Apply SVD on donors choose

In [1]:

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

1.1 Loading Data

In [2]:

```
data = pd.read_csv('preprocessed_data.csv', nrows=50000)
data.head(2)
```

Out[2]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_s
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL

2 rows × 29 columns

In [3]:

```
data.columns
```

Out[3]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_ti
tle',
       'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approve
d',
       'clean_categories', 'clean_subcategories', 'essay', 'price', 'quant
ity',
       'Numerical digits in summary', 'titles_sw', 'essays_sw',
       'preprocessed_project_grade_category', 'preprocessed_essays',
       'preprocessed_titles', 'sentimental_score',
       'preprocessed_essay_word_count', 'preprocessed_title_word_count'],
      dtype='object')
```

In [4]:

```
data['project_is_approved'].value_counts()
```

Out[4]:

```
1    42286
0     7714
Name: project_is_approved, dtype: int64
```

In [5]:

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

Out[5]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_s
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL

2 rows × 28 columns

1.2 Concatenating preprocessed essays & titles

In [6]:

```
X['preprocessed_essays'][0]
```

Out[6]:

```
'students english learners working english second third languages melting
pot refugees immigrants native born americans bringing gift language schoo
l 24 languages represented english learner program students every level ma
stery also 40 countries represented families within school student brings
wealth knowledge experiences us open eyes new cultures beliefs respect lim
its language limits world ludwig wittgenstein english learner strong suppo
rt system home begs resources many times parents learning read speak engli
sh along side children sometimes creates barriers parents able help child
learn phonetics letter recognition reading skills providing dvd players st
udents able continue mastery english language even one home able assist fa
milies students within level 1 proficiency status offered part program edu
cational videos specially chosen english learner teacher sent home regular
ly watch videos help child develop early reading skills parents access dvd
player opportunity check dvd player use year plan use videos educational d
vd years come el students nannan'
```

In [7]:

```
X['preprocessed_titles'][0]
```

Out[7]:

```
'educational support english learners home'
```

In [8]:

```
X['combined']=X['preprocessed_essays']+' '+X['preprocessed_titles']
```

In [9]:

```
X['combined'][0]
```

Out[9]:

```
'students english learners working english second third languages melting
pot refugees immigrants native born americans bringing gift language schoo
l 24 languages represented english learner program students every level ma
stery also 40 countries represented families within school student brings
wealth knowledge experiences us open eyes new cultures beliefs respect lim
its language limits world ludwig wittgenstein english learner strong suppo
rt system home begs resources many times parents learning read speak engli
sh along side children sometimes creates barriers parents able help child
learn phonetics letter recognition reading skills providing dvd players st
udents able continue mastery english language even one home able assist fa
milies students within level 1 proficiency status offered part program edu
cational videos specially chosen english learner teacher sent home regular
ly watch videos help child develop early reading skills parents access dvd
player opportunity check dvd player use year plan use videos educational d
vd years come el students nannan educational support english learners hom
e'
```

In [10]:

```
X.head(2)
```

Out[10]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_s
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL

2 rows × 29 columns

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

In [11]:

```
# 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, stratify=y)
```

In [12]:

```
print("Shape of train & test data:")
print("Train:", X_train.shape, y_train.shape)
print("Test:", X_test.shape, y_test.shape)
```

```
Shape of train & test data:
Train: (33500, 29) (33500,)
Test: (16500, 29) (16500,)
```

1.4 Make Data Model Ready: encoding combined column containing preprocessed essays & project_title using TFIDF

In [13]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer1 = TfidfVectorizer(min_df=10,ngram_range=(1,2), max_features=5000)
vectorizer1.fit(X_train['combined'].values.astype('U'))

X_train_comb_tfidf = vectorizer1.transform(X_train['combined'].values.astype('U'))
```

In [14]:

```
f1=vectorizer1.get_feature_names()
print("After vectorization")
print(X_train_comb_tfidf.shape, y_train.shape)
print("="*100)
```

```
After vectorization
(33500, 5000) (33500,)
====================================================================================================
```

1.4.1 Selecting top 2K words based on idf values

In [15]:

```
print(len(vectorizer1.idf_))
print(len(f1))
```

```
5000
5000
```

In [16]:

```
idf= list(vectorizer1.idf_)

indices = np.argsort(idf)[::-1] #https://stackoverflow.com/questions/16486252/is-it-possible-to-use-argsort-in-descending-order
names = vectorizer1.get_feature_names()
```

In [17]:

```
top2k_idf=[idf[i] for i in indices[0:2000] ]
top2k_words=[names[i] for i in indices[0:2000] ]
print(len(top2k_idf))
print(len(top2k_words))
```

```
2000
2000
```

In [18]:

```
words_idf= list(zip(top2k_words,top2k_idf))
```

In [19]:

```
type(words_idf)
```

Out[19]:

```
list
```

In [20]:

```
#top 20 words
print(words_idf[:20])
```

```
[('chess', 7.44903865456178), ('makey', 7.341793124208182), ('yearbook',
7.2146379487229355), ('piano', 7.115265474909732), ('ozobots', 7.037303933
44002), ('boogie', 7.024881413441463), ('breakout', 7.024881413441463),
('french', 7.024881413441463), ('guitar', 6.988513769270588), ('virtual re
ality', 6.953422449459318), ('pedometers', 6.897541991064862), ('butterfl
y', 6.886731074960646), ('dash dot', 6.876035785843898), ('deaf', 6.865453
6765133605), ('macbook', 6.8654536765133605), ('studio', 6.85498237664606
5), ('volleyball', 6.854982376646065), ('recycling', 6.854982376646065),
('calculator', 6.8242107179793114), ('baseball', 6.794357754829631)]
```

2.0 Co-occurance matrix for the 2K words

In [22]:

```
#ref:https://datascience.stackexchange.com/questions/40038/how-to-implement-word-to-word-co-occurence-matrix-in-python

unique=top2k_words;

coo_mat = np.zeros((len(unique), len(unique)))

context = []
window_size = 5

for text in X_train['combined']:
    words = str(text).split(' ')

    for i, _ in enumerate(words):
        context.append(words[i])

        if len(context) > (window_size * 2) + 1:
            context.pop(0)

        pos = int(len(context) / 2)
        for j, _ in enumerate(context):
            if context[j]  in unique and words[i] in unique:
                coo_mat[unique.index(context[j]), unique.index(words[i])] += 1


np.fill_diagonal( coo_mat, 0 )
coo_mat_df = pd.DataFrame(coo_mat)
coo_mat_df.index = unique
coo_mat_df.columns = unique
coo_mat_df.head(5)
```

Out[22]:

	chess	makey	yearbook	piano	ozobots	boogie	breakout	french	guitar
chess	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
makey	0.0	0.0	0.0	1.0	2.0	0.0	0.0	0.0	1.0
yearbook	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
piano	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0
ozobots	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 2000 columns

In [23]:

```
print(coo_mat.shape)
```

```
(2000, 2000)
```

In [24]:

```
type(coo_mat)
```

Out[24]:

```
numpy.ndarray
```

In [25]:

```
np.save('coo_mat', coo_mat)
```

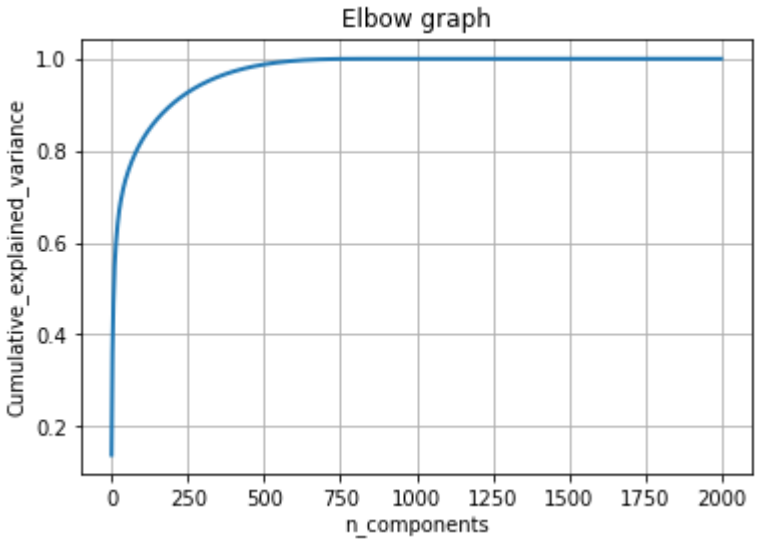
In [26]:

```
coo_mat = np.load('coo_mat.npy')
```

3.0 TruncatedSVD for dimensionality reduction

In [28]:

```
#elbow method

# initializing the SVD
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=1999)
svd.fit(coo_mat)

percentage_var_explained = svd.explained_variance_/ np.sum(svd.explained_variance_);

cum_var_explained = np.cumsum(percentage_var_explained)

# Plot the SVD spectrum
plt.figure(1, 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.title("Elbow graph")
plt.show()
```



- From the above graph it can be observed that 98% of the variance can be explained with n_components=400

In [29]:

```
from sklearn.decomposition import TruncatedSVD
svd1 = TruncatedSVD(n_components= 400 )
truncated_coo= svd1.fit_transform(coo_mat)
```

In [30]:

```
print("After reduction")
print(truncated_coo.shape)
```

```
After reduction
(2000, 400)
```

In [36]:

```
np.save('truncated_coo', truncated_coo)
#truncated_coo = np.load('truncated_coo.npy')
```

In [34]:

```
# storing the truncated co-occurence matrix in a dataframe
#index = coo_mat.index
tr_df = pd.DataFrame(truncated_coo, index = top2k_words)
tr_df.head()
```

Out[34]:

	0	1	2	3	4	5	6	
chess	0.074584	0.103558	0.288197	1.555338	0.248253	0.401996	1.954060	-1.47
makey	0.120132	0.090230	1.315063	0.501967	0.407113	0.479036	3.486067	-2.49
yearbook	0.149600	1.128524	0.306459	0.513299	0.389743	0.484325	3.680056	-2.66
piano	0.091446	0.109265	2.684034	0.452856	0.331005	0.501073	3.382097	-2.28
ozobots	0.042112	0.065266	1.366135	0.208430	0.145700	0.171056	1.286171	-0.86

5 rows × 400 columns

4.0 Vectorizing the combined text using truncated SVD matrix.

Creating a dictionary with word as key and vector as value

In [37]:

```
word_vec= dict()
a=0
for i in tr_df.values:
    word_vec[tr_df.index[a]]=i
    a+=1
```

Vectorizing the text

In [39]:

```
features=word_vec.keys()
```

In [44]:

```
def avgw2v(data, features):

    V = [] # average word 2 vec for each essay

    for text in tqdm(data):

        svec = np.zeros(400)
        count = 0

        for word in str(text).split():
            if word in features:
                vec = word_vec[word]  # Extracting the vector for the word from the dic
tionary
                svec += vec     # Adding the vectors for all the words
                count += 1

        if count != 0:
            svec /= count  # Taking the average
        V.append(svec)

    return V
```

In [45]:

```
X_tr_avg = np.asarray(avgw2v(X_train['combined'], features))

print(X_tr_avg.shape)
```

```
100%|██████████| 33500/33500 [00:01<00:00, 20375.89it/s]
(33500, 400)
```

In [46]:

```
X_test_avg = np.asarray(avgw2v(X_test['combined'], features))

print(X_test_avg.shape)
```

```
100%|██████████| 16500/16500 [00:00<00:00, 20046.62it/s]
(16500, 400)
```

5.0 Make Data Model Ready: encoding numerical, categorical features

Encoding categorical features: School State

In [47]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_state = vectorizer.transform(X_train['school_state'].values)
X_test_state = vectorizer.transform(X_test['school_state'].values)
f5=vectorizer.get_feature_names()
print("After vectorizations")
print(X_train_state.shape, y_train.shape)
print(X_test_state.shape, y_test.shape)
print(f5)
print("="*100)
```

```
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', 'md', 'me', 'mi', 'mn', 'm
o', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'o
k', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'w
i', 'wv', 'wy']
====================================================================================================
```

Encoding categorical features: teacher_prefix

In [48]:

```
X_train['teacher_prefix'].unique()
```

Out[48]:

```
array(['Mrs.', 'Ms.', 'Mr.', 'Teacher', 'Dr.', 'none'], dtype=object)
```

In [49]:

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

X_train_teacher = vectorizer.transform(X_train['teacher_prefix'].values)
X_test_teacher = vectorizer.transform(X_test['teacher_prefix'].values)

f6=vectorizer.get_feature_names()
print("After vectorizations")
print(X_train_teacher.shape, y_train.shape)
print(X_test_teacher.shape, y_test.shape)
print(f6)
print("="*100)
```

```
After vectorizations
(33500, 6) (33500,)
(16500, 6) (16500,)
['dr', 'mr', 'mrs', 'ms', 'none', 'teacher']
====================================================================================================
```

Encoding categorical features: project_grade_category

In [50]:

```
#This step is to intialize a vectorizer with vocab from train data
#Ref: https://www.kaggle.com/shashank49/donors-choose-knn#Concatinating-all-features-(TFIDF)
from collections import Counter
my_counter = Counter()
for word in X_train['project_grade_category'].values:
    my_counter.update([word[i:i+14] for i in range(0, len(word),14)]) #https://www.geeksforgeeks.org/python-string-split/

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
project_grade_category_dict = dict(my_counter)
sorted_project_grade_category_dict = dict(sorted(project_grade_category_dict.items(), key=lambda kv: kv[1]))
```

In [51]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_project_grade_category_dict.keys()), lowercase=False, binary=True,max_features=4)
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only on train data

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

f7=vectorizer.get_feature_names()
print("After vectorizations")
print(X_train_grade.shape, y_train.shape)
print(X_test_grade.shape, y_test.shape)
print(f7)
```

```
After vectorizations
(33500, 4) (33500,)
(16500, 4) (16500,)
['Grades 6-8', 'Grades 9-12', 'Grades PreK-2', 'Grades 3-5']
```

Encoding categorical features: clean_categories

In [52]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_cat = vectorizer.transform(X_train['clean_categories'].values)
X_test_cat = vectorizer.transform(X_test['clean_categories'].values)

f8=vectorizer.get_feature_names()
print("After vectorizations")
print(X_train_cat.shape, y_train.shape)
print(X_test_cat.shape, y_test.shape)
print(f8)
print("="*100)
```

```
After vectorizations
(33500, 9) (33500,)
(16500, 9) (16500,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'lit
eracy_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
====================================================================
==========================
```

Encoding categorical features: clean_subcategories

In [53]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_subcat = vectorizer.transform(X_train['clean_subcategories'].values)
X_test_subcat = vectorizer.transform(X_test['clean_subcategories'].values)

f9=vectorizer.get_feature_names()
print("After vectorizations")
print(X_train_subcat.shape, y_train.shape)
print(X_test_subcat.shape, y_test.shape)
print(f9)
print("="*100)
```

```
After vectorizations
(33500, 30) (33500,)
(16500, 30) (16500,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_governmen
t', 'college_careerprep', 'communityservice', 'earlydevelopment', 'economi
cs', 'environmentalscience', 'esl', 'extracurricular', 'financialliterac
y', 'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_welln
ess', 'history_geography', 'literacy', 'literature_writing', 'mathematic
s', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performi
ngarts', 'socialsciences', 'specialneeds', 'teamsports', 'visualarts', 'wa
rmth']
====================================================================
==========================
```

Encoding numerical features: Price

In [54]:

```
from sklearn.preprocessing import Normalizer
normalizer1 = Normalizer()
# normalizer.fit(X_train['price'].values)
#this will rise an error Expected 2D array, got 1D array instead:
normalizer1.fit(X_train['price'].values.reshape(-1,1))

X_train_price_norm = normalizer1.transform(X_train['price'].values.reshape(-1,1))
X_test_price_norm = normalizer1.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("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

Encoding numerical features: Quantity

In [55]:

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

X_train_quantity_norm = normalizer.transform(X_train['quantity'].values.reshape(-1,1))
X_test_quantity_norm = normalizer.transform(X_test['quantity'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_quantity_norm.shape, y_train.shape)
print(X_test_quantity_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

Encoding numerical features: teacher_number_of_previously_posted_projects

In [56]:

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

X_train_projects_norm = normalizer.transform(X_train['teacher_number_of_previously_post
ed_projects'].values.reshape(-1,1))
X_test_projects_norm = normalizer.transform(X_test['teacher_number_of_previously_posted
_projects'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_projects_norm.shape, y_train.shape)
print(X_test_projects_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================
==========================
```

Encoding numerical features: sentimental_score

In [57]:

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

X_train_senti_norm = normalizer.transform(X_train['sentimental_score'].values.reshape(-
1,1))
X_test_senti_norm = normalizer.transform(X_test['sentimental_score'].values.reshape(-1,
1))

print("After vectorizations")
print(X_train_senti_norm.shape, y_train.shape)
print(X_test_senti_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================
==========================
```

Encoding numerical features: preprocessed_essay_word_count

In [58]:

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

X_train_ewc_norm = normalizer.transform(X_train['preprocessed_essay_word_count'].values
.reshape(-1,1))
X_test_ewc_norm = normalizer.transform(X_test['preprocessed_essay_word_count'].values.r
eshape(-1,1))

print("After vectorization")
print(X_train_ewc_norm.shape, y_train.shape)
print(X_test_ewc_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorization
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================
==========================
```

Encoding numerical features: preprocessed_title_word_count

In [59]:

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

X_train_twc_norm = normalizer.transform(X_train['preprocessed_title_word_count'].values
.reshape(-1,1))
X_test_twc_norm = normalizer.transform(X_test['preprocessed_title_word_count'].values.r
eshape(-1,1))

print("After vectorization")
print(X_train_twc_norm.shape, y_train.shape)
print(X_test_twc_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorization
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================
==========================
```

6.0 Concatinating all categorical features + numerical features + TruncatedSVD matrix

In [61]:

```
from scipy.sparse import hstack
X_tr_avgw2v = hstack((X_tr_avg, X_train_state, X_train_teacher, X_train_grade, X_train_
cat, X_train_subcat, X_train_price_norm, X_train_quantity_norm, X_train_projects_norm,X
_train_senti_norm,X_train_ewc_norm,X_train_twc_norm)).tocsr()


X_test_avgw2v = hstack((X_test_avg, X_test_state, X_test_teacher, X_test_grade, X_test_
cat, X_test_subcat, X_test_price_norm, X_test_quantity_norm, X_test_projects_norm,X_tes
t_senti_norm,X_test_ewc_norm,X_test_twc_norm)).tocsr()

print("Final Data Matrix")
print(X_tr_avgw2v.shape, y_train.shape)
print(X_test_avgw2v.shape, y_test.shape)
```

```
Final Data Matrix
(33500, 506) (33500,)
(16500, 506) (16500,)
```

In [63]:

```
# https://stackoverflow.com/questions/8955448/save-load-scipy-sparse-csr-matrix-in-port
able-data-format
from scipy import sparse

sparse.save_npz("X_tr_avgw2v.npz", X_tr_avgw2v)
sparse.save_npz("X_test_avgw2v.npz", X_test_avgw2v)
#X_tr_avgw2v = sparse.load_npz("X_tr_avgw2v.npz")
#X_test_avgw2v = sparse.load_npz("X_test_avgw2v.npz")
```

In [67]:

```
#https://www.geeksforgeeks.org/numpy-save/
np.save('y_train', y_train)
np.save('y_test', y_test)
```

7.0 Apply XGboost

7.1 Utility functions

In [71]:

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
 of the positive class
    # not the predicted outputs
    y_data_pred = []
    pred_labels=[]
    tr_loop = data.shape[0] - data.shape[0]%1000;
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 =
 49000
    # in this for loop we will iterate until 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 predict
ing for the last data points
        pred_labels.extend(clf.predict(data[i:i+1000]))
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])
        pred_labels.extend(clf.predict(data[tr_loop:]))

    return y_data_pred,pred_labels
```

Confusion matrix

In [72]:

```
## we will pick a threshold that will give the least fpr

def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("The maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)),"for threshold", np.roun
d(t,3))
    return t

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

In [73]:

```
#function to get heatmap of confusion matrix
# Reference: https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-ma
trix

def cm_heatmap(cm):
    #y_pred = clf.predict(X_te)
    df_cm = pd.DataFrame(cm, range(2),range(2))
    df_cm.columns = ['Predicted NO','Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})
    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='d')
```

7.2 Hyperparameter tuning

In [68]:

```
#https://dask-ml.readthedocs.io/en/stable/modules/generated/dask_ml.xgboost.XGBClassifier.html
#https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/

from sklearn.metrics import roc_auc_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from xgboost import XGBClassifier

xgb = XGBClassifier()
parameters = {'n_estimators': [4, 8, 16, 32, 64, 100],'max_depth': [4, 6, 8, 10, 20, 25]}
model = RandomizedSearchCV(xgb, parameters, cv=5, scoring='roc_auc',return_train_score=True,n_jobs=-1)
rs1 = model.fit(X_tr_avgw2v, y_train)
```

In [69]:

```
dfm=pd.DataFrame(model.cv_results_)
dfm.head(2)
```

Out[69]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max
0	38.296566	1.956063	0.556959	0.644905	4
1	116.844834	1.987120	0.564163	0.913207	8

2 rows × 22 columns

In [70]:

```
dfm.to_csv('hyp.csv')
```

In [4]:

```
dfm=pd.read_csv('hyp.csv')
```

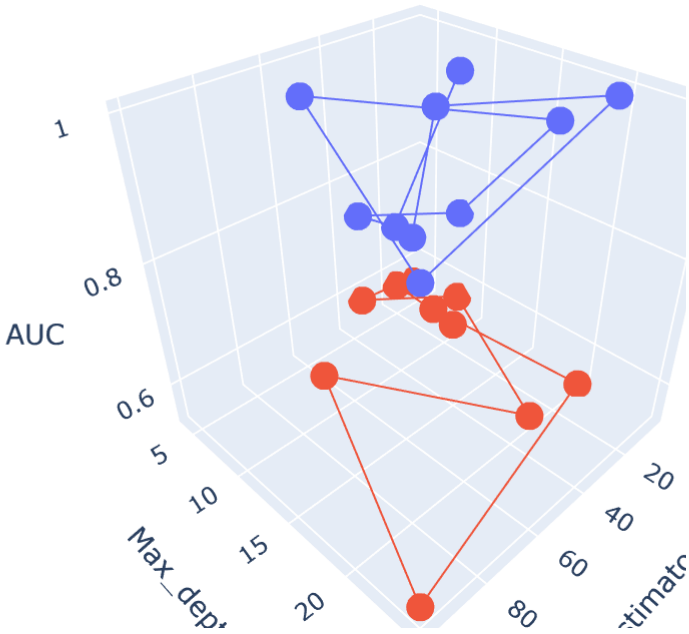
7.3 3D-Plot

In [6]:

```
%matplotlib inline
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np

def enable_plotly_in_cell():
    import IPython
    from plotly.offline import init_notebook_mode
    display(IPython.core.display.HTML('''<script src="/static/components/requirejs/requ
ire.js"></script>'''))
    init_notebook_mode(connected=False)
```

In [7]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=dfm['param_n_estimators'],y=dfm['param_max_depth'],z=dfm['mean_
train_score'], name = 'train')
trace2 = go.Scatter3d(x=dfm['param_n_estimators'],y=dfm['param_max_depth'],z=dfm['mean_
test_score'], name = 'Cross validation')
data = [trace1, trace2]
enable_plotly_in_cell()

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

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



7.4 Best Hyperparameters

In [76]:

```
print(model.best_estimator_)
print('Score on train data :', {model.score(X_tr_avgw2v,y_train)})
print('Mean cross-validated score of the best_estimator :', {model.best_score_})
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=4,
              min_child_weight=1, missing=None, n_estimators=32, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
Score on train data : {0.7118564810534882}
Mean cross-validated score of the best_estimator : {0.575659729218284}
```

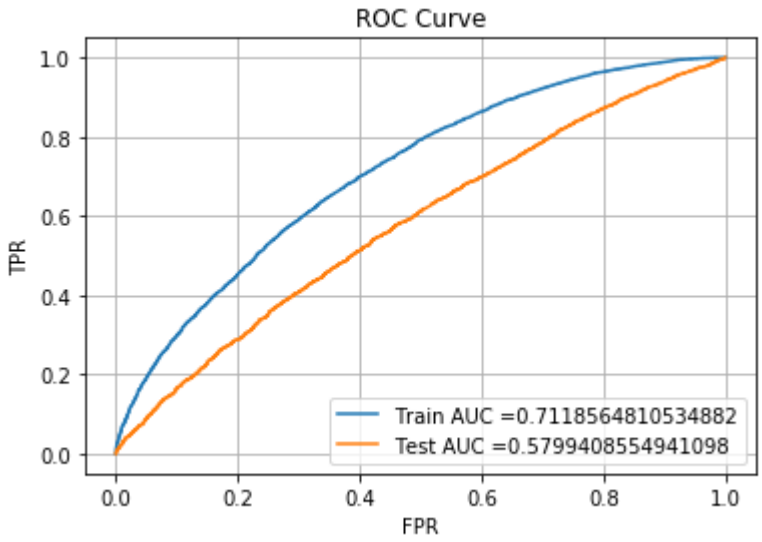
In [ ]:

```
best_parameters = {'n_estimators': [32],'max_depth': [4]}
```

7.5 Applying Best Hyperparameters on train & test data & plotting ROC curve

In [77]:

```
xg_best= XGBClassifier(n_estimators= 32 , max_depth= 4)

xg_best.fit(X_tr_avgw2v, y_train)

y_train_pred_avg_best,pred_labels_train = batch_predict(xg_best, X_tr_avgw2v)
y_test_pred_avg_best,pred_labels_test = batch_predict(xg_best, X_test_avgw2v)

train_tpr_avg, train_fpr_avg, tr_thresholds_avg = roc_curve(y_train, y_train_pred_avg_b
est)
test_tpr_avg, test_fpr_avg, te_thresholds_avg = roc_curve(y_test, y_test_pred_avg_best)

plt.plot(train_tpr_avg, train_fpr_avg,label="Train AUC ="+str(auc(train_tpr_avg, train_
fpr_avg)))
plt.plot(test_tpr_avg, test_fpr_avg, label="Test AUC ="+str(auc(test_tpr_avg, test_fpr_
avg)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.grid()
plt.show()
```



7.6 Plot confusion matrix

In [78]:

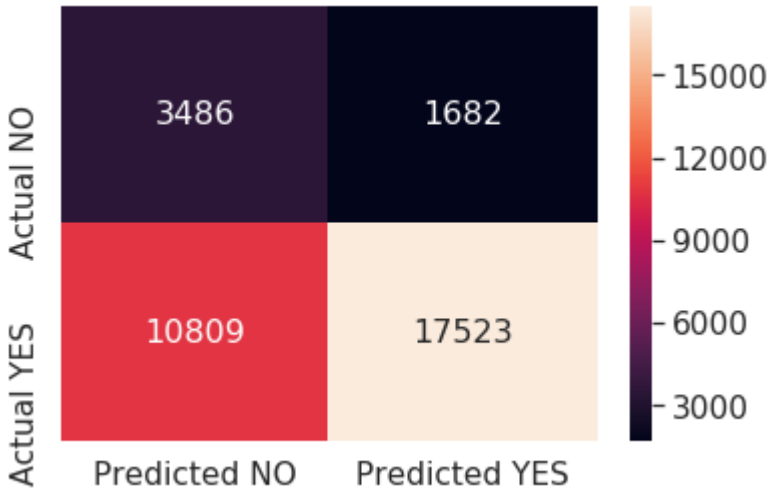
```
from sklearn.metrics import confusion_matrix
best_t_avg = find_best_threshold(tr_thresholds_avg, train_fpr_avg, train_tpr_avg)
print("Train confusion matrix")
cm_train_avg=confusion_matrix(y_train, predict_with_best_t(y_train_pred_avg_best, best_
t_avg))
print(cm_train_avg)
print("Test confusion matrix")
cm_test_avg=confusion_matrix(y_test, predict_with_best_t(y_test_pred_avg_best, best_t_a
vg))
print(cm_test_avg)
```

```
The maximum value of tpr*(1-fpr) 0.12416859591425683 for threshold 0.832
Train confusion matrix
[[ 3486  1682]
 [10809 17523]]
Test confusion matrix
[[1369 1177]
 [5873 8081]]
```

In [79]:

```
# confusion matrix heatmap for train data
print("Train confusion matrix heatmap")
cm_heatmap(cm_train_avg)
```

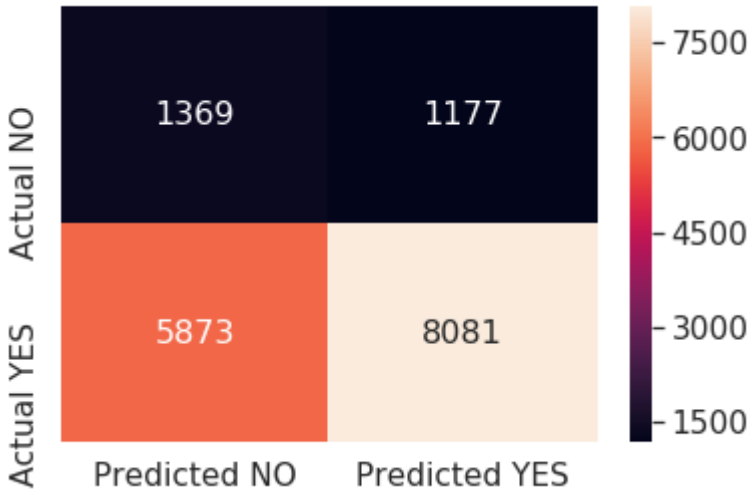
```
Train confusion matrix heatmap
```



In [80]:

```
# confusion matrix heatmap for test data
print("Test confusion matrix heatmap")
cm_heatmap(cm_test_avg)
```

```
Test confusion matrix heatmap
```



8.0 Observations

In [8]:

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

x = PrettyTable()
x.field_names = ["Vectorizer","Model","max_depth","n_estimators" ,"Train AUC","Test AU
C"]
x.add_row(["Custom W2V","XGBoost", 4, 32, 0.71,0.58])
print(x)
```

```
+------------+---------+-----------+--------------+-----------+----------+
| Vectorizer |  Model  | max_depth | n_estimators | Train AUC | Test AUC |
+------------+---------+-----------+--------------+-----------+----------+
| Custom W2V | XGBoost |     4     |      32      |    0.71   |   0.58   |
+------------+---------+-----------+--------------+-----------+----------+
```

Adding on to the above observations:

- From the elbow graph it could be observed that 98% of the variance was explained with 400 features of the original 2500 features