Importing libraries

In [1]:

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

from sklearn.ensemble import RandomForestClassifier
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 wordcloud import WordCloud
from prettytable import PrettyTable
from sklearn import decomposition
from sklearn.decomposition import TruncatedSVD
from sklearn.metrics import roc_curve, auc
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.preprocessing import Normalizer
import itertools
from sklearn import tree
import pydotplus
import graphviz
from numpy import sort
import scipy
```

1. Importing Data

```
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
```

2. Splitting Data

In [3]:

```
# 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, 11) (22445,)
(11055, 11) (11055,)
(16500, 11) (16500,)
```

In [4]:

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

3. Vectorizing data

3.1 BOW Vectorization of Clean_essay

In [5]:

```
from sklearn.feature_extraction.text import CountVectorizer
vec_essay = CountVectorizer(min_df=10,ngram_range=(2,2), max_features=5000)
vec_essay.fit(X_train['clean_essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vec_essay.transform(X_train['clean_essay'].values)
X_cv_essay_bow = vec_essay.transform(X_cv['clean_essay'].values)
X_test_essay_bow = vec_essay.transform(X_test['clean_essay'].values)

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
```

```
After vectorizations
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
```

- https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

3.2 response coding- state

In [6]:

```
response_state = pd.DataFrame(columns=['State','Class=0','Class=1'])
response_state
```

Out[6]:

	State	Class=0	Class=1

In [7]:

```
vec_state = CountVectorizer(min_df=10)
vec_state.fit(X_train['school_state'].values)
state_names=vec_state.get_feature_names()
```

In [8]:

```
for i in range(len(state_names)):
    class_0=0
    class_1=0
    response_state.loc[i,'State']=state_names[i].upper()
    for j in range(22445):
        if(X_train.loc[j,'school_state']==state_names[i].upper() and y_train[j]==0):
            class_0+=1
        if(X_train.loc[j,'school_state']==state_names[i].upper() and y_train[j]==1):
            class_1+=1
    response_state.loc[i,'Class=0']=class_0
    response_state.loc[i,'Class=1']=class_1
```

In [9]:

```
response_state.tail()
```

Out[9]:

	State	Class=0	Class=1
46	VT	1	14
47	WA	55	442
48	WI	57	300
49	WV	12	86
50	WY	4	21

In [10]:

```
encode_state = pd.DataFrame(columns=['State=0','State=1'])
encode_state
```

Out[10]:

	State=0	State=1

In [11]:

```
for i in range(22445):
    search=X_train.loc[i,'school_state']
    a=response_state[response_state['State']==search]['Class=1']
    b=response_state[response_state['State']==search]['Class=0']
    total_sum=a+b
    if(total_sum.empty==False):
        encode_state.loc[i,'State=0']=float(b/total_sum)
        encode_state.loc[i,'State=1']=float(a/total_sum)
    else:
        encode_state.loc[i,'State=0']=0.5
        encode_state.loc[i,'State=1']=0.5
```

In [12]:

Out[12]:

	State=0	State=1
22440	0.200913	0.799087
22441	0.152074	0.847926
22442	0.150748	0.849252
22443	0.131356	0.868644
22444	0.198364	0.801636

In [13]:

```
data=encode_state['State=0'].values
data=np.asarray(data,dtype=np.float64)
data=data.reshape(1,-1).T
encode_state_train=csr_matrix(data)
```

In [14]:

```
encode_state_train
```

Out[14]:

```
<22445x1 sparse matrix of type '<class 'numpy.float64'>'
 with 22445 stored elements in Compressed Sparse Row format>
```

In [15]:

```
encode_state = pd.DataFrame(columns=['State=0','State=1'])
for i in range(11055):
    search=X_cv.loc[i,'school_state']
    a=response_state[response_state['State']==search]['Class=1']
    b=response_state[response_state['State']==search]['Class=0']
    total_sum=a+b
    if(total_sum.empty==False):
        encode_state.loc[i,'State=0']=float(b/total_sum)
        encode_state.loc[i,'State=1']=float(a/total_sum)
    else:
        encode_state.loc[i,'State=0']=0.5
        encode_state.loc[i,'State=1']=0.5
```

In [16]:

```
data=encode_state['State=0'].values
data=np.asarray(data,dtype=np.float64)
data=data.reshape(1,-1).T
encode_state_cv=csr_matrix(data)
```

In [17]:

```
encode_state = pd.DataFrame(columns=['State=0','State=1'])
for i in range(16500):
    search=X_test.loc[i,'school_state']
    a=response_state[response_state['State']==search]['Class=1']
    b=response_state[response_state['State']==search]['Class=0']
    total_sum=a+b
    if(total_sum.empty==False):
        encode_state.loc[i,'State=0']=float(b/total_sum)
        encode_state.loc[i,'State=1']=float(a/total_sum)
    else:
        encode_state.loc[i,'State=0']=0.5
        encode_state.loc[i,'State=1']=0.5
```

```
data=np.asarray(data,dtype=np.float64)
data=data.reshape(1,-1).T
encode_state_test=csr_matrix(data)
```

In [19]:

```
print("After vectorizations")
print(encode_state_train.shape, y_train.shape)
print(encode_state_cv.shape, y_cv.shape)
print(encode_state_test.shape, y_test.shape)
print("="*100)
```

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

3.3 response coding- teacher_prefix

In [20]:

```
state_names=['Mr.','Mrs.','Ms.','Teacher']
response_state = pd.DataFrame(columns=['State','Class=0','Class=1'])
for i in range(len(state_names)):
    class_0=0
    class_1=0
    response_state.loc[i,'State']=state_names[i]
    for j in range(22445):
        if(X_train.loc[j,'teacher_prefix']==state_names[i] and y_train[j]==0):
            class_0+=1
        if(X_train.loc[j,'teacher_prefix']==state_names[i] and y_train[j]==1):
            class_1+=1
    response_state.loc[i,'Class=0']=class_0
    response_state.loc[i,'Class=1']=class_1
```

In [21]:

```
response_state
```

Out[21]:

	State	Class=0	Class=1
0	Mr.	336	1819
1	Mrs.	1772	10077
2	Ms.	1249	6689
3	Teacher	106	396

In [22]:

```
encode_state = pd.DataFrame(columns=['State=0','State=1'])
for i in range(22445):
    search=X_train.loc[i,'teacher_prefix']
    a=response_state[response_state['State']==search]['Class=1']
    b=response_state[response_state['State']==search]['Class=0']
    total_sum=a+b
    if(total_sum.empty==False):
        encode_state.loc[i,'State=0']=float(b/total_sum)
        encode_state.loc[i,'State=1']=float(a/total_sum)
    else:
        encode_state.loc[i,'State=0']=0.5
        encode_state.loc[i,'State=1']=0.5
```

In [23]:

```
data=encode_state['State=0'].values
data=np.asarray(data,dtype=np.float64)
data=data.reshape(1,-1).T
encode_prefix_train=csr_matrix(data)
```

In [24]:

```
encode_state = pd.DataFrame(columns=['State=0','State=1'])
for i in range(11055):
    search=X_cv.loc[i,'teacher_prefix']
    a=response_state[response_state['State']==search]['Class=1']
    b=response_state[response_state['State']==search]['Class=0']
    total_sum=a+b
    if(total_sum.empty==False):
        encode_state.loc[i,'State=0']=float(b/total_sum)
        encode_state.loc[i,'State=1']=float(a/total_sum)
    else:
        encode_state.loc[i,'State=0']=0.5
        encode_state.loc[i,'State=1']=0.5
```

In [25]:

```
data=encode_state['State=0'].values
data=np.asarray(data,dtype=np.float64)
data=data.reshape(1,-1).T
encode_prefix_cv=csr_matrix(data)
```

In [26]:

```
encode_state = pd.DataFrame(columns=['State=0','State=1'])
for i in range(16500):
    search=X_test.loc[i,'teacher_prefix']
    a=response_state[response_state['State']==search]['Class=1']
    b=response_state[response_state['State']==search]['Class=0']
    total_sum=a+b
    if(total_sum.empty==False):
        encode_state.loc[i,'State=0']=float(b/total_sum)
        encode_state.loc[i,'State=1']=float(a/total_sum)
    else:
        encode_state.loc[i,'State=0']=0.5
        encode_state.loc[i,'State=1']=0.5
```

In [27]:

```
data=encode_state['State=0'].values
data=np.asarray(data,dtype=np.float64)
data=data.reshape(1,-1).T
encode_prefix_test=csr_matrix(data)
```

In [28]:

```
print("After vectorizations")
print(encode_prefix_train.shape, y_train.shape)
print(encode_prefix_cv.shape, y_cv.shape)
print(encode_prefix_test.shape, y_test.shape)
print("="*100)
```

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

3.4 response coding: project_grade

```
response_state = pd.DataFrame(columns=['State','Class=0','Class=1'])
state_names=['Grades 3-5','Grades 6-8','Grades 9-12','Grades PreK-2']
for i in range(len(state_names)):
    class_0=0
    class_1=0
    response_state.loc[i,'State']=state_names[i]
    for j in range(22445):
        if(X_train.loc[j,'project_grade_category']==state_names[i] and y_train[j]==0):
            class_0+=1
        if(X_train.loc[j,'project_grade_category']==state_names[i] and y_train[j]==1):
            class_1+=1
    response_state.loc[i,'Class=0']=class_0
    response_state.loc[i,'Class=1']=class_1
```

In [30]:

```
response_state
```

Out[30]:

	State	Class=0	Class=1
0	Grades 3-5	1131	6572
1	Grades 6-8	539	2901
2	Grades 9-12	381	1811
3	Grades PreK-2	1412	7698

In [31]:

```
encode_state = pd.DataFrame(columns=['State=0','State=1'])
for i in range(22445):
    search=X_train.loc[i,'project_grade_category']
    a=response_state[response_state['State']==search]['Class=1']
    b=response_state[response_state['State']==search]['Class=0']
    total_sum=a+b
    if(total_sum.empty==False):
        encode_state.loc[i,'State=0']=float(b/total_sum)
        encode_state.loc[i,'State=1']=float(a/total_sum)
    else:
        encode_state.loc[i,'State=0']=0.5
        encode_state.loc[i,'State=1']=0.5
```

In [32]:

```
data=encode_state['State=0'].values
data=np.asarray(data,dtype=np.float64)
data=data.reshape(1,-1).T
encode_grade_train=csr_matrix(data)
```

In [33]:

```
encode_state = pd.DataFrame(columns=['State=0','State=1'])
for i in range(11055):
    search=X_cv.loc[i,'project_grade_category']
    a=response_state[response_state['State']==search]['Class=1']
    b=response_state[response_state['State']==search]['Class=0']
    total_sum=a+b
    if(total_sum.empty==False):
        encode_state.loc[i,'State=0']=float(b/total_sum)
        encode_state.loc[i,'State=1']=float(a/total_sum)
    else:
        encode_state.loc[i,'State=0']=0.5
        encode_state.loc[i,'State=1']=0.5
```

In [34]:

```
data=data.reshape(1,-1).T
encode_grade_cv=csr_matrix(data)
```

In [35]:

```
encode_state = pd.DataFrame(columns=['State=0','State=1'])
for i in range(16500):
    search=X_test.loc[i,'project_grade_category']
    a=response_state[response_state['State']==search]['Class=1']
    b=response_state[response_state['State']==search]['Class=0']
    total_sum=a+b
    if(total_sum.empty==False):
        encode_state.loc[i,'State=0']=float(b/total_sum)
        encode_state.loc[i,'State=1']=float(a/total_sum)
    else:
        encode_state.loc[i,'State=0']=0.5
        encode_state.loc[i,'State=1']=0.5
```

In [36]:

```
data=encode_state['State=0'].values
data=np.asarray(data,dtype=np.float64)
data=data.reshape(1,-1).T
encode_grade_test=csr_matrix(data)
```

In [37]:

```
print("After vectorizations")
print(encode_grade_train.shape, y_train.shape)
print(encode_grade_cv.shape, y_cv.shape)
print(encode_grade_test.shape, y_test.shape)
print("="*100)
```

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

3.5 one hot encoding the catogorical features: project_title

In [38]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_ohe = vec_title.transform(X_train['clean_title'].values)
X_cv_title_ohe = vec_title.transform(X_cv['clean_title'].values)
X_test_title_ohe = vec_title.transform(X_test['clean_title'].values)

print("After vectorizations")
print(X_train_title_ohe.shape, y_train.shape)
print(X_cv_title_ohe.shape, y_cv.shape)
print(X_test_title_ohe.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1143) (22445,)
(11055, 1143) (11055,)
(16500, 1143) (16500,)
====================================================================================================
```

3.6 response coding: project_category

In [39]:

```
response_state = pd.DataFrame(columns=['State','Class=0','Class=1'])
state_names=['Math_Science','Literacy_Language','AppliedLearning','Music_Arts','Health_Sports','Spe
cialNeeds']
for i in range(len(state_names)):
    class_0=0
    class_1=0
    response_state.loc[i,'State']=state_names[i]
    for j in range(22445):
        if(X_train.loc[j,'clean_categories']==state_names[i] and y_train[j]==0):
            class_0+=1
        if(X_train.loc[j,'clean_categories']==state_names[i] and y_train[j]==1):
            class_1+=1
    response_state.loc[i,'Class=0']=class_0
    response_state.loc[i,'Class=1']=class_1
```

In [40]:

```
response_state
```

Out[40]:

	State	Class=0	Class=1
0	Math_Science	615	2800
1	Literacy_Language	700	4243
2	AppliedLearning	144	623
3	Music_Arts	145	914
4	Health_Sports	330	1769
5	SpecialNeeds	163	716

In [41]:

```
encode_state = pd.DataFrame(columns=['State=0','State=1'])
for i in range(22445):
    search=X_train.loc[i,'clean_categories']
    a=response_state[response_state['State']==search]['Class=1']
    b=response_state[response_state['State']==search]['Class=0']
    total_sum=a+b
    if(total_sum.empty==False):
        encode_state.loc[i,'State=0']=float(b/total_sum)
        encode_state.loc[i,'State=1']=float(a/total_sum)
    else:
        encode_state.loc[i,'State=0']=0.5
        encode_state.loc[i,'State=1']=0.5
```

In [42]:

```
data=encode_state['State=0'].values
data=np.asarray(data,dtype=np.float64)
data=data.reshape(1,-1).T
encode_cat_train=csr_matrix(data)
```

In [43]:

```
encode_state = pd.DataFrame(columns=['State=0','State=1'])
for i in range(11055):
    search=X_cv.loc[i,'clean_categories']
    a=response_state[response_state['State']==search]['Class=1']
    b=response_state[response_state['State']==search]['Class=0']
    total_sum=a+b
    if(total_sum.empty==False):
        encode_state.loc[i,'State=0']=float(b/total_sum)
        encode_state.loc[i,'State=1']=float(a/total_sum)
    else:
        encode_state.loc[i,'State=0']=0.5
```

In [44]:

```
data=encode_state['State=0'].values
data=np.asarray(data,dtype=np.float64)
data=data.reshape(1,-1).T
encode_cat_cv=csr_matrix(data)
```

In [45]:

```
encode_state = pd.DataFrame(columns=['State=0','State=1'])
for i in range(16500):
    search=X_test.loc[i,'clean_categories']
    a=response_state[response_state['State']==search]['Class=1']
    b=response_state[response_state['State']==search]['Class=0']
    total_sum=a+b
    if(total_sum.empty==False):
        encode_state.loc[i,'State=0']=float(b/total_sum)
        encode_state.loc[i,'State=1']=float(a/total_sum)
    else:
        encode_state.loc[i,'State=0']=0.5
        encode_state.loc[i,'State=1']=0.5
```

In [46]:

```
data=encode_state['State=0'].values
data=np.asarray(data,dtype=np.float64)
data=data.reshape(1,-1).T
encode_cat_test=csr_matrix(data)
```

In [47]:

```
print("After vectorizations")
print(encode_cat_train.shape, y_train.shape)
print(encode_cat_cv.shape, y_cv.shape)
print(encode_cat_test.shape, y_test.shape)
print("="*100)
```

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

3.7 response coding: project_subcategory

In [48]:

```
response_state = pd.DataFrame(columns=['State','Class=0','Class=1'])
state_names=['Mathematics','Literacy','Health_Wellness','EarlyDevelopment','PerformingArts','Colleg
e_CareerPrep','ESL','SpecialNeeds','AppliedSciences','History_Geography','Literature_Writing','Visu
alArts']
for i in range(len(state_names)):
    class_0=0
    class_1=0
    response_state.loc[i,'State']=state_names[i]
    for j in range(22445):
        if(X_train.loc[j,'clean_subcategories']==state_names[i] and y_train[j]==0):
            class_0+=1
        if(X_train.loc[j,'clean_subcategories']==state_names[i] and y_train[j]==1):
            class_1+=1
    response_state.loc[i,'Class=0']=class_0
    response_state.loc[i,'Class=1']=class_1
```

In [49]:

```
response_state
```

Out[49]:

	State	Class=0	Class=1
0	Mathematics	186	882
1	Literacy	233	1806
2	Health_Wellness	105	646
3	EarlyDevelopment	38	156
4	PerformingArts	12	87
5	College_CareerPrep	15	58
6	ESL	16	69
7	SpecialNeeds	163	716
8	AppliedSciences	92	400
9	History_Geography	23	85
10	Literature_Writing	162	774
11	VisualArts	73	361

In [50]:

```
encode_state = pd.DataFrame(columns=['State=0','State=1'])
for i in range(22445):
    search=X_train.loc[i,'clean_subcategories']
    a=response_state[response_state['State']==search]['Class=1']
    b=response_state[response_state['State']==search]['Class=0']
    total_sum=a+b
    if(total_sum.empty==False):
        encode_state.loc[i,'State=0']=float(b/total_sum)
        encode_state.loc[i,'State=1']=float(a/total_sum)
    else:
        encode_state.loc[i,'State=0']=0.5
        encode_state.loc[i,'State=1']=0.5
```

In [51]:

```
data=encode_state['State=0'].values
data=np.asarray(data,dtype=np.float64)
data=data.reshape(1,-1).T
encode_subcat_train=csr_matrix(data)
```

In [52]:

```
encode_state = pd.DataFrame(columns=['State=0','State=1'])
for i in range(11055):
    search=X_cv.loc[i,'clean_subcategories']
    a=response_state[response_state['State']==search]['Class=1']
    b=response_state[response_state['State']==search]['Class=0']
    total_sum=a+b
    if(total_sum.empty==False):
        encode_state.loc[i,'State=0']=float(b/total_sum)
        encode_state.loc[i,'State=1']=float(a/total_sum)
    else:
        encode_state.loc[i,'State=0']=0.5
        encode_state.loc[i,'State=1']=0.5
```

In [53]:

```
data=encode_state['State=0'].values
data=np.asarray(data,dtype=np.float64)
data=data.reshape(1,-1).T
encode_subcat_cv=csr_matrix(data)
```

```
encode_state = pd.DataFrame(columns=['State=0','State=1'])
for i in range(16500):
    search=X_test.loc[i,'clean_subcategories']
    a=response_state[response_state['State']==search]['Class=1']
    b=response_state[response_state['State']==search]['Class=0']
    total_sum=a+b
    if(total_sum.empty==False):
        encode_state.loc[i,'State=0']=float(b/total_sum)
        encode_state.loc[i,'State=1']=float(a/total_sum)
    else:
        encode_state.loc[i,'State=0']=0.5
        encode_state.loc[i,'State=1']=0.5
```

In [55]:

```
data=encode_state['State=0'].values
data=np.asarray(data,dtype=np.float64)
data=data.reshape(1,-1).T
encode_subcat_test=csr_matrix(data)
```

In [56]:

```
print("After vectorizations")
print(encode_subcat_train.shape, y_train.shape)
print(encode_subcat_cv.shape, y_cv.shape)
print(encode_subcat_test.shape, y_test.shape)
print("="*100)
```

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

3.8 Price

In [57]:

```
X_train_price_norm = X_train['price'].values.reshape(1,-1).T
X_cv_price_norm = X_cv['price'].values.reshape(1,-1).T
X_test_price_norm = 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)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

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

3.9 teacher no of previously posted projects

In [58]:

```
X_train_post_norm = X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1).T
X_cv_post_norm = X_cv['teacher_number_of_previously_posted_projects'].values.reshape(1,-1).T
X_test_post_norm = 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)
```

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

3.10 resource summary

In [59]:

```
X_train_digit_norm = X_train['Is_digit_present'].values.reshape(1,-1).T
X_cv_digit_norm = X_cv['Is_digit_present'].values.reshape(1,-1).T
X_test_digit_norm = 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 quantity

In [60]:

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

Concatenation of all the vectorized data

In [62]:

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

X_tr =
hstack((X_train_essay_bow,X_train_title_ohe,encode_state_train,encode_prefix_train,encode_grade_tra
in,encode_cat_train,encode_subcat_train,X_train_price_norm,X_train_post_norm,X_train_digit_norm,X_
train_quantity_norm))
X_cr =
hstack((X_cv_essay_bow,X_cv_title_ohe,encode_state_cv,encode_prefix_cv,encode_grade_cv,encode_cat_c
v,encode_subcat_cv,X_cv_price_norm,X_cv_post_norm,X_cv_digit_norm,X_cv_quantity_norm))
X_te =
hstack((X_test_essay_bow,X_test_title_ohe,encode_state_test,encode_prefix_test,encode_grade_test,e
```

```
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, 6152) (22445,)
(11055, 6152) (11055,)
(16500, 6152) (16500,)
====================================================================================================
```

While applying Random Forest, we will allow the trees to grow out fully , so values for max depth of trees will be large. It is done to create high variance Base Models.

Applying Random Forest on BOW

In [62]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
```

In [66]:

```
RF = RandomForestClassifier(random_state=0,class_weight='balanced')
RF
```

Out[66]:

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators='warn', n_jobs=None, oob_score=False,
            random_state=0, verbose=0, warm_start=False)
```

In [67]:

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

RF = RandomForestClassifier(random_state=0,class_weight='balanced')


parameters = {'max_depth':[1, 5, 10, 50, 100, 200],'n_estimators':[1, 5, 10, 50, 100, 200]}
clf = GridSearchCV(RF, parameters, cv=3, scoring='roc_auc')
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 [72]:

```
clf.best_params_
```

Out[72]:

```
{'max_depth': 100, 'n_estimators': 200}
```

```
# 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='min_samples_split'),
        zaxis = dict(title='AUC'),))

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

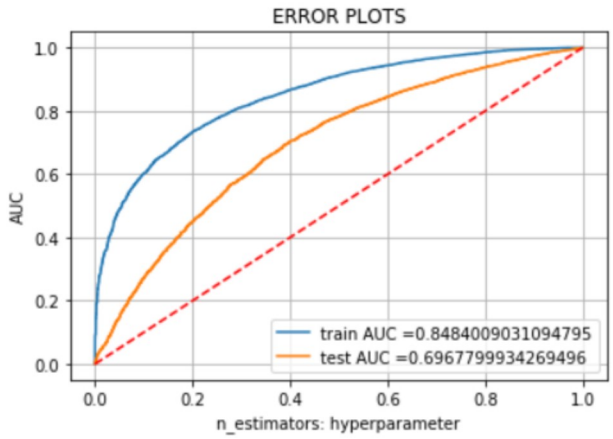
In [67]:

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



RF = RandomForestClassifier(random_state=0,class_weight='balanced',max_depth=10,n_estimators=200)
RF.fit(X_tr, y_train)

y_train_pred = RF.predict_proba(X_tr)
preds = y_train_pred[:,1]
y_test_pred = RF.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 [68]:

```
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.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [69]:

```
# 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 [70]:

```
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) 0.5873586396559769 for threshold 0.502
the maximum value of tpr*(1-fpr) 0.4239855091147313 for threshold 0.503
```

In [71]:

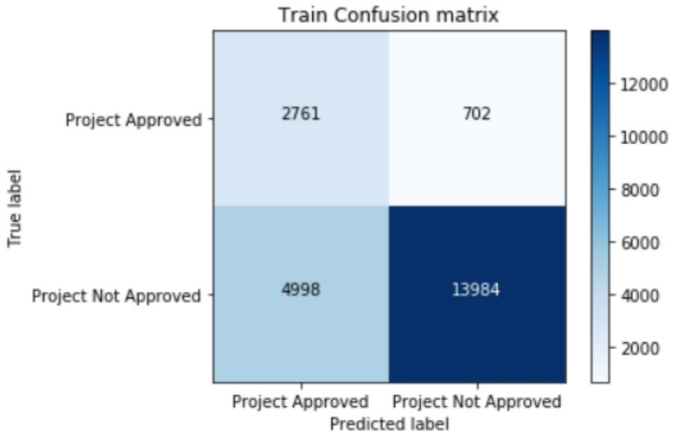
```
#sns heatmap confusion matrix -https://scikit-
learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    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.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 [72]:

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

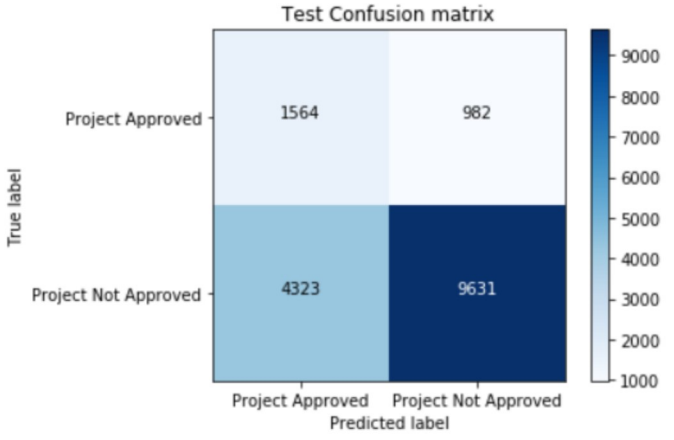
```
Confusion matrix, without normalization
[[ 2761   702]
 [ 4998 13984]]
```



In [73]:

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

```
Confusion matrix, without normalization
[[1564  982]
 [4323 9631]]
```



Conclusion

On applying RandomForest BOW on DonorsChoose dataset with max_depth- 10, n_estimators-200 we got the AUC value of 0.69 on test data

Applying RandomForest on TFIDF

TFIDF Vectorization of Clean_essay

In [74]:

```
vec_essay = TfidfVectorizer(min_df=10, max_features=5000)
vec_essay.fit(X_train['clean_essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf = vec_essay.transform(X_train['clean_essay'].values)
X_cv_essay_tfidf = vec_essay.transform(X_cv['clean_essay'].values)
X_test_essay_tfidf = vec_essay.transform(X_test['clean_essay'].values)

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

```
After vectorizations
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
```

TFIDF Vectorization of Clean_title

In [75]:

```
vec_title = TfidfVectorizer(min_df=10, max_features=5000)
vec_title.fit(X_train['clean_title'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_tfidf = vec_title.transform(X_train['clean_title'].values)
X_cv_title_tfidf = vec_title.transform(X_cv['clean_title'].values)
X_test_title_tfidf = vec_title.transform(X_test['clean_title'].values)

print("After vectorizations")
print(X_train_title_tfidf.shape, y_train.shape)
print(X_cv_title_tfidf.shape, y_cv.shape)
print(X_test_title_tfidf.shape, y_test.shape)
```

```
After vectorizations
(22445, 1143) (22445,)
(11055, 1143) (11055,)
(16500, 1143) (16500,)
```

Concatenation of all the vectorized data

In [76]:

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

X_tr =
hstack((X_train_essay_tfidf,X_train_title_tfidf,encode_state_train,encode_prefix_train,encode_grade
_train,encode_cat_train,encode_subcat_train,X_train_price_norm,X_train_post_norm,X_train_digit_norm
,X_train_quantity_norm))
X_cr = hstack((X_cv_essay_tfidf,X_cv_title_tfidf,encode_state_cv,encode_prefix_cv,encode_grade_cv,
encode_cat_cv,encode_subcat_cv,X_cv_price_norm,X_cv_post_norm,X_cv_digit_norm,X_cv_quantity_norm))
X_te =
hstack((X_test_essay_tfidf,X_test_title_tfidf,encode_state_test,encode_prefix_test,encode_grade_tes
t,encode_cat_test,encode_subcat_test,X_test_price_norm,X_test_post_norm,X_test_digit_norm,X_test_qu
```

```
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, 6152) (22445,)
(11055, 6152) (11055,)
(16500, 6152) (16500,)
====================================================================================================
```

In [86]:

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

RF = RandomForestClassifier(random_state=0,class_weight='balanced')


parameters = {'max_depth':[1, 5, 10, 50, 100, 200],'n_estimators':[1, 5, 10, 50, 100, 200]}
clf = GridSearchCV(RF, parameters, cv=3, scoring='roc_auc')
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 [88]:

```
clf.best_params_
```

Out[88]:

```
{'max_depth': 50, 'n_estimators': 200}
```

In [87]:

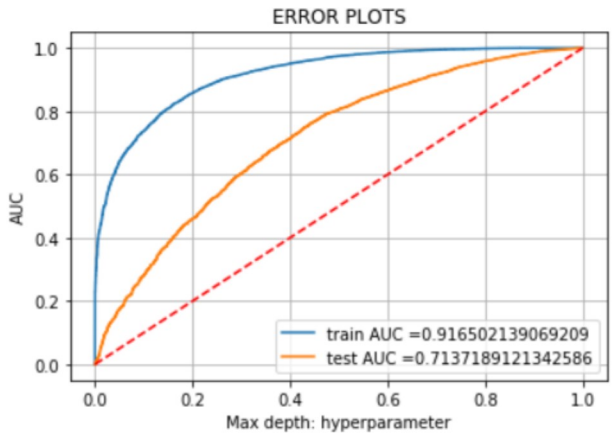
```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=parameters['max_depth'],y=parameters['n_estimators'],z=train_auc, name = 't
rain')
trace2 = go.Scatter3d(x=parameters['max_depth'],y=parameters['n_estimators'],z=cv_auc, name = 'Cros
s validation')
data = [trace1, trace2]

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

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

In [79]:

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



RF = RandomForestClassifier(random_state=0,class_weight='balanced',max_depth=10,n_estimators=200)
RF.fit(X_tr, y_train)

y_train_pred = RF.predict_proba(X_tr)
preds = y_train_pred[:,1]
y_test_pred = RF.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 [80]:

```
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("Max depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



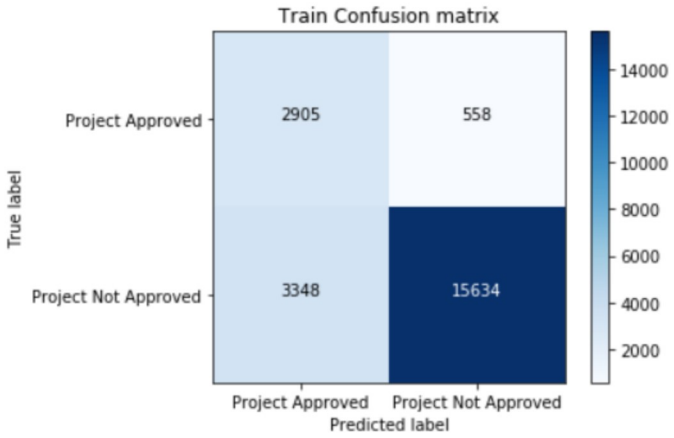
In [81]:

```
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) 0.6909104854963437 for threshold 0.508
the maximum value of tpr*(1-fpr) 0.4329807815399741 for threshold 0.519
```

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

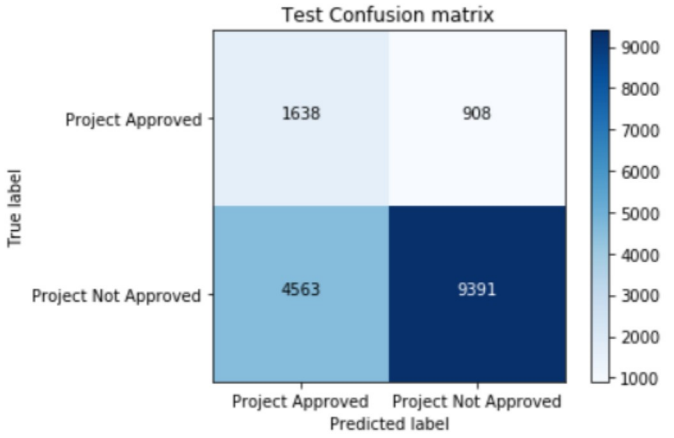
```
Confusion matrix, without normalization
[[ 2905   558]
 [ 3348 15634]]
```



In [83]:

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

```
Confusion matrix, without normalization
[[1638  908]
 [4563 9391]]
```



Conclusion

On applying RandomForest TFIDF on DonorsChoose dataset with max_depth- 10, n_estimators- 200 we got the AUC value of 0.71 on test data

Applying RandomForest on AVG W2V

AVG_W2V Vectorization of Clean_essay

In [84]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
ve-and-load-variables-in-python/
# make sure you have the glove vectors file
```

```
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [85]:

```
# 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(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_essay.append(vector)
```

AVG_W2V Vectorization of Clean_title

In [86]:

```
# Similarly you can vectorize for title also
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(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_title.append(vector)
```

In [87]:

```
avg_w2v_vectors_title=csr_matrix(avg_w2v_vectors_title)
avg_w2v_vectors_essay=csr_matrix(avg_w2v_vectors_essay)
avg_w2v_vectors_title.shape
```

Out[87]:

```
(50000, 300)
```

In [88]:

```
#splitting avg_w2v_vectors_title into train and test
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 [89]:

```
#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)
```

Concatenation of all the vectorized data

In [90]:

```
X_tr =
hstack((X_train_avg_title,X_train_avg_essay,encode_state_train,encode_prefix_train,encode_grade_tra
in,encode_cat_train,encode_subcat_train,X_train_price_norm,X_train_post_norm,X_train_digit_norm,X_
train_quantity_norm))
X_cr =
hstack((X_cv_avg_title,X_cv_avg_essay,encode_state_cv,encode_prefix_cv,encode_grade_cv,encode_cat_c
v,encode_subcat_cv,X_cv_price_norm,X_cv_post_norm,X_cv_digit_norm,X_cv_quantity_norm))
X_te = hstack((X_test_avg_t,X_test_avg_e,encode_state_test,encode_prefix_test,encode_grade_test,en
code_cat_test,encode_subcat_test,X_test_price_norm,X_test_post_norm,X_test_digit_norm,X_test_quanti
ty_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, 609) (22445,)
(11055, 609) (11055,)
(16500, 609) (16500,)
====================================================================================================
```

In [117]:

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

RF = RandomForestClassifier(random_state=0,class_weight='balanced')


parameters = {'max_depth':[1, 5, 10, 50, 100, 200],'n_estimators':[1, 5, 10, 50, 100, 200]}
clf = GridSearchCV(RF, parameters, cv=3, scoring='roc_auc')
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 [120]:

```
clf.best_params_
```

Out[120]:

```
{'max_depth': 5, 'n_estimators': 200}
```

In [122]:

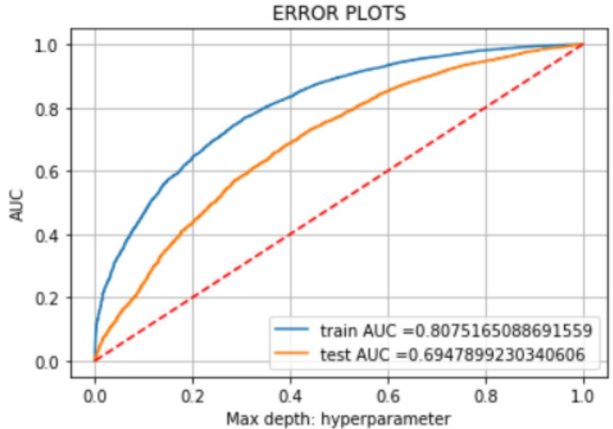
```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=parameters['max_depth'],y=parameters['n_estimators'],z=train_auc, name = 't
rain')
trace2 = go.Scatter3d(x=parameters['max_depth'],y=parameters['n_estimators'],z=cv_auc, name = 'Cros
s validation')
data = [trace1, trace2]

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

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

In [96]:

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



RF = RandomForestClassifier(random_state=0,class_weight='balanced',max_depth=5,n_estimators=200)
RF.fit(X_tr, y_train)

y_train_pred = RF.predict_proba(X_tr)
preds = y_train_pred[:,1]
y_test_pred = RF.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 [97]:

```
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("Max depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [98]:

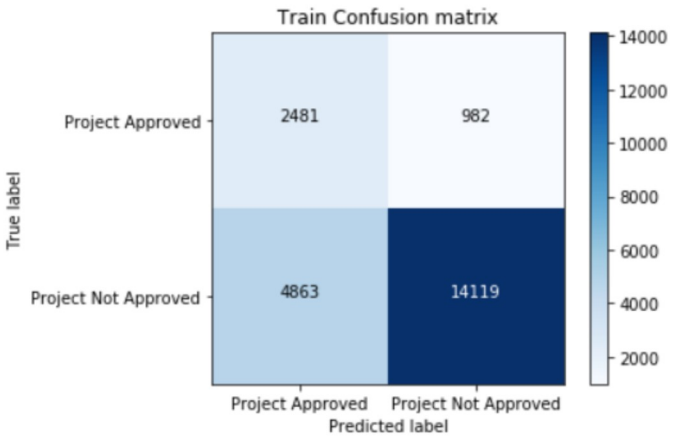
```
train_conf=confusion_matrix(y_train, predict(preds, tr_thresholds, train_fpr, train_tpr))
test_conf=confusion_matrix(y_test, predict(preds2, te_thresholds, test_fpr, test_tpr))
```

```
the maximum value of tpr*(1-fpr) 0.5328883697378184 for threshold 0.505
the maximum value of tpr*(1-fpr) 0.4155796494845988 for threshold 0.516
```

In [99]:

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

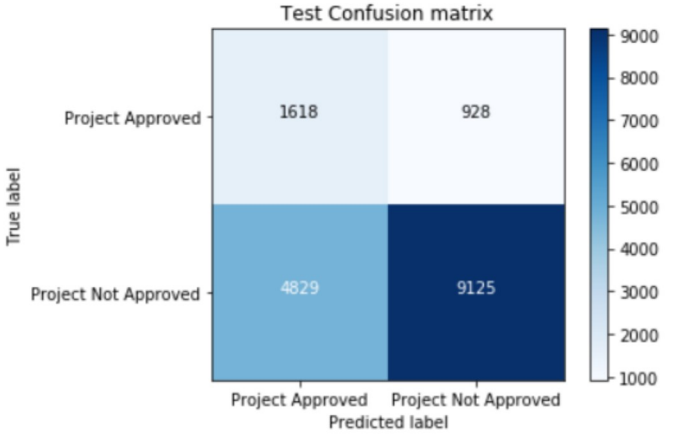
```
Confusion matrix, without normalization
[[ 2481   982]
 [ 4863 14119]]
```



In [100]:

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

```
Confusion matrix, without normalization
[[1618  928]
 [4829 9125]]
```



Conclusion

On applying RandomForest AVG W2V on DonorsChoose dataset with max_depth- 5, n_estimators-200 we got the AUC value of 0.69 on test data

Applying RandomForest on TFIDF W2V

TFIDF_W2V Vectorization of Clean_essay

In [101]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X['clean_essay'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [102]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_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(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_essay.append(vector)
```

TFIDF_W2V Vectorization of Clean_title

In [103]:

```
# Similarly you can vectorize for title also
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X['clean_title'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [104]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_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(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_title.append(vector)
```

In [105]:

```
tfidf_w2v_vectors_essay=csr_matrix(tfidf_w2v_vectors_essay)
tfidf_w2v_vectors_title.shape
```

Out[105]:

```
(50000, 300)
```

In [106]:

```
#splitting tfidf_w2v_vectors_title into train and test
X_train_tfidf_t, X_test_tfidf_t, y_train, y_test = train_test_split(tfidf_w2v_vectors_title, y, tes
t_size=0.33, stratify=y,random_state=0)
X_train_tfidf_title, X_cv_tfidf_title, y_train, y_cv = train_test_split(X_train_tfidf_t, y_train, t
est_size=0.33, stratify=y_train,random_state=0)
```

In [107]:

```
#splitting tfidf_w2v_vectors_title into train and test
X_train_tfidf_e, X_test_tfidf_e, y_train, y_test = train_test_split(tfidf_w2v_vectors_essay, y, tes
t_size=0.33, stratify=y,random_state=0)
X_train_tfidf_essay, X_cv_tfidf_essay, y_train, y_cv = train_test_split(X_train_tfidf_e, y_train, t
est_size=0.33, stratify=y_train,random_state=0)
```

Concatenation of all the vectorized data

In [108]:

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

X_tr =
hstack((X_train_tfidf_title,X_train_tfidf_essay,encode_state_train,encode_prefix_train,encode_grade
_train,encode_cat_train,encode_subcat_train,X_train_price_norm,X_train_post_norm,X_train_digit_norm
,X_train_quantity_norm))
X_cr = hstack((X_cv_tfidf_title,X_cv_tfidf_essay,encode_state_cv,encode_prefix_cv,encode_grade_cv,
encode_cat_cv,encode_subcat_cv,X_cv_price_norm,X_cv_post_norm,X_cv_digit_norm,X_cv_quantity_norm))
X_te =
hstack((X_test_tfidf_t,X_test_tfidf_e,encode_state_test,encode_prefix_test,encode_grade_test,encode
_cat_test,encode_subcat_test,X_test_price_norm,X_test_post_norm,X_test_digit_norm,X_test_quantity_n
orm))

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, 609) (22445,)
(11055, 609) (11055,)
(16500, 609) (16500,)
====================================================================================================
```

In [158]:

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

RF = RandomForestClassifier(random_state=0,class_weight='balanced')


parameters = {'max_depth':[1, 5, 10, 50, 100, 200],'n_estimators':[1, 5, 10, 50, 100, 200]}
clf = GridSearchCV(RF, parameters, cv=3, scoring='roc_auc')
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 [160]:

```
# 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')
```

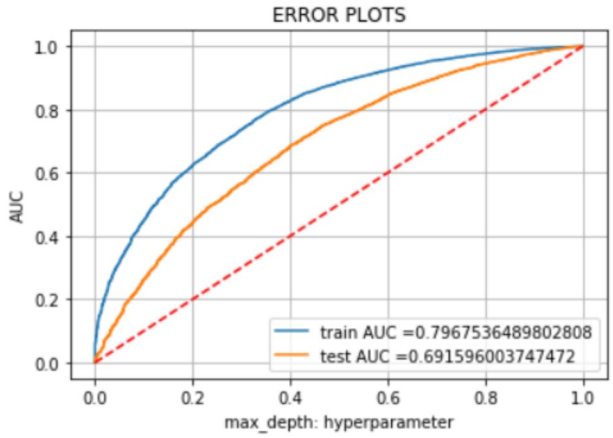
In [109]:

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



RF = RandomForestClassifier(random_state=0,class_weight='balanced',max_depth=5,n_estimators=200)
RF.fit(X_tr, y_train)

y_train_pred = RF.predict_proba(X_tr)
preds = y_train_pred[:,1]
y_test_pred = RF.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 [110]:

```
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.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [111]:

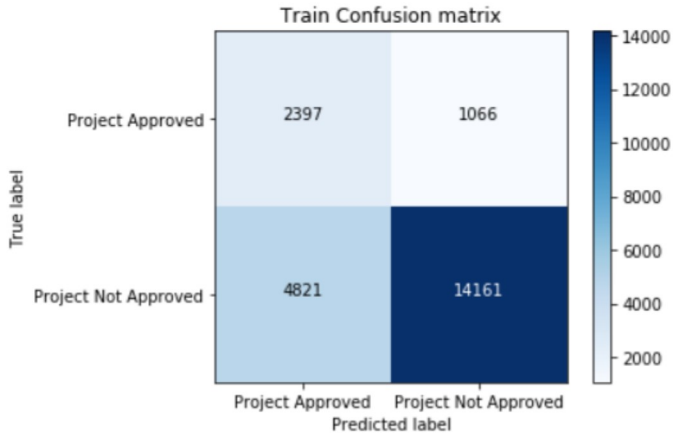
```
train_conf=confusion_matrix(y_train, predict(preds, tr_thresholds, train_fpr, train_tpr))
test_conf=confusion_matrix(y_test, predict(preds2, te_thresholds, test_fpr, test_tpr))
```

```
the maximum value of tpr*(1-fpr) 0.5163777206991513 for threshold 0.496
the maximum value of tpr*(1-fpr) 0.4108451503937131 for threshold 0.503
```

In [112]:

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

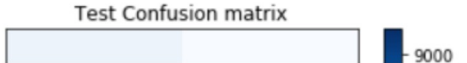
```
Confusion matrix, without normalization
[[ 2397  1066]
 [ 4821 14161]]
```

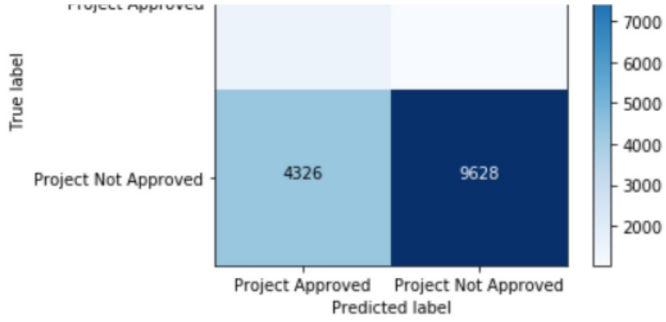


In [113]:

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

```
Confusion matrix, without normalization
[[1516 1030]
 [4326 9628]]
```





Conclusion

On applying RandomForest TFIDF W2V on DonorsChoose dataset with max_depth- 5, n_estimators-200 we got the AUC value of 0.69 on test data

While applying XGBoost we will create trees of shallow depth , so values for max depth of trees will be small. It is done to create high bias Base Models.

XGBoost BOW

In [166]:

```
import xgboost as xgb
```

In [167]:

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

X_tr =
hstack((X_train_essay_bow,X_train_title_ohe,encode_state_train,encode_prefix_train,encode_grade_tra
in,encode_cat_train,encode_subcat_train,X_train_price_norm,X_train_post_norm,X_train_digit_norm,X_
train_quantity_norm))
X_cr =
hstack((X_cv_essay_bow,X_cv_title_ohe,encode_state_cv,encode_prefix_cv,encode_grade_cv,encode_cat_c
v,encode_subcat_cv,X_cv_price_norm,X_cv_post_norm,X_cv_digit_norm,X_cv_quantity_norm))
X_te =
hstack((X_test_essay_bow,X_test_title_ohe,encode_state_test,encode_prefix_test,encode_grade_test,e
ncode_cat_test,encode_subcat_test,X_test_price_norm,X_test_post_norm,X_test_digit_norm,X_test_quant
ity_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, 6152) (22445,)
(11055, 6152) (11055,)
(16500, 6152) (16500,)
====================================================================================================
```

In [182]:

```
# 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 = GridSearchCV(XG, parameters, cv=3, scoring='roc_auc')
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 [185]:

```
clf.best_params_
```

Out[185]:

```
{'max_depth': 3, 'n_estimators': 200}
```

In [186]:

```
# 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')
```

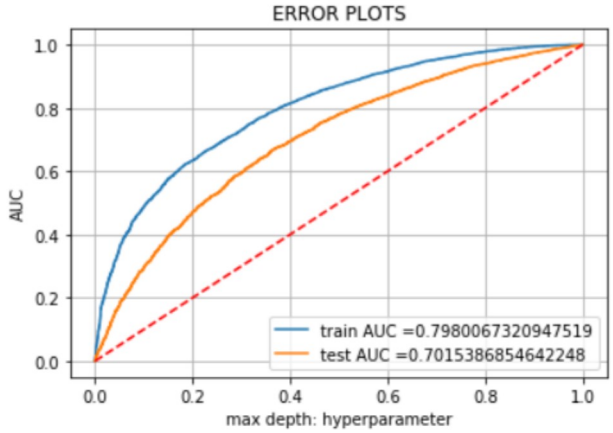
In [188]:

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

```
ate=0,class_weight='balanced', reg_alpha=1,reg_lambda=0)
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 [189]:

```
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("max depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [190]:

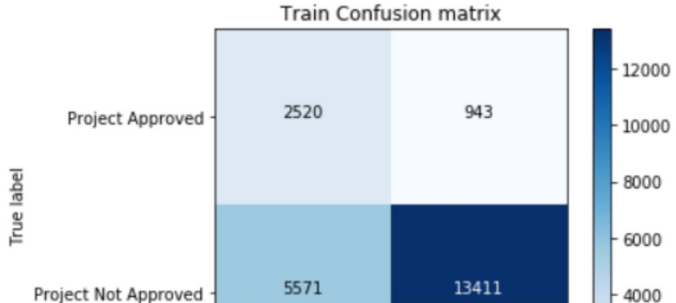
```
train_conf=confusion_matrix(y_train, predict(preds, tr_thresholds, train_fpr, train_tpr))
test_conf=confusion_matrix(y_test, predict(preds2, te_thresholds, test_fpr, test_tpr))
```

```
the maximum value of tpr*(1-fpr) 0.5141232481503748 for threshold 0.832
the maximum value of tpr*(1-fpr) 0.42452605750619726 for threshold 0.839
```

In [191]:

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

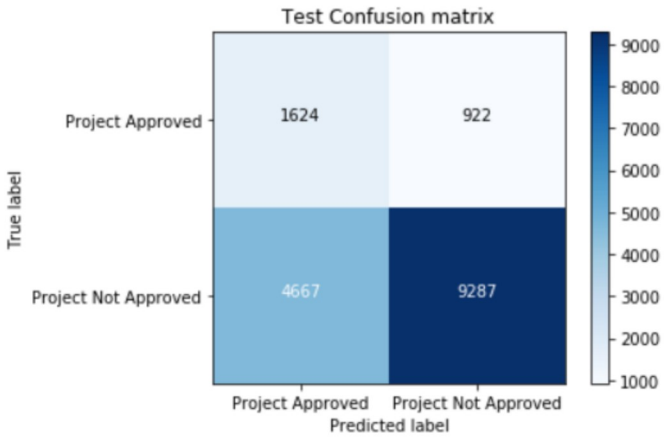
```
Confusion matrix, without normalization
[[ 2520   943]
 [ 5571 13411]]
```



In [192]:

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

```
Confusion matrix, without normalization
[[1624  922]
 [4667 9287]]
```



Conclusion

On applying XGBoost BOW on DonorsChoose dataset with max depth 3 and n estimators as 200 we got the AUC value of 0.70 on test data

XGBoost TFIDF

In [193]:

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

X_tr =
hstack((X_train_essay_tfidf,X_train_title_tfidf,encode_state_train,encode_prefix_train,encode_grade
_train,encode_cat_train,encode_subcat_train,X_train_price_norm,X_train_post_norm,X_train_digit_norm
,X_train_quantity_norm))
X_cr = hstack((X_cv_essay_tfidf,X_cv_title_tfidf,encode_state_cv,encode_prefix_cv,encode_grade_cv,
encode_cat_cv,encode_subcat_cv,X_cv_price_norm,X_cv_post_norm,X_cv_digit_norm,X_cv_quantity_norm))
X_te =
hstack((X_test_essay_tfidf,X_test_title_tfidf,encode_state_test,encode_prefix_test,encode_grade_tes
t,encode_cat_test,encode_subcat_test,X_test_price_norm,X_test_post_norm,X_test_digit_norm,X_test_qu
antity_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, 6152) (22445,)
(11055, 6152) (11055,)
(16500, 6152) (16500,)
====================================================================================================
```

```
# 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)


parameters = {'max_depth':[1,2,3,4,5],'n_estimators':[1, 5, 10, 50, 100, 200]}
clf = GridSearchCV(XG, parameters, cv=3, scoring='roc_auc')
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 [196]:

```
clf.best_params_
```

Out[196]:

```
{'max_depth': 5, 'n_estimators': 200}
```

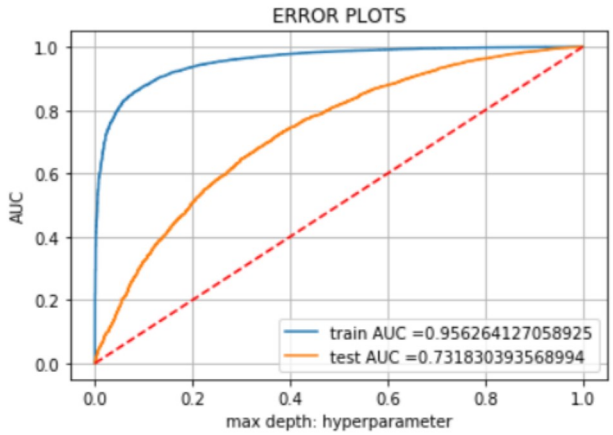
In [197]:

```
# 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')
```

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



XG = xgb.XGBClassifier(max_depth=5,n_estimators= 200,subsample=0.7, colsample_bytree=0.7, random_st
ate=0,class_weight='balanced', reg_alpha=1,reg_lambda=0)
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 [199]:

```
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("max depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [200]:

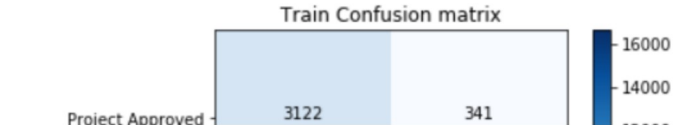
```
train_conf=confusion_matrix(y_train, predict(preds, tr_thresholds, train_fpr, train_tpr))
test_conf=confusion_matrix(y_test, predict(preds2, te_thresholds, test_fpr, test_tpr))
```

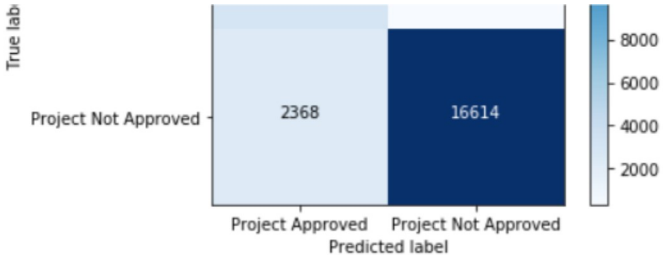
```
the maximum value of tpr*(1-fpr) 0.7890647531395383 for threshold 0.799
the maximum value of tpr*(1-fpr) 0.4534504630352609 for threshold 0.855
```

In [201]:

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

```
Confusion matrix, without normalization
[[ 3122   341]
 [ 2368 16614]]
```

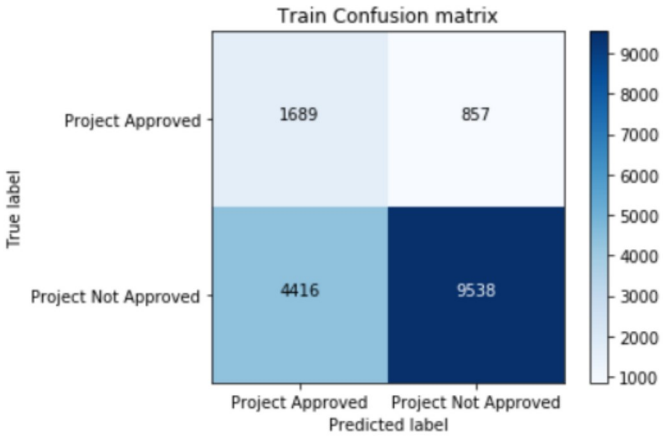




In [202]:

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

```
Confusion matrix, without normalization
[[1689  857]
 [4416 9538]]
```



Conclusion

On applying XGBoost TFIDF on DonorsChoose dataset with max depth 5 and n estimators as 200 we got the AUC value of 0.73 on test data

XGBoost AVG W2V

In [203]:

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

X_tr =
hstack((X_train_avg_title,X_train_avg_essay,encode_state_train,encode_prefix_train,encode_grade_tra
in,encode_cat_train,encode_subcat_train,X_train_price_norm,X_train_post_norm,X_train_digit_norm,X_
train_quantity_norm))
X_cr =
hstack((X_cv_avg_title,X_cv_avg_essay,encode_state_cv,encode_prefix_cv,encode_grade_cv,encode_cat_c
v,encode_subcat_cv,X_cv_price_norm,X_cv_post_norm,X_cv_digit_norm,X_cv_quantity_norm))
X_te = hstack((X_test_avg_t,X_test_avg_e,encode_state_test,encode_prefix_test,encode_grade_test,en
code_cat_test,encode_subcat_test,X_test_price_norm,X_test_post_norm,X_test_digit_norm,X_test_quanti
ty_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
```

```
(16500, 609) (16500,)
==========================================================================================================
```

In [204]:

```
# 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)


parameters = {'max_depth':[1,2,3,4,5],'n_estimators':[1, 5, 10, 50, 100, 200]}
clf = GridSearchCV(XG, parameters, cv=3, scoring='roc_auc')
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 [205]:

```
# 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')
```

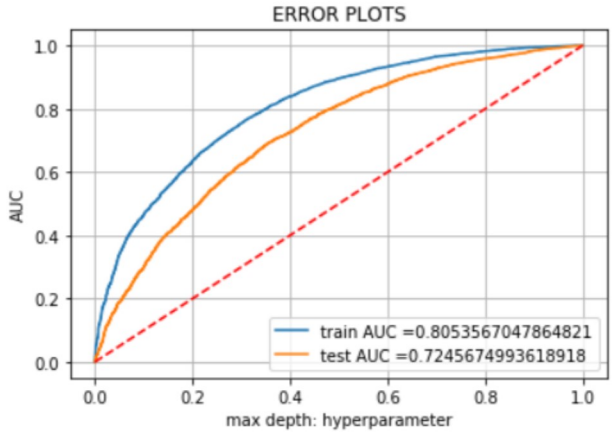
In [206]:

```
# https://scikit-
```

```
XG = xgb.XGBClassifier(max_depth=2,n_estimators= 200,subsample=0.7, colsample_bytree=0.7, random_st
ate=0,class_weight='balanced', reg_alpha=1,reg_lambda=0)
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("max depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [208]:

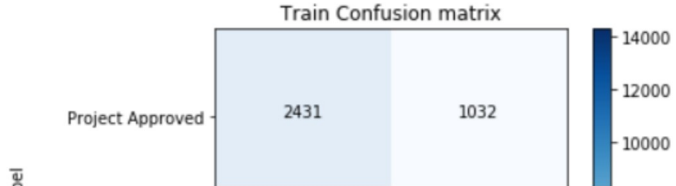
```
train_conf=confusion_matrix(y_train, predict(preds, tr_thresholds, train_fpr, train_tpr))
test_conf=confusion_matrix(y_test, predict(preds2, te_thresholds, test_fpr, test_tpr))
```

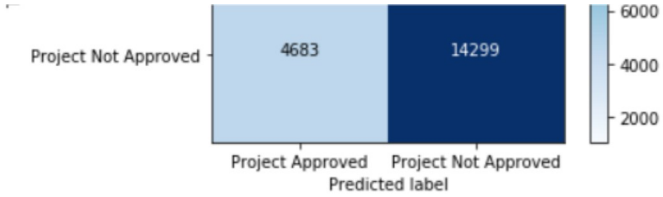
```
the maximum value of tpr*(1-fpr) 0.5288057445975309 for threshold 0.826
the maximum value of tpr*(1-fpr) 0.4464348745023627 for threshold 0.839
```

In [209]:

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

```
Confusion matrix, without normalization
[[ 2431  1032]
 [ 4683 14299]]
```

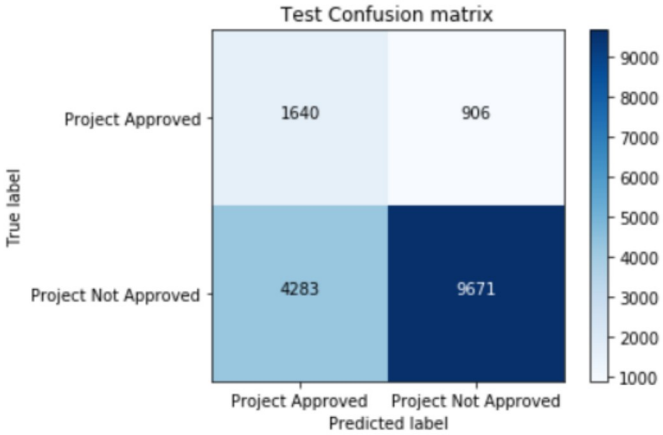




In [210]:

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

```
Confusion matrix, without normalization
[[1640  906]
 [4283 9671]]
```



Conclusion

On applying XGBoost AVG W2V on DonorsChoose dataset with max depth 2 and n estimators as 200 we got the AUC value of 0.72 on test data

XGBoost TFIDF W2V

In [211]:

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

X_tr =
hstack((X_train_tfidf_title,X_train_tfidf_essay,encode_state_train,encode_prefix_train,encode_grade
_train,encode_cat_train,encode_subcat_train,X_train_price_norm,X_train_post_norm,X_train_digit_norm
,X_train_quantity_norm))
X_cr = hstack((X_cv_tfidf_title,X_cv_tfidf_essay,encode_state_cv,encode_prefix_cv,encode_grade_cv,
encode_cat_cv,encode_subcat_cv,X_cv_price_norm,X_cv_post_norm,X_cv_digit_norm,X_cv_quantity_norm))
X_te =
hstack((X_test_tfidf_t,X_test_tfidf_e,encode_state_test,encode_prefix_test,encode_grade_test,encode
_cat_test,encode_subcat_test,X_test_price_norm,X_test_post_norm,X_test_digit_norm,X_test_quantity_n
orm))

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, 609) (22445,)
(11055, 609) (11055,)
(16500, 609) (16500,)
```

In [212]:

```
# 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)


parameters = {'max_depth':[1,2,3,4,5],'n_estimators':[1, 5, 10, 50, 100, 200]}
clf = GridSearchCV(XG, parameters, cv=3, scoring='roc_auc')
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 [213]:

```
# 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')
```

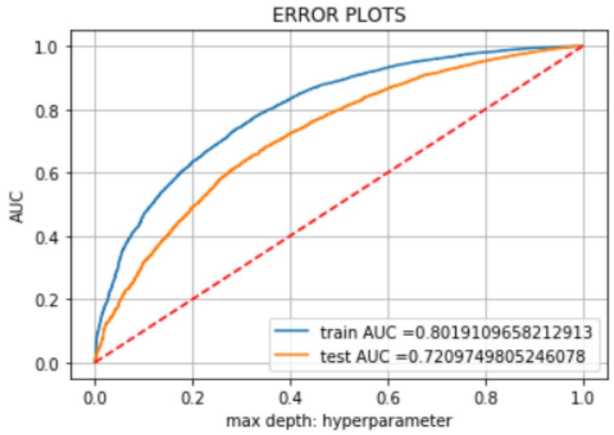
In [214]:

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

```
XG = xgb.XGBClassifier(max_depth=2,n_estimators= 200,subsample=0.7, colsample_bytree=0.7, random_st
ate=0,class_weight='balanced', reg_alpha=1,reg_lambda=0)
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 [215]:

```
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("max depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [216]:

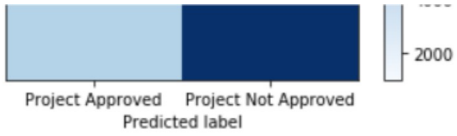
```
train_conf=confusion_matrix(y_train, predict(preds, tr_thresholds, train_fpr, train_tpr))
test_conf=confusion_matrix(y_test, predict(preds2, te_thresholds, test_fpr, test_tpr))
```

```
the maximum value of tpr*(1-fpr) 0.5247998673941692 for threshold 0.828
the maximum value of tpr*(1-fpr) 0.4429248565677755 for threshold 0.841
```

In [217]:

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

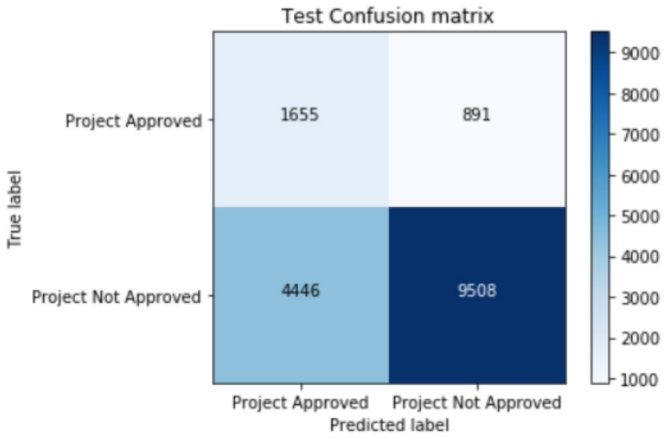
```
Confusion matrix, without normalization
[[ 2462  1001]
 [ 4970 14012]]
```



In [218]:

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

```
Confusion matrix, without normalization
[[1655  891]
 [4446 9508]]
```



Conclusion

On applying XGBoost TFIDF W2V on DonorsChoose dataset with max depth 2 and n estimators as 200 we got the AUC value of 0.72 on test data

In [114]:

```
#code copied from -http://zetcode.com/python/prettytable/

x = PrettyTable()
x.field_names = ["Vectorizer", "Max depth","N estimatior", "AUC"]
x.add_row(["Random Forest BOW",  10,200, 0.69])
x.add_row(["Random Forest TFIDF",  10,200,0.71])
x.add_row(["Random Forest AVG W2V",  5, 200, 0.69])
x.add_row(["Random Forest TFIDF W2V",  5,200,0.69])
x.add_row(["XGBoost BOW",  3,200, 0.70])
x.add_row(["XGBoost TFIDF",  5,200,0.73])
x.add_row(["XGBoost AVG W2V",  2,200,0.72])
x.add_row(["XGBoost TFIDF W2V",  2,200,0.72])
print(x)
```

```
+------------------------+-----------+--------------+------+
|       Vectorizer       | Max depth | N estimatior | AUC  |
+------------------------+-----------+--------------+------+
|    Random Forest BOW   |     10    |     200      | 0.69 |
|   Random Forest TFIDF  |     10    |     200      | 0.71 |
|  Random Forest AVG W2V |     5     |     200      | 0.69 |
| Random Forest TFIDF W2V |     5     |     200      | 0.69 |
|       XGBoost BOW      |     3     |     200      | 0.7  |
|      XGBoost TFIDF     |     5     |     200      | 0.73 |
|     XGBoost AVG W2V    |     2     |     200      | 0.72 |
|    XGBoost TFIDF W2V   |     2     |     200      | 0.72 |
+------------------------+-----------+--------------+------+
```

TFIDF