Importing libraries

In [73]:

```
%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
```

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)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)

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,)
```

3. Vectorizing data

3.1 BOW Vectorization of Clean_essay

In [4]:

```
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 one hot encoding the catogorical features: state

In [5]:

```
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
```

```
', 'wy']
==================================================================================================
```

3.3 one hot encoding the catogorical features: teacher_prefix

In [6]:

```
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 [7]:

```
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']
==================================================================================================
```

3.5 one hot encoding the catogorical features: project_title

In [8]:

```
vec_title = CountVectorizer()
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_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, 7858) (22445,)
(11055, 7858) (11055,)
(16500, 7858) (16500,)
====================================================================================================
```

3.6 one hot encoding the catogorical features: project_category

In [9]:

```
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("After vectorizations")
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 [10]:

```
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',
```

```
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
==========================================================================================
```

3.8 Normalizing the numerical features: Price

In [11]:

```
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)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

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

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

In [12]:

```
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

```
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 [14]:

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

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

Concatenation of all the vectorized data

In [15]:

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

X_tr =
hstack((X_train_essay_bow,X_train_state_ohe,X_train_teacher_ohe,X_train_grade_ohe,X_train_title_ohe
,X_train_category_ohe,X_train_subcategory_ohe,X_train_price_norm,X_train_post_norm,X_train_digit_no
rm,X_train_quantity_norm))
X_cr = hstack((X_cv_essay_bow,X_cv_state_ohe,X_cv_teacher_ohe,X_cv_grade_ohe,X_cv_title_ohe,X_cv_ca
tegory_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_essay_bow,X_test_state_ohe,X_test_teacher_ohe,X_test_grade_ohe,X_test_title_ohe,X_te
st_category_ohe,X_test_subcategory_ohe,X_test_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)
```

```
Final Data matrix
(22445, 12959) (22445,)
(11055, 12959) (11055,)
(16500, 12959) (16500,)
==========================================================================================================
```

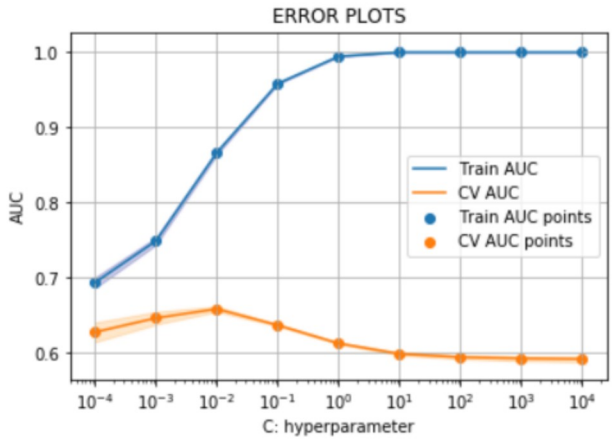
Applying LogisticRegression on BOW

In [52]:

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


LR = LogisticRegression(class_weight='balanced')
parameters = {'C':[10**-4,10**-3,10**-2,10**-1,1,10**1,10**2,10**3,10**4]}
clf = GridSearchCV(LR, 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 [53]:

```
plt.plot(parameters['C'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['C'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.2,color='darkblue')

plt.plot(parameters['C'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['C'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='dar
korange')

plt.scatter(parameters['C'], train_auc, label='Train AUC points')
plt.scatter(parameters['C'], cv_auc, label='CV AUC points')
pyplot.xscale('log')

plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

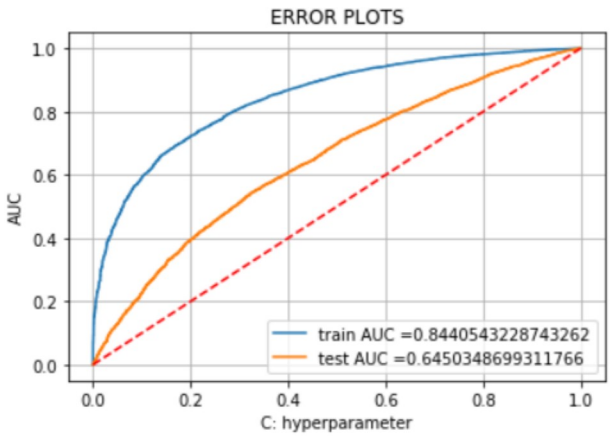


From the above graph we can see that for the value of C less than 0.01, it is over fitting the data and for the value of C more than 0.01, it is underfitting the data. So the best value of C will be 0.01 for this

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
best_C = 0.01


LR = LogisticRegression(class_weight='balanced', C=best_C)
LR.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
y_train_pred = LR.predict_proba(X_tr)
preds = y_train_pred[:,1] #code copied from https://stackoverflow.com/questions/25009284/how-to-pl
ot-roc-curve-in-python to remove the error-bad input shape
y_test_pred = LR.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 [55]:

```
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 [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(fpr*(1-tpr))]

    # (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 [56]:

```
test_conf=confusion_matrix(y_test, predict(preds2, tr_thresholds, test_fpr, test_fpr))
```

```
the maximum value of tpr*(1-fpr) 0.24999997915341 for threshold 0.4
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.441
```

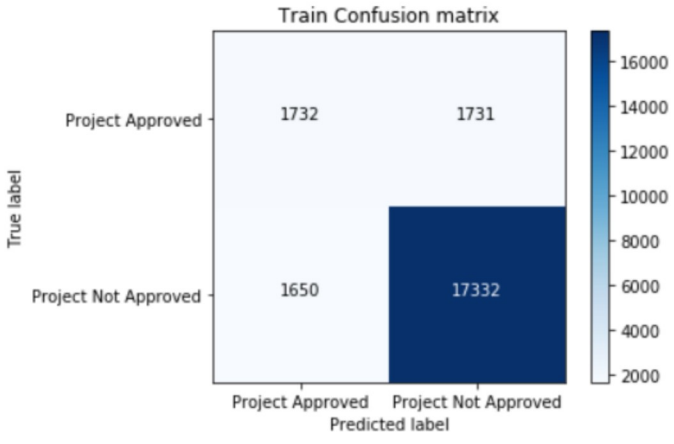
In [70]:

```
#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.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 [57]:

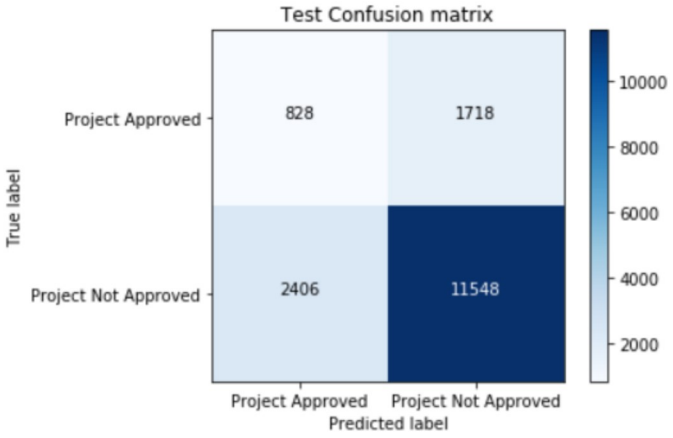
```
plot_confusion_matrix(train_conf, classes=class_names,
                      title='Train Confusion matrix')
```

```
Confusion matrix, without normalization
[[ 1732  1731]
 [ 1650 17332]]
```



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

```
Confusion matrix, without normalization
[[  828  1718]
 [ 2406 11548]]
```



Conclusion

On applying LogisticRegression BOW on DonorsChoose dataset with C value as 0.01 we got the AUC value of 0.64 on test data

Applying LogisticRegression on TFIDF

TFIDF Vectorization of Clean_essay

In [49]:

```
vec_essay = TfidfVectorizer(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_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 [50]:

```
vec_title = TfidfVectorizer()
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)
```

```
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, 7858) (22445,)
(11055, 7858) (11055,)
(16500, 7858) (16500,)
```

Concatenation of all the vectorized data

In [59]:

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

X_tr =
hstack((X_train_essay_tfidf,X_train_state_ohe,X_train_teacher_ohe,X_train_grade_ohe,X_train_title_t
fidf,X_train_category_ohe,X_train_subcategory_ohe,X_train_price_norm,X_train_post_norm,X_train_digi
t_norm,X_train_quantity_norm))
X_cr = hstack((X_cv_essay_tfidf,X_cv_state_ohe,X_cv_teacher_ohe,X_cv_grade_ohe,X_cv_title_tfidf,X_c
v_category_ohe,X_cv_subcategory_ohe,X_cv_price_norm,X_cv_post_norm,X_cv_digit_norm,X_cv_quantity_nc
rm))
X_te =
hstack((X_test_essay_tfidf,X_test_state_ohe,X_test_teacher_ohe,X_test_grade_ohe,X_test_title_tfidf
,X_test_category_ohe,X_test_subcategory_ohe,X_test_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, 12959) (22445,)
(11055, 12959) (11055,)
(16500, 12959) (16500,)
====================================================================================================
```

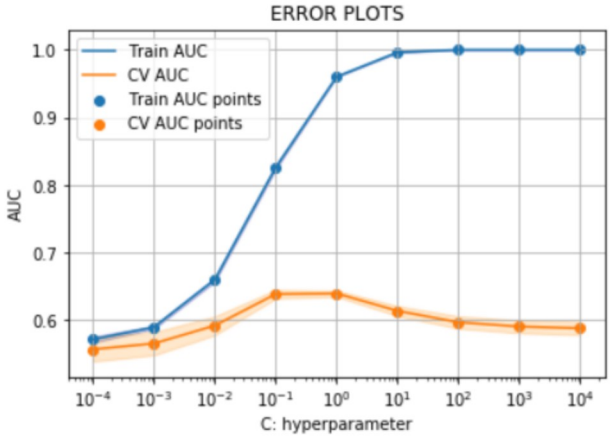
In [60]:

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


LR = LogisticRegression(class_weight='balanced')
parameters = {'C':[10**-4,10**-3,10**-2,10**-1,1,10**1,10**2,10**3,10**4]}
clf = GridSearchCV(LR, 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 [61]:

```
plt.plot(parameters['C'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['C'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.2,color='darkblue')

plt.plot(parameters['C'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['C'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='dar
korange')

plt.scatter(parameters['C'], train_auc, label='Train AUC points')
```

```
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



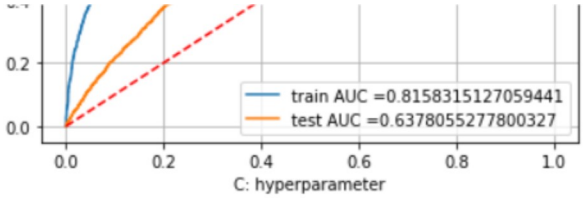
From the above graph we can see that for the value of C less than 0.1, it is over fitting the data and for the value of C more than 0.1, it is underfitting the data. So the best value of C will be 0.1 for this

In [62]:

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

best_C = 0.1


LR = LogisticRegression(class_weight='balanced', C=best_C)
LR.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
y_train_pred = LR.predict_proba(X_tr)
preds = y_train_pred[:,1] #code copied from https://stackoverflow.com/questions/25009284/how-to-pl
ot-roc-curve-in-python to remove the error-bad input shape
y_test_pred = LR.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 [63]:

```
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 [65]:

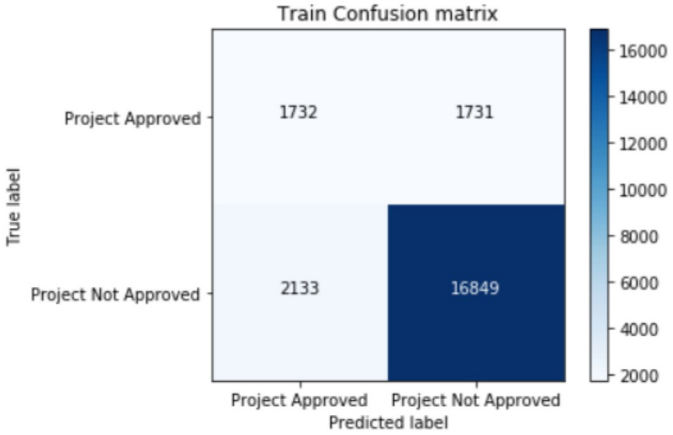
```
train_conf=confusion_matrix(y_train, predict(preds, tr_thresholds, train_fpr, train_fpr))
test_conf=confusion_matrix(y_test, predict(preds2, tr_thresholds, test_fpr, test_fpr))
```

```
the maximum value of tpr*(1-fpr) 0.24999997915341 for threshold 0.432
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.467
```

In [66]:

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

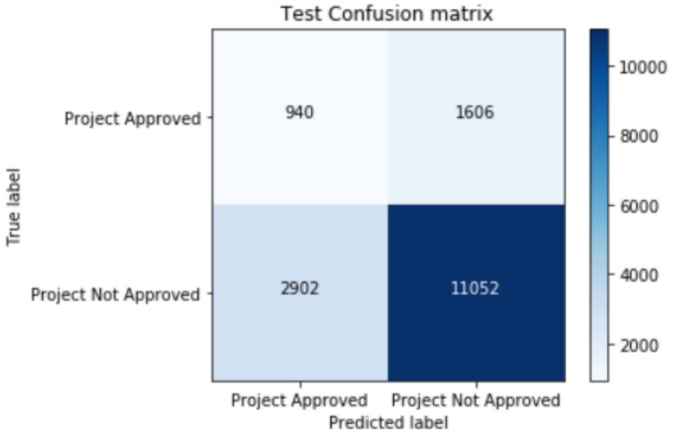
```
Confusion matrix, without normalization
[[ 1732  1731]
 [ 2133 16849]]
```



In [67]:

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

```
Confusion matrix, without normalization
[[  940  1606]
 [ 2902 11052]]
```



Conclusion

On applying LogisticRegression TFIDF on DonorsChoose dataset with C value as 0.1 we got the AUC value of 0.64 on test data

Applying LogisticRegression on AVG W2V

AVG_W2V Vectorization of Clean_essay

In [68]:

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

with open('glove_vectors.txt', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [69]:

```
# 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 [70]:

```
# 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 [71]:

```
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[71]:

```
(50000, 300)
```

In [72]:

```
=0.33, stratify=y)
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)
```

In [73]:

```
#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)
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)
```

Concatenation of all the vectorized data

In [74]:

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

X_tr =
hstack((X_train_avg_title,X_train_avg_essay,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_no
rm,X_train_quantity_norm))
X_cr = hstack((X_cv_avg_title,X_cv_avg_essay,X_cv_state_ohe,X_cv_teacher_ohe,X_cv_grade_ohe,X_cv_ca
tegory_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_avg_t,X_test_avg_e,X_test_state_ohe,X_test_teacher_ohe,X_test_grade_ohe,X_test_categ
ory_ohe,X_test_subcategory_ohe,X_test_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, 701) (22445,)
(11055, 701) (11055,)
(16500, 701) (16500,)
====================================================================================================
```

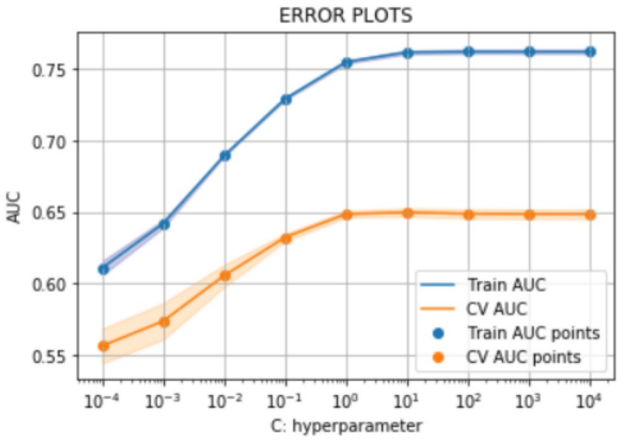
In [75]:

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


LR = LogisticRegression(class_weight='balanced')
parameters = {'C':[10**-4,10**-3,10**-2,10**-1,1,10**1,10**2,10**3,10**4]}
clf = GridSearchCV(LR, 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 [76]:

```
plt.plot(parameters['C'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['C'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.2,color='darkblue')

plt.plot(parameters['C'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['C'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='dar
```

```
plt.scatter(parameters['C'], train_auc, label='Train AUC points')
plt.scatter(parameters['C'], cv_auc, label='CV AUC points')
pyplot.xscale('log')

plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



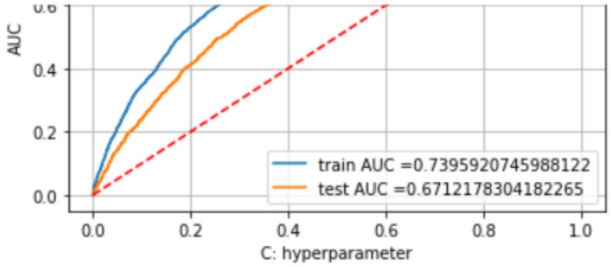
From the above graph we can see that for the value of C less than 1, it is over fitting the data and for the value of C more than 1, it is underfitting the data. So the best value of C will be 1 for this

In [77]:

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

best_C = 1


LR = LogisticRegression(class_weight='balanced', C=best_C)
LR.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
y_train_pred = LR.predict_proba(X_tr)
preds = y_train_pred[:,1] #code copied from https://stackoverflow.com/questions/25009284/how-to-pl
ot-roc-curve-in-python to remove the error-bad input shape
y_test_pred = LR.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 [78]:

```
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 [79]:

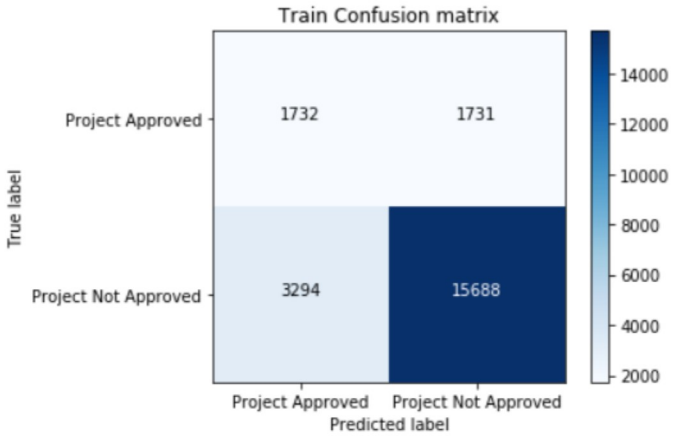
```
train_conf=confusion_matrix(y_train, predict(preds, tr_thresholds, train_fpr, train_fpr))
test_conf=confusion_matrix(y_test, predict(preds2, tr_thresholds, test_fpr, test_fpr))
```

```
the maximum value of tpr*(1-fpr) 0.24999997915341 for threshold 0.404
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.482
```

In [80]:

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

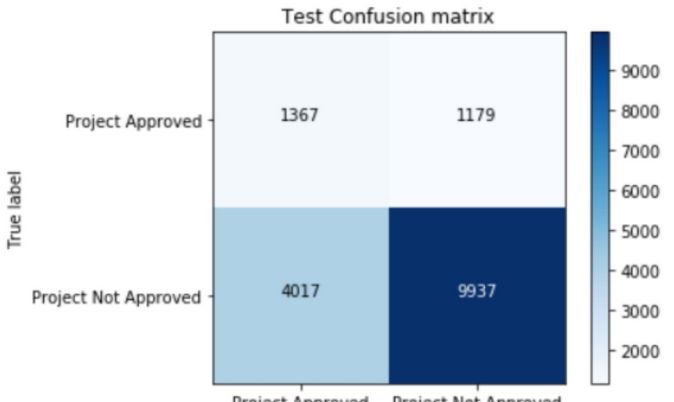
```
Confusion matrix, without normalization
[[ 1732  1731]
 [ 3294 15688]]
```



In [81]:

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

```
Confusion matrix, without normalization
[[1367 1179]
 [4017 9937]]
```



Conclusion

On applying LogisticRegression AVG W2V on DonorsChoose dataset with C value as 1 we got the AUC value of 0.67 on test data

Applying LogisticRegression on TFIDF W2V

TFIDF_W2V Vectorization of Clean_essay

In [82]:

```
# 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 [83]:

```
# 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 [84]:

```
# 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 [85]:

```
# 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
```

```
            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 [86]:

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

Out[86]:

```
(50000, 300)
```

In [87]:

```
#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)
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)
```

In [88]:

```
#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)
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)
```

Concatenation of all the vectorized data

In [89]:

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

X_tr =
hstack((X_train_tfidf_essay,X_train_tfidf_title,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_digi
t_norm,X_train_quantity_norm))
X_cr = hstack((X_cv_tfidf_essay,X_cv_tfidf_title,X_cv_state_ohe,X_cv_teacher_ohe,X_cv_grade_ohe,X_c
v_category_ohe,X_cv_subcategory_ohe,X_cv_price_norm,X_cv_post_norm,X_cv_digit_norm,X_cv_quantity_nc
rm))
X_te = hstack((X_test_tfidf_e,X_test_tfidf_t,X_test_state_ohe,X_test_teacher_ohe,X_test_grade_ohe,X
_test_category_ohe,X_test_subcategory_ohe,X_test_price_norm,X_test_post_norm,X_test_digit_norm,X_t
est_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, 701) (22445,)
(11055, 701) (11055,)
(16500, 701) (16500,)
====================================================================================================
```

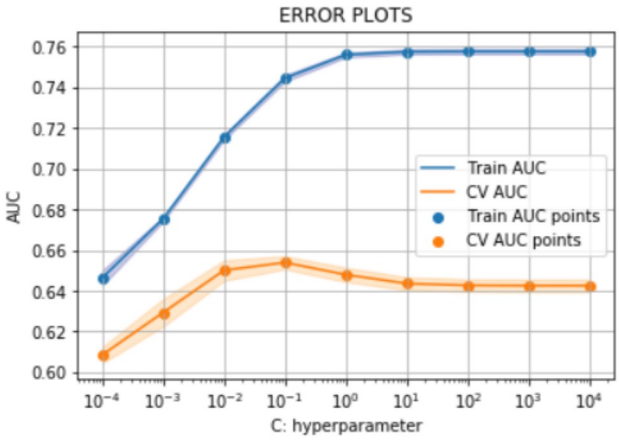
In [90]:

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

```
parameters = {'C':[10**-4,10**-3,10**-2,10**-1,1,10**1,10**2,10**3,10**4]}
clf = GridSearchCV(LR, 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 [91]:

```
plt.plot(parameters['C'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['C'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.2,color='darkblue')

plt.plot(parameters['C'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['C'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='dar
korange')

plt.scatter(parameters['C'], train_auc, label='Train AUC points')
plt.scatter(parameters['C'], cv_auc, label='CV AUC points')
pyplot.xscale('log')

plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



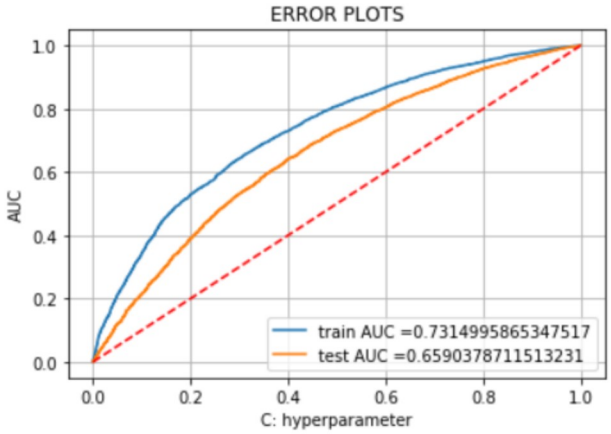
From the above graph we can see that for the value of C less than 0.1, it is over fitting the data and for the value of C more than 0.1, it is underfitting the data. So the best value of C will be 0.1 for this

In [92]:

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

best_C = 0.1


LR = LogisticRegression(class_weight='balanced', C=best_C)
LR.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
y_train_pred = LR.predict_proba(X_tr)
preds = y_train_pred[:,1] #code copied from https://stackoverflow.com/questions/25009284/how-to-pl
ot-roc-curve-in-python to remove the error-bad input shape
y_test_pred = LR.predict_proba(X_te)
preds2=y_test_pred[:,1]
```

In [93]:

```
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 [94]:

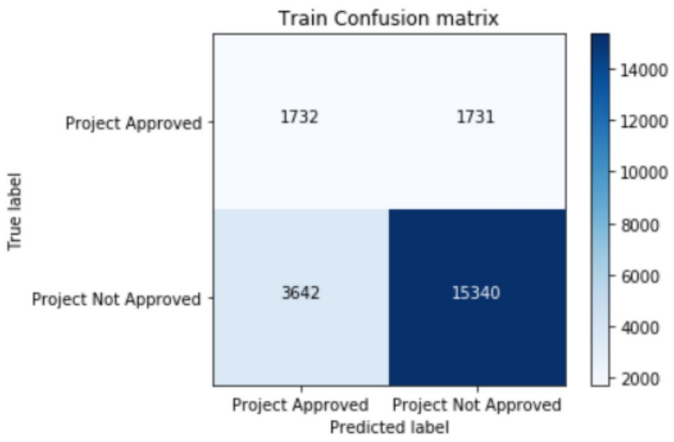
```
train_conf=confusion_matrix(y_train, predict(preds, tr_thresholds, train_fpr, train_fpr))
test_conf=confusion_matrix(y_test, predict(preds2, tr_thresholds, test_fpr, test_fpr))
```

```
the maximum value of tpr*(1-fpr) 0.24999997915341 for threshold 0.415
the maximum value of tpr*(1-fpr) 0.24999984572938835 for threshold 0.478
```

In [95]:

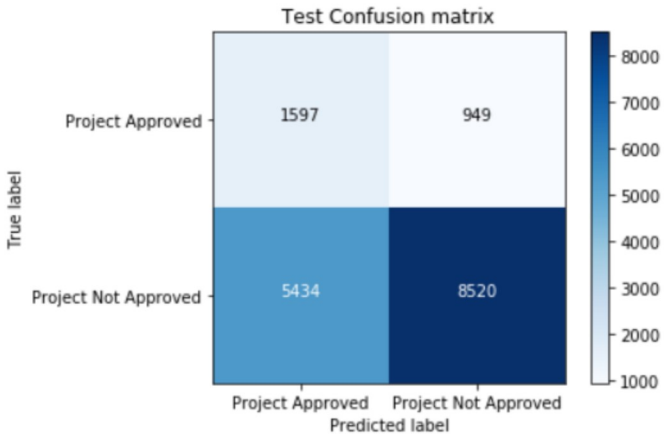
```
plot_confusion_matrix(train_conf, classes=class_names,
                      title='Train Confusion matrix')
```

```
Confusion matrix, without normalization
[[ 1732  1731]
 [ 3642 15340]]
```



In [96]:

```
Confusion matrix, without normalization
[[1597  949]
 [5434 8520]]
```



Conclusion

On applying LogisticRegression TFIDF W2V on DonorsChoose dataset with C value as 0.1 we got the AUC value of 0.65 on test data

Logistic Regression with added Features `Set 5`

In [16]:

```
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 [19]:

```
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 [47]:

```
#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 [51]:

```
X.tail(2)
```

Out[51]:

	teacher_prefix	school_state	project_grade_category	teacher_number_of_previously_posted_projects	price	quar
49998	Mrs.	CT	Grades PreK-2	37	102.25	6
49999	Mrs.	KY	Grades PreK-2	0	505.43	41

Splitting Data

In [52]:

```
# train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)

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

```
(22445, 14) (22445,)
(11055, 14) (11055,)
(16500, 14) (16500,)
```

Normalizing the numerical features: Title_count

In [57]:

```
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 [58]:

```
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 [59]:

```
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 [61]:

```
# 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_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_state_ohe,X_cv_teacher_ohe,X_c
v_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_state_ohe,X_test_teacher_ohe
,X_test_grade_ohe,X_test_category_ohe,X_test_subcategory_ohe,X_test_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, 104) (22445,)
```

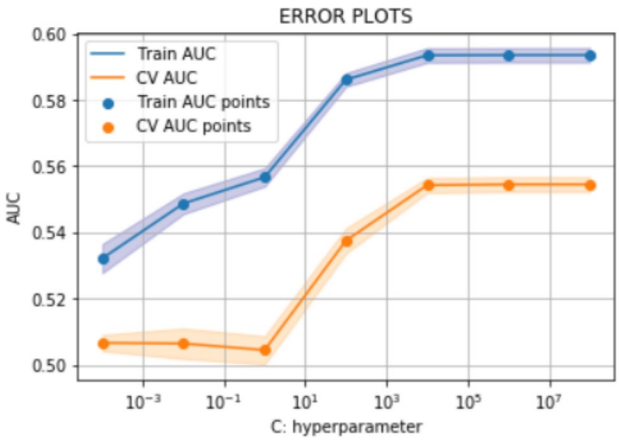
==========

In [64]:

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


LR = LogisticRegression(class_weight='balanced')
parameters = {'C':[10**-4,10**-2,1,10**2,10**4,10**6,10**8]}
clf = GridSearchCV(LR, 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 [65]:

```
plt.plot(parameters['C'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['C'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.2,color='darkblue')

plt.plot(parameters['C'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['C'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='dar
korange')

plt.scatter(parameters['C'], train_auc, label='Train AUC points')
plt.scatter(parameters['C'], cv_auc, label='CV AUC points')
pyplot.xscale('log')

plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



From the above graph we can see that for the value of C less than 10000, it is over fitting the data and for the value of C more than 10000, it is underfitting the data. So the best value of C will be 10000 for this

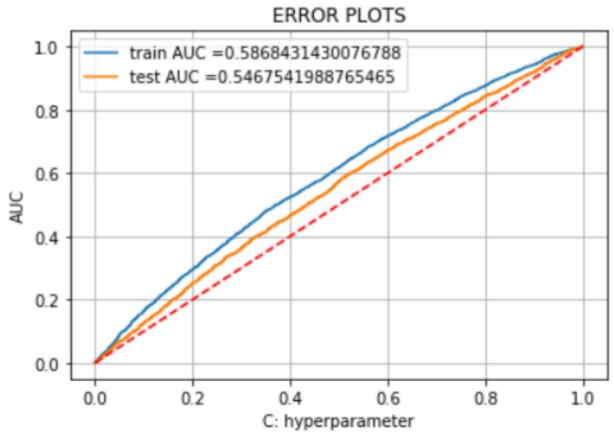
In [66]:

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

best_C = 10000


LR = LogisticRegression(class_weight='balanced', C=best_C)
```

```
class
# not the predicted outputs
y_train_pred = LR.predict_proba(X_tr)
preds = y_train_pred[:,1] #code copied from https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python to remove the error-bad input shape
y_test_pred = LR.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 [67]:

```
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 [71]:

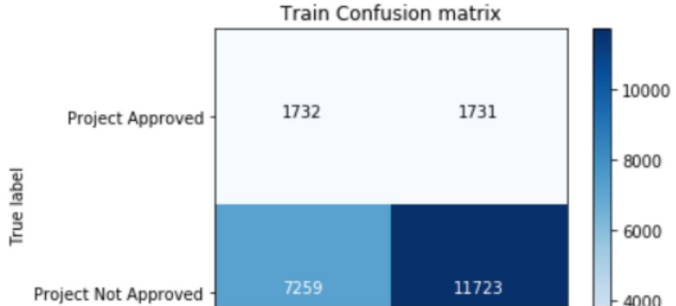
```
train_conf=confusion_matrix(y_train, predict(preds, tr_thresholds, train_fpr, train_fpr))
test_conf=confusion_matrix(y_test, predict(preds2, tr_thresholds, test_fpr, test_fpr))
```

```
the maximum value of tpr*(1-fpr) 0.24999997915341 for threshold 0.481
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.503
```

In [74]:

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

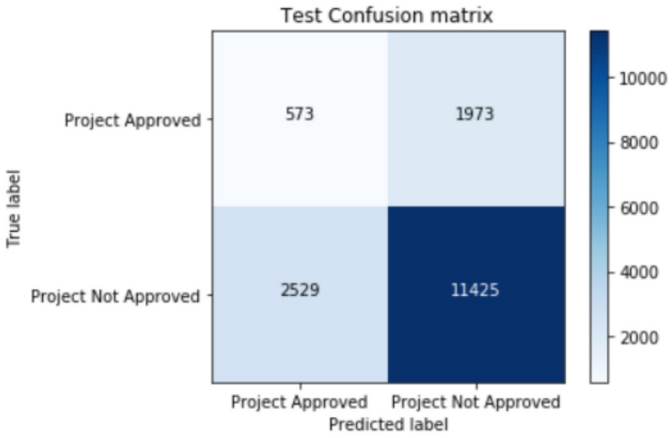
```
Confusion matrix, without normalization
[[ 1732  1731]
 [ 7259 11723]]
```



In [75]:

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

```
Confusion matrix, without normalization
[[  573  1973]
 [ 2529 11425]]
```



Conclusion- Text features are very important for a accurate prediction in DonorsChoose Dataset as without the text features we got the AUC score of 0.54 on test data

In [76]:

```
#code copied from -http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Hyper Parameter (alpha)", "AUC"]
x.add_row(["BOW",  0.01, 0.64])
x.add_row(["TFIDF",  0.1, 0.63])
x.add_row(["AVG W2V",  1, 0.67])
x.add_row(["TFIDF W2V",  0.1, 0.65])
x.add_row(["Without text features",  10000, 0.54])
print(x)
```

```
+----------------------+-------------------------+------+
|      Vectorizer      | Hyper Parameter (alpha) | AUC  |
+----------------------+-------------------------+------+
|         BOW          |           0.01          | 0.64 |
|        TFIDF         |           0.1           | 0.63 |
|       AVG W2V        |            1            | 0.67 |
|      TFIDF W2V       |           0.1           | 0.65 |
| Without text features |          10000          | 0.54 |
+----------------------+-------------------------+------+
```

Conclusion- The best vectorizer was found to be AVG W2V while applying Logistic Regression.