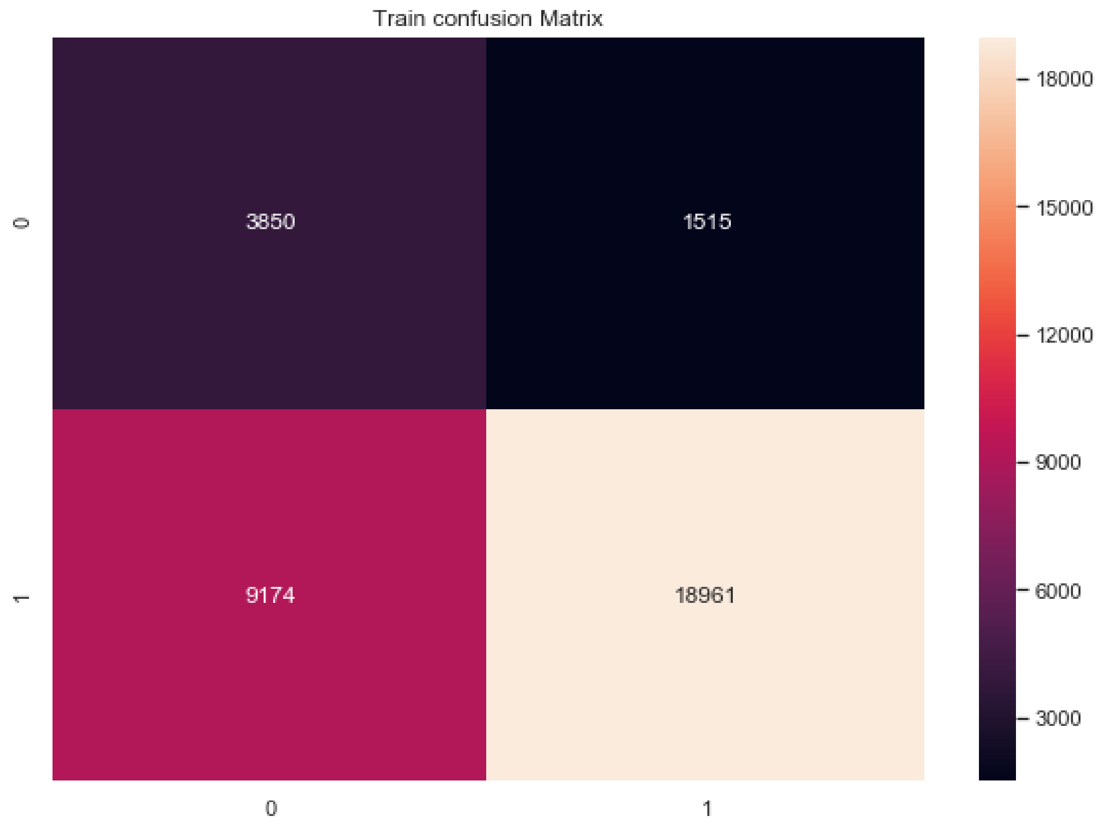
```
xgbdt = xgb.XGBClassifier(max_depth = best_max_depth, n_estimators = best_n_e
xgbdt.fit(X_train_matrix, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability esti
# not the predicted outputs

y_train_pred = batch_predict(xgbdt, X_train_matrix)
y_test_pred = batch_predict(xgbdt, X_test_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()
```

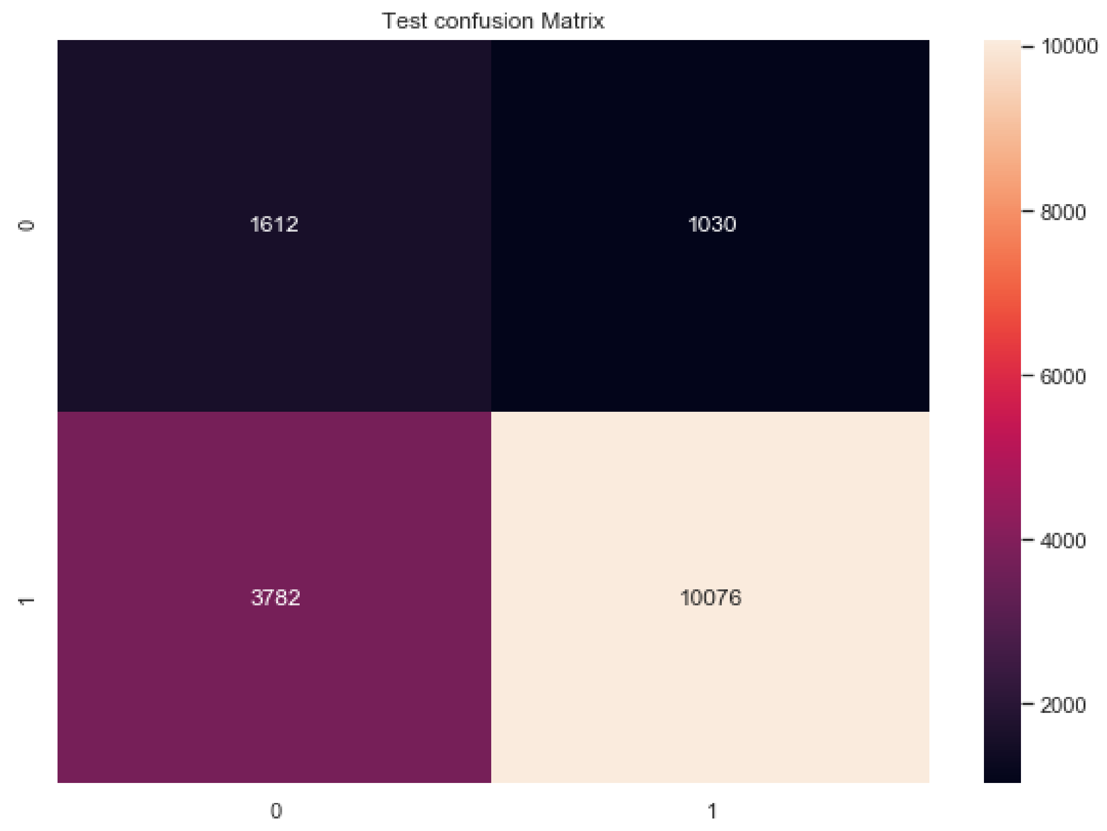


In [58]:

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

#https://stackoverflow.com/a/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) 0.4836211906678805 for thresh
old 0.836
```




Test confusion Matrix
1612
1030
3782
10076
0
1
0
1
10000
8000
6000
4000
2000

Conclusion

In [53]:

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

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

table.add_row(['AVG W2V', 'GBDT', ('Max Depth = '+str(best_max_depth) + ', n_
print(table)
```

```
+------------+-------+-------------------------------------+----
----+
| Vectorizer | Model |           Hyper Parameters          |  AU
C   |
+------------+-------+-------------------------------------+----
----+
|  AVG W2V   |  GBDT | Max Depth = 1, n_estimators = 300 | 0.7
189 |
+------------+-------+-------------------------------------+----
----+
```

Summary

- Concatinated titles and essays.
- Selected top 2k words based on their IDF values from the concatinated text.
- Used Truncated SVD to reduce dimensions to 100 which explains more than 95% varience.
- Calculated AVG W2V with the dictionary made from top 2000 words with 100 dimensions.
- The obtained Vectorizer gave AUC 0.7279 with Max Depth = 1, Min Samples = 300