Importing libraries

In [1]:

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

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

warnings.filterwarnings("ignore")
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

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

from matplotlib import pyplot
from scipy.sparse import coo_matrix
from scipy.sparse import csr_matrix
from scipy.sparse import hstack
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
import itertools
```

1. Importing Data

In [2]:

```
X = pd.read_csv('DonorsChoose_processed.csv')
project_data = pd.read_csv('train_data.csv')
project_data=project_data[0:50000]
y = project_data['project_is_approved'].values
```

In [3]:

In [4]:

```
X.head(1)
```

Out[4]:

	teacher_prefix	school_state	project_grade_category	teacher_number_of_previously_posted_projects	price	quantity	c
0	Mrs.	IN	Grades PreK-2	0	154.6	23	s e l v e s

In [6]:

```
vectorizer = TfidfVectorizer(max_features=2000)
X_title_essay = vectorizer.fit_transform(X["Title and Essay"])
print(X_title_essay.shape)
```

```
(50000, 2000)
```

In [14]:

```
vocab=[]
for k in vectorizer.vocabulary_:
    vocab.append(k)
```

In [88]:

```
from pandas import DataFrame
matrix = DataFrame ( columns = vocab)
matrix
```

Out[88]:

	students	english	learners	working	second	third	languages	native	bringing	gift	...	flow	osmo	involvement	readily

0 rows × 2000 columns

In [89]:

```
for i in range(2000):
    matrix.loc[i]=-1
```

In [91]:

```
matrix.shape
```

Out[91]:

```
(2000, 2000)
```

In [214]:

```
text_corpus=[]
for i in range(50000):
    text_corpus.append(X["Title and Essay"][i])
```

```
print(len(text_corpus))
```

```
50000
```

Function to compute Co-Occurance matrix

In [7]:

```
def co_occur(main_word,find_word,window):

    # This function computes the co occurance matrix

    cnt=0
    if(main_word==find_word):
        return cnt
    else:
        for word in text_corpus:
            string=word.split(" ")
            for i,j in enumerate(string):
                if(j==main_word):
                    a=max(i-window,0)
                    b=min(i+window,(len(string)-1))
                    for k in range(a,b+1):
                        if(string[k]==find_word):
                            cnt=cnt+1
        return cnt
```

In [96]:

```
for i in range(len(vocab)):
    m=vocab[i]
    for j in range(len(vocab)):
        if(matrix.at[i,vocab[j]]==-1):
            value=co_occur(m,vocab[j],5)
            matrix.at[i,vocab[j]]=value
            matrix.at[j,vocab[i]]=value
```

In [19]:

```
for i in range(len(vocab)):
    matrix.rename(index={i:vocab[i]}, inplace=True)
```

Co-Occurance Matrix

In [128]:

```
matrix.iloc[0:10,0:10]
```

Out[128]:

	students	english	learners	working	language	school	every	level	also	families
students	0	1671	1997	1688	1759	12924	2133	1493	2872	1342
english	1671	0	854	35	1482	388	45	53	109	94
learners	1997	854	0	63	870	491	113	71	119	64
working	1688	35	63	0	55	414	73	55	97	107
language	1759	1482	870	55	0	387	64	37	147	73
school	12924	388	491	414	387	0	1028	284	599	656
every	2133	45	113	73	64	1028	0	69	78	55
level	1493	53	71	55	37	284	69	0	115	33

	students	english	learners	working	language	school	every	level	also	families
families	1342	94	64	107	73	656	55	33	109	0

Applying TruncatedSVD on the computed Co-Occurance Matrix

In [31]:

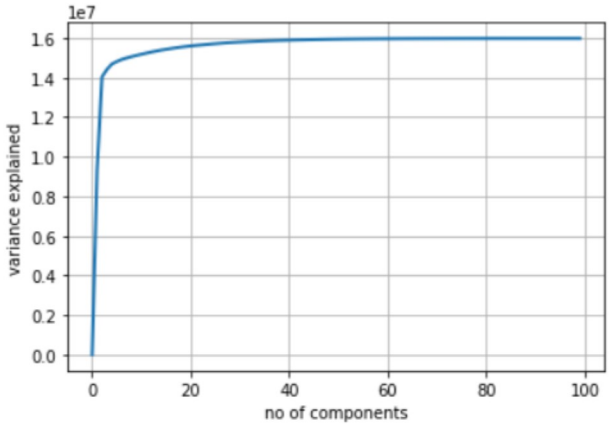
```
from sklearn.decomposition import TruncatedSVD
```

In [90]:

```
variance=[]
for i in range(100):
    svd = TruncatedSVD(n_components=i, n_iter=7, random_state=0)
    svd.fit(matrix)
    variance.append(np.sum(svd.explained_variance_))
```

In [92]:

```
plt.plot(variance,linewidth=2)
plt.grid()
plt.xlabel('no of components')
plt.ylabel('variance explained')
plt.show()
```



From the above graph we can see that as maximum variance can be explained by no of componets = 40, we will reduce the dimension of our Co-Occurance matrix to 40

In [132]:

```
svd = TruncatedSVD(n_components=40, n_iter=7, random_state=0)
matrix_2=svd.fit_transform(matrix)
```

In [133]:

```
matrix_2=pd.DataFrame(matrix_2)
for i in range(len(vocab)):
    matrix_2.rename(index={i:vocab[i]}, columns={i:vocab[i]},inplace=True)
```

Co-Occurance Matrix after applying TruncatedSVD

In [134]:

```
matrix_2.iloc[0:10,0:10]
```

Out[134]:

	students	english	learners	working	language	school	every	level	
students	25787.239263	18534.351780	225.769217	331.588303	137.286869	-111.619542	-64.357323	-23.758546	-20.815
english	1813.603272	-730.971943	375.464049	-0.781375	-182.293901	-331.293410	-79.251223	29.770020	-3.6347
learners	2360.764355	-759.269161	179.564743	-61.214326	123.015118	-78.886176	-144.679285	-14.750176	-20.996
working	1839.213618	-763.683038	46.493752	-105.247867	-7.770798	38.638353	37.740415	-27.911300	20.2323
language	1995.083552	-716.673495	232.358676	-8.750708	-150.041801	-338.373296	-64.709954	34.164725	4.26917
school	13403.384500	-6646.561620	2441.030032	4075.999995	108.331022	-48.837817	73.632018	-171.324518	-100.691
every	3064.639233	-521.131448	565.956772	75.033001	197.981025	931.994804	365.939207	1.010295	-271.840
level	1723.549490	-592.931586	-171.766397	13.053747	-139.246909	-516.115363	546.003332	56.028320	5.64173
also	3486.239773	-1033.053489	-476.813493	-44.799348	-139.701861	-350.624928	-53.387022	134.425553	61.7953
families	1627.105177	-458.708698	644.532163	-56.254463	71.130594	-255.086948	-72.568382	66.539023	-18.663

In [136]:

```
def vec(w):

    # this function returns the word vector corresponding to the word

    return matrix_2.loc[w].as_matrix()
```

In [162]:

```
# creating a set of vocabulary

vocab_set=set(vocab)
```

In [165]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_essay = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in X['clean_essay'].values: # for each review/sentence
    vector = np.zeros(40) # 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 vocab_set:
            vector += vec(word)
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_essay.append(vector)
```

In [167]:

```
avg_w2v_vectors_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in X['clean_title'].values: # for each review/sentence
    vector = np.zeros(40) # 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 vocab_set:
            vector += vec(word)
            cnt_words += 1
    if cnt_words != 0:
```

In [213]:

```
avg_w2v_vectors_title=csr_matrix(avg_w2v_vectors_title)
avg_w2v_vectors_essay=csr_matrix(avg_w2v_vectors_essay)
```

In [174]:

```
#splitting avg_w2v_vectors_title into train and test
from sklearn.model_selection import train_test_split
X_train_avg_t, X_test_avg_t, y_train, y_test = train_test_split(avg_w2v_vectors_title, y, test_size
=0.33, stratify=y,random_state=0)
X_train_avg_title, X_cv_avg_title, y_train, y_cv = train_test_split(X_train_avg_t, y_train, test_si
ze=0.33, stratify=y_train, random_state=0)
```

In [175]:

```
#splitting avg_w2v_vectors_essay into train and test
X_train_avg_e, X_test_avg_e, y_train, y_test = train_test_split(avg_w2v_vectors_essay, y, test_size
=0.33, stratify=y, random_state=0)
X_train_avg_essay, X_cv_avg_essay, y_train, y_cv = train_test_split(X_train_avg_e, y_train, test_si
ze=0.33, stratify=y_train, random_state=0)
```

In [191]:

```
#counting no of words in clean title

sub_title = list(X['clean_title'].values)
j=0
X['Title_Count']=0
for i in sub_title :
    count=1;
    for k in i:
        if(k==' '):
            count=count+1;
    X.loc[j,'Title_Count']=count
    j=j+1
```

In [192]:

```
#counting no of words in clean essay

sub_essay = list(X['clean_essay'].values)
j=0
X['Essay_Count']=0
for i in sub_essay :
    count=1;
    for k in i:
        if(k==' '):
            count=count+1;
    X.loc[j,'Essay_Count']=count
    j=j+1
```

In [193]:

```
#sentiment analysis python -- https://programminghistorian.org/en/lessons/sentiment-analysis
from nltk.sentiment.vader import SentimentIntensityAnalyzer
SIA = SentimentIntensityAnalyzer()
j=0
for i in sub_essay :
    scores = SIA.polarity_scores(i)
    X.loc[j,'Sentiment_Score']=scores['compound']
    j=j+1
```

In [194]:

```
X.tail(1)
```

	teacher_prefix	school_state	project_grade_category	teacher_number_of_previously_posted_projects	price	quar
49999	Mrs.	KY	Grades PreK-2	0	505.43	41

2. Splitting Data

In [195]:

```
# 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, random_state=
0)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train,
random_state=0)

print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
```

```
(22445, 15) (22445,)
(11055, 15) (11055,)
(16500, 15) (16500,)
```

In [196]:

```
X_train=X_train.reset_index()
X_cv=X_cv.reset_index()
X_test=X_test.reset_index()
```

3. Vectorizing data

3.2 one hot encoding the catogorical features: state

In [178]:

```
vec_state = CountVectorizer(min_df=10)
vec_state.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_ohe = vec_state.transform(X_train['school_state'].values)
X_cv_state_ohe = vec_state.transform(X_cv['school_state'].values)
X_test_state_ohe = vec_state.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vec_state.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 51) (22445,)
(11055, 51) (11055,)
(16500, 51) (16500,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm',
'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv
', 'wy']
```

3.3 one hot encoding the catogorical features: teacher_prefix

In [179]:

```
vec_tpre = CountVectorizer(min_df=10)
vec_tpre.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vec_tpre.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = vec_tpre.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vec_tpre.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vec_tpre.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 4) (22445,)
(11055, 4) (11055,)
(16500, 4) (16500,)
['mr', 'mrs', 'ms', 'teacher']
====================================================================================================
```

3.4 one hot encoding the catogorical features: project_grade

In [180]:

```
vec_grade = CountVectorizer(min_df=10)
vec_grade.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_ohe = vec_grade.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vec_grade.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vec_grade.transform(X_test['project_grade_category'].values)

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vec_grade.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 3) (22445,)
(11055, 3) (11055,)
(16500, 3) (16500,)
['12', 'grades', 'prek']
====================================================================================================
```

3.6 one hot encoding the catogorical features: project_category

In [182]:

```
vec_cate = CountVectorizer(min_df=10)
vec_cate.fit(X_train['clean_categories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_category_ohe = vec_cate.transform(X_train['clean_categories'].values)
X_cv_category_ohe = vec_cate.transform(X_cv['clean_categories'].values)
X_test_category_ohe = vec_cate.transform(X_test['clean_title'].values)
```

```
print(X_train_category_ohe.shape, y_train.shape)
print(X_cv_category_ohe.shape, y_cv.shape)
print(X_test_category_ohe.shape, y_test.shape)
print(vec_cate.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 9) (22445,)
(11055, 9) (11055,)
(16500, 9) (16500,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
====================================================================================================
```

3.7 one hot encoding the catogorical features: project_subcategory

In [183]:

```
vec_scate = CountVectorizer(min_df=10)
vec_scate.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_subcategory_ohe = vec_scate.transform(X_train['clean_subcategories'].values)
X_cv_subcategory_ohe = vec_scate.transform(X_cv['clean_subcategories'].values)
X_test_subcategory_ohe = vec_scate.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_subcategory_ohe.shape, y_train.shape)
print(X_cv_subcategory_ohe.shape, y_cv.shape)
print(X_test_subcategory_ohe.shape, y_test.shape)
print(vec_scate.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 30) (22445,)
(11055, 30) (11055,)
(16500, 30) (16500,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
====================================================================================================
```

3.8 Normalizing the numerical features: Price

In [184]:

```
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['price'].values.reshape(1,-1))

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(1,-1)).T
X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1)).T
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1)).T

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_cv_price_norm.shape, y_cv.shape)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
==================================================================================================
```

3.9 Normalizing the numerical features: teacher no of previously posted projects

In [186]:

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

X_train_post_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].v
alues.reshape(1,-1)).T
X_cv_post_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.
reshape(1,-1)).T
X_test_post_norm =
normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-1)).T

print("After vectorizations")
print(X_train_post_norm.shape, y_train.shape)
print(X_cv_post_norm.shape, y_cv.shape)
print(X_test_post_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
==================================================================================================
```

3.10 Normalizing the numerical features: resource summary

In [188]:

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

X_train_digit_norm = normalizer.transform(X_train['Is_digit_present'].values.reshape(1,-1)).T
X_cv_digit_norm = normalizer.transform(X_cv['Is_digit_present'].values.reshape(1,-1)).T
X_test_digit_norm = normalizer.transform(X_test['Is_digit_present'].values.reshape(1,-1)).T

print("After vectorizations")
print(X_train_digit_norm.shape, y_train.shape)
print(X_cv_digit_norm.shape, y_cv.shape)
print(X_test_digit_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
==================================================================================================
```

3.11 Normalizing the numerical features: quantity

In [189]:

```
normalizer.fit(X_train['quantity'].values.reshape(1,-1))

X_train_quantity_norm = normalizer.transform(X_train['quantity'].values.reshape(1,-1)).T
X_cv_quantity_norm = normalizer.transform(X_cv['quantity'].values.reshape(1,-1)).T
X_test_quantity_norm = normalizer.transform(X_test['quantity'].values.reshape(1,-1)).T

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

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Normalizing the numerical features: Title_count

In [197]:

```
normalizer = Normalizer()
normalizer.fit(X_train['Title_Count'].values.reshape(1,-1))

X_train_tcount_norm = normalizer.transform(X_train['Title_Count'].values.reshape(1,-1)).T
X_cv_tcount_norm = normalizer.transform(X_cv['Title_Count'].values.reshape(1,-1)).T
X_test_tcount_norm = normalizer.transform(X_test['Title_Count'].values.reshape(1,-1)).T

print("After vectorizations")
print(X_train_tcount_norm.shape, y_train.shape)
print(X_cv_tcount_norm.shape, y_cv.shape)
print(X_test_tcount_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Normalizing the numerical features: Essay_count

In [198]:

```
normalizer = Normalizer()
normalizer.fit(X_train['Essay_Count'].values.reshape(1,-1))

X_train_ecount_norm = normalizer.transform(X_train['Essay_Count'].values.reshape(1,-1)).T
X_cv_ecount_norm = normalizer.transform(X_cv['Essay_Count'].values.reshape(1,-1)).T
X_test_ecount_norm = normalizer.transform(X_test['Essay_Count'].values.reshape(1,-1)).T

print("After vectorizations")
print(X_train_ecount_norm.shape, y_train.shape)
print(X_cv_ecount_norm.shape, y_cv.shape)
print(X_test_ecount_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Normalizing the numerical features: Sentiment score

In [199]:

```
normalizer = Normalizer()
normalizer.fit(X_train['Sentiment_Score'].values.reshape(1,-1))

X_train_sent_norm = normalizer.transform(X_train['Sentiment_Score'].values.reshape(1,-1)).T
X_cv_sent_norm = normalizer.transform(X_cv['Sentiment_Score'].values.reshape(1,-1)).T
X_test_sent_norm = normalizer.transform(X_test['Sentiment_Score'].values.reshape(1,-1)).T

print("After vectorizations")
print(X_train_sent_norm.shape, y_train.shape)
print(X_cv_sent_norm.shape, y_cv.shape)
print(X_test_sent_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Concatenation of all the vectorized data

In [200]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039

X_tr =
hstack((X_train_tcount_norm,X_train_ecount_norm,X_train_sent_norm,X_train_avg_title,X_train_avg_ess
ay,X_train_state_ohe,X_train_teacher_ohe,X_train_grade_ohe,X_train_category_ohe,X_train_subcategory
_ohe,X_train_price_norm,X_train_post_norm,X_train_digit_norm,X_train_quantity_norm))
X_cr = hstack((X_cv_tcount_norm,X_cv_ecount_norm,X_cv_sent_norm,X_cv_avg_title,X_cv_avg_essay,X_cv_
state_ohe,X_cv_teacher_ohe,X_cv_grade_ohe,X_cv_category_ohe,X_cv_subcategory_ohe,X_cv_price_norm,X
_cv_post_norm,X_cv_digit_norm,X_cv_quantity_norm))
X_te = hstack((X_test_tcount_norm,X_test_ecount_norm,X_test_sent_norm,X_test_avg_t,X_test_avg_e,X_
test_state_ohe,X_test_teacher_ohe,X_test_grade_ohe,X_test_category_ohe,X_test_subcategory_ohe,X_tes
t_price_norm,X_test_post_norm,X_test_digit_norm,X_test_quantity_norm))

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 184) (22445,)
(11055, 184) (11055,)
(16500, 184) (16500,)
====================================================================================================
```

Applying XGBoost on Custom Word Embedding

In [201]:

```
import xgboost as xgb
```

In [202]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

XG = xgb.XGBClassifier(subsample=0.7, colsample_bytree=0.7, random_state=0,class_weight='balanced',
reg_alpha=1,reg_lambda=0)
```

```
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [204]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=parameters['max_depth'],y=parameters['n_estimators'],z=train_auc, name = 't
rain')
trace2 = go.Scatter3d(x=parameters['max_depth'],y=parameters['n_estimators'],z=cv_auc, name = 'Cros
s validation')
data = [trace1, trace2]

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

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

In [203]:

```
clf.best_params_
```

Out[203]:

```
{'max_depth': 3, 'n_estimators': 100}
```

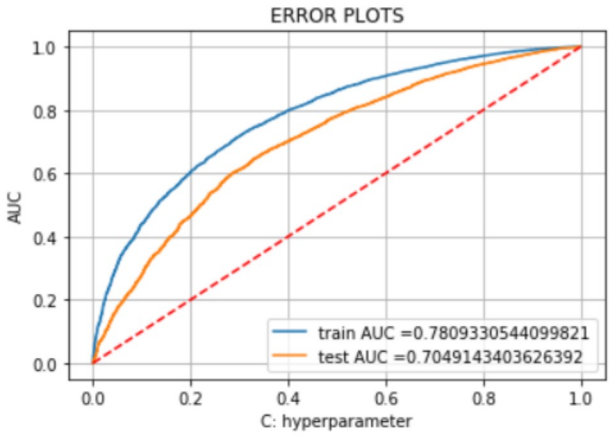
In [205]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
```

```
XG.fit(X_tr, y_train)

y_train_pred = XG.predict_proba(X_tr)
preds = y_train_pred[:,1]
y_test_pred = XG.predict_proba(X_te)
preds2=y_test_pred[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, preds)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, preds2)
```

In [207]:

```
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.plot([0, 1], [0, 1],'r--') #code copied from -https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [208]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [209]:

```
train_conf=confusion_matrix(y_train, predict(preds, tr_thresholds, train_fpr, train_tpr))
test_conf=confusion_matrix(y_test, predict(preds2, te_thresholds, test_fpr, test_tpr))
```

```
the maximum value of tpr*(1-fpr) 0.5030993844252589 for threshold 0.835
the maximum value of tpr*(1-fpr) 0.43259831062020526 for threshold 0.854
```

In [210]:

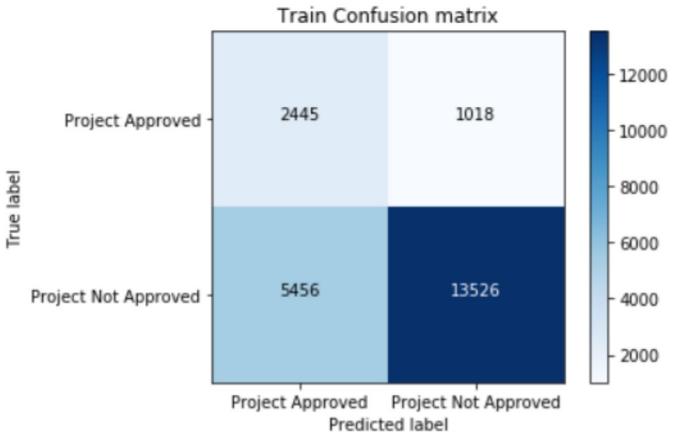
```
#ans heatmap confusion matrix - https://scikit
```

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
class_names=['Project Approved','Project Not Approved']
```

In [211]:

```
plot_confusion_matrix(train_conf, classes=class_names,
                      title='Train Confusion matrix')
```

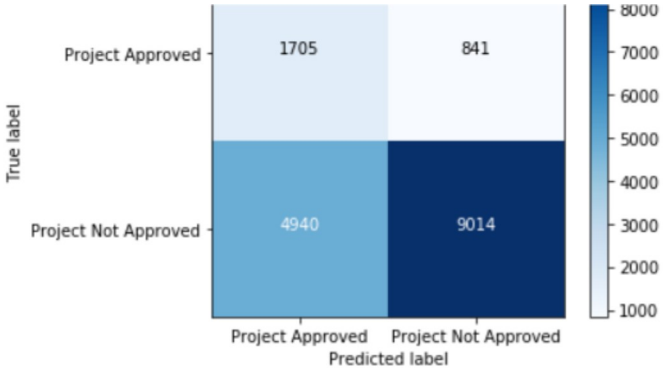
```
Confusion matrix, without normalization
[[ 2445  1018]
 [ 5456 13526]]
```



In [212]:

```
plot_confusion_matrix(test_conf, classes=class_names,
                      title='Test Confusion matrix')
```

```
Confusion matrix, without normalization
[[1705  841]
 [4940 9014]]
```



Conclusion

On applying XGBoost on custom word embedding on DonorsChoose dataset we got the AUC value of 0.70 on test data

In [1]:

```
#code copied from -http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Max Depth", "No of estimators", "AUC"]
x.add_row(["XGBoost",  3, 100,0.70])
print(x)
```

```
+------------+-----------+------------------+-----+
| Vectorizer | Max Depth | No of estimators | AUC |
+------------+-----------+------------------+-----+
|  XGBoost   |     3     |       100        | 0.7 |
+------------+-----------+------------------+-----+
```