DonorsChoose

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of funding. Right now, a large number of volunteers is needed to manually screen each submission before it's approved to be posted on the DonorsChoose.org website.

Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, there are three main problems they need to solve:

- How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as quickly and as efficiently as possible
- How to increase the consistency of project vetting across different volunteers to improve the experience for teachers
- How to focus volunteer time on the applications that need the most assistance

The goal of the competition is to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved, using the text of project descriptions as well as additional metadata about the project, teacher, and school. DonorsChoose.org can then use this information to identify projects most likely to need further review before approval.

About the DonorsChoose Data Set

The `train.csv` data set provided by DonorsChoose contains the following features:

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal code). **Example:** `WY`
	One or more (comma-separated) subject subcategories for the project. **Examples:**

Feature	Description
	• `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy materials to manage sensory needs!`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*
`project_essay_4`	Fourth application essay*
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teacher. **Example:** `2`

* See the section **Notes on the Essay Data** for more details about these features.

Additionally, the `resources.csv` data set provides more data about the resources required for each project. Each line in this file represents a resource required by a project:

Feature	Description
`id`	A `project_id` value from the `train.csv` file. **Example:** `p036502`
`description`	Desciption of the resource. **Example:** `Tenor Saxophone Reeds, Box of 25`
`quantity`	Quantity of the resource required. **Example:** `3`
`price`	Price of the resource required. **Example:** `9.95`

Note: Many projects require multiple resources. The `id` value corresponds to a `project_id` in train.csv, so you use it as a key to retrieve all resources needed for a project:

The data set contains the following label (the value you will attempt to predict):

Label	Description
`project_is_approved`	A binary flag indicating whether DonorsChoose approved the project. A value of `0` indicates the project was not approved, and a value of `1` indicates the project was approved.

Notes on the Essay Data

Prior to May 17, 2016, the prompts for the essays were as follows:

- __project_essay_1:__ "Introduce us to your classroom"
- __project_essay_2:__ "Tell us more about your students"
- __project_essay_3:__ "Describe how your students will use the materials you're requesting"
- __project_essay_4:__ "Close by sharing why your project will make a difference"

Starting on May 17, 2016, the number of essays was reduced from 4 to 2, and the prompts for the first 2 essays were changed to the following:

your neighborhood, and your school are all helpful.

- __project_essay_2:__ "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

  For all projects with project_submitted_datetime of 2016-05-17 and later, the values of project_essay_3 and project_essay_4 will be NaN.

1.1 Importing libraries

In [1]:

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

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# 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 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.neighbors import KNeighborsClassifier
```

1.2 Reading Data

In [2]:

```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [3]:

```
# merge two column text dataframe:
```

```
                          project_data["project_essay_3"].map(str) + \
                          project_data["project_essay_4"].map(str)
```

In [4]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

In [5]:

```
project_data.head(1)
```

Out[5]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	proje
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grade

In [6]:

```
project_data=project_data[0:50000]
```

1.3 Text preprocessing

1.3.1 Essay Text

In [7]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [8]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
from nltk.tokenize import word_tokenize
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
```

```
'these', 'those', \
           'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
           'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
           'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
           'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
           'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
           'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
           's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
           've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "d
esn't", 'hadn',\
           "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
           "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
           'won', "won't", 'wouldn', "wouldn't"]
```

In [9]:

```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in project_data['essay'].values:
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent=sent.lower()
    word_tokens = word_tokenize(sent)
    # https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
    sent = ' '.join(w for w in word_tokens if not w in stopwords)
    #sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essays.append(sent)
```

In [10]:

```
project_data['clean_essay'] = preprocessed_essays
project_data.drop(['project_essay_1',
'essay','project_essay_2','project_essay_3','project_essay_4'], axis=1, inplace=True)
```

1.3.2 Project title Text

In [11]:

```
# similarly you can preprocess the titles also
```

In [12]:

```
preprocessed_title=[]
# tqdm is for printing the status bar
for sentance in project_data['project_title'].values:
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent=sent.lower()
    word_tokens = word_tokenize(sent)
    # https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
    sent = ' '.join(w for w in word_tokens if not w in stopwords)
    #sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_title.append(sent)
```

In [13]:

```
project_data['clean_title'] = preprocessed_title
project_data.drop(['project_title'], axis=1, inplace=True)
```

1.3.3 Categories

In [14]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())
```

In [15]:

```
project_data['clean_categories'] = cat_list
project_data.drop(['project_subject_categories'], axis=1, inplace=True)
```

1.3.4 Subcategories

In [16]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())
```

In [17]:

```
project_data['clean_subcategories'] = sub_cat_list
```

In [18]:

```
#check presence of digit in string python- https://stackoverflow.com/questions/19859282/check-if-a
-string-contains-a-number
def num_there(s):
    if(type(s)==int or type(s)==float):
        return True
    return any(i.isdigit() for i in s)
# No common columns to perform merge on. https://stackoverflow.com/questions/40468069/merge-two-da
taframes-by-index
sub_summary = list(project_data['project_resource_summary'].values)
project_data['Is_digit_present']=0
j=0
for i in sub_summary :
    a=num_there(i)
    if(a==True):
        project_data.loc[j,'Is_digit_present']=1
    j=j+1
```

In [19]:

```
project_data.drop(['project_resource_summary'], axis=1, inplace=True)
```

1. 4 Preparing data for models

In [20]:

```
project_data.columns
```

Out[20]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'price', 'quantity', 'clean_essay', 'clean_title', 'clean_categories',
       'clean_subcategories', 'Is_digit_present'],
      dtype='object')
```

we are going to consider

```
- school_state : categorical data
- clean_categories : categorical data
- clean_subcategories : categorical data
- project_grade_category : categorical data
- teacher_prefix : categorical data

- project_title : text data
- text : text data
- project_resource_summary: text data

- quantity : numerical
- teacher_number_of_previously_posted_projects : numerical
- price : numerical
```

We are going to remove all the unnecessary columns from the dataset

In [21]:

```
project_data.drop(['id','teacher_id','project_submitted_datetime'], axis=1, inplace=True)
```

In [22]:

```
cols = [0]
project_data.drop(project_data.columns[cols],axis=1,inplace=True)
```

In [23]:

```
y = project_data['project_is_approved'].values
project_data.drop(['project_is_approved'], axis=1, inplace=True)
```

In [24]:

```
project_data.tail(1)
```

Out[24]:

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

In [25]:

```
project_data.shape
```

Out[25]:

```
(50000, 11)
```

In [26]:

```
X=project_data
```

In [27]:

```
#saving processed data to csv file for future use
X.to_csv('DonorsChoose_processed.csv', encoding='utf-8', index=False)
```

In [28]:

```
#how to fill Nan values in a column- https://youtu.be/fCMrO_VzeL8
X['teacher_prefix'].fillna(value='Teacher', inplace=True)
```

2. Splitting Data

In [29]:

```
# 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

In [30]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.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 = vectorizer.transform(X_train['clean_essay'].values)
X_cv_essay_bow = vectorizer.transform(X_cv['clean_essay'].values)
X_test_essay_bow = vectorizer.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 [31]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.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(vectorizer.get_feature_names())
print("="*100)
```

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

3.3 one hot encoding the catogorical features: teacher_prefix

In [32]:

```
vectorizer = CountVectorizer(min_df=10)
vectorizer.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 = vectorizer.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
```

```
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 [33]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.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(vectorizer.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 [34]:

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.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 = vectorizer.transform(X_train['clean_title'].values)
X_cv_title_ohe = vectorizer.transform(X_cv['clean_title'].values)
X_test_title_ohe = vectorizer.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, 1675) (22445,)
(11055, 1675) (11055,)
(16500, 1675) (16500,)
====================================================================================================
```

3.6 one hot encoding the catogorical features: project_category

In [35]:

```
vectorizer.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 = vectorizer.transform(X_train['clean_categories'].values)
X_cv_category_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_category_ohe = vectorizer.transform(X_test['clean_categories'].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(vectorizer.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 [36]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_subcategory_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_subcategory_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_subcategory_ohe = vectorizer.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(vectorizer.get_feature_names())
print("="*100)
```

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

3.8 Normalizing the numerical features: Price

In [37]:

```
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_test_price_norm = normalizer.transform(X_test['price'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_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 [38]:

```
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))
X_cv_post_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.
reshape(-1,1))
X_test_post_norm =
normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

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

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

3.10 Normalizing the numerical features: resource summary

In [39]:

```
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))
X_cv_digit_norm = normalizer.transform(X_cv['Is_digit_present'].values.reshape(-1,1))
X_test_digit_norm = normalizer.transform(X_test['Is_digit_present'].values.reshape(-1,1))

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 [40]:

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

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

print("After vectorizations")
print(X_train_quantity_norm.shape, y_train.shape)
print(X_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,)
====================================================================================================
```

In [41]:

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

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

Concatenation of all the vectorized data

In [42]:

```
# 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)
print("="*100)
```

```
Final Data matrix
```

```
(16500, 6776) (16500,)
==========================================================================================================
```

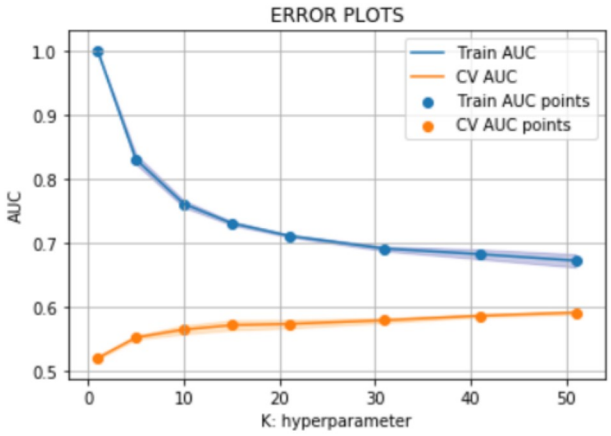
Applying KNN brute force on BOW

In [44]:

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


neigh = KNeighborsClassifier()
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51]}
clf = GridSearchCV(neigh, 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 [45]:

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

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

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')


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



From the above graph we can see that for the value of k less than 13, it is over fitting the data and for the value of k more than 13, there is not that much change in the curve of Train and CV and it is also underfitting the data. So the best value of k will be 13 for this

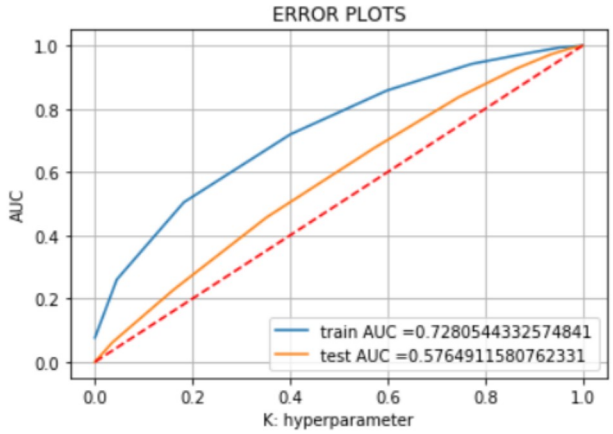
In [50]:

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

```
neigh = KNeighborsClassifier(n_neighbors=best_k, algorithm='brute')
neigh.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 = neigh.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 = neigh.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 [51]:

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



In [52]:

```
# 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 [54]:

```
from sklearn.metrics import confusion_matrix
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) 0.24466285391992368 for threshold 0.765
```

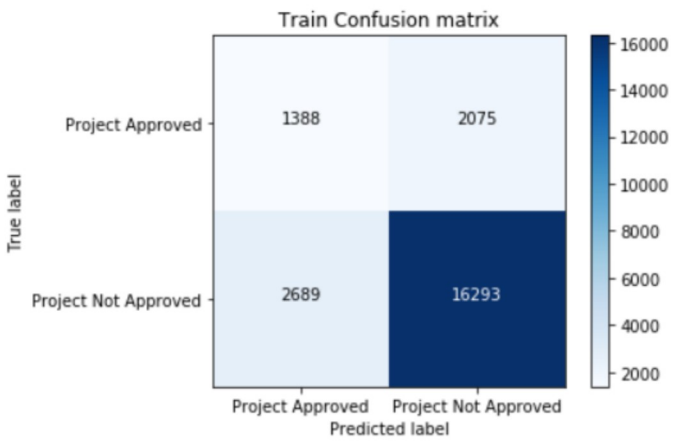
In [55]:

```
#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]:

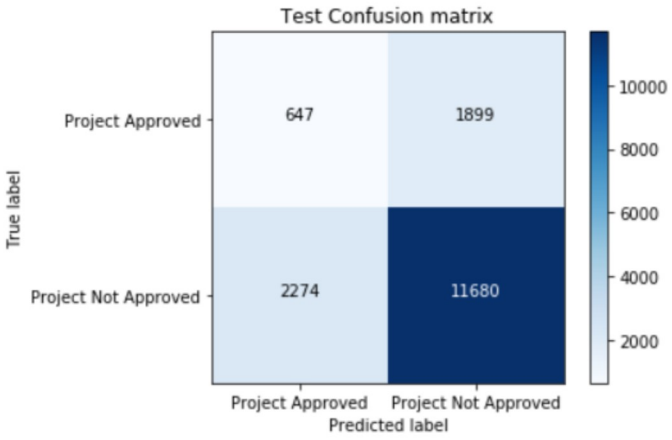
```
import itertools
plot_confusion_matrix(train_conf, classes=class_names,
                      title='Train Confusion matrix')
```

```
Confusion matrix, without normalization
[[ 1388  2075]
 [ 2689 16293]]
```



In [58]:

```
Confusion matrix, without normalization
[[  647  1899]
 [ 2274 11680]]
```



Conclusion

On applying KNN BOW on DonorsChoose dataset with k value as 13 we got the AUC value of 0.57 on test data

In [2]:

```
#code copied from -http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyper Parameter (k)", "AUC"]
x.add_row(["BOW", "Brute", 13, 0.57])
x.add_row(["TFIDF", "Brute", 11, 0.53])
x.add_row(["TFIDF--Top 2000 feature", "Brute", 11, 0.51])
x.add_row(["Avg_W2V", "Brute", 10, 0.51])
x.add_row(["Tfidf_W2V", "Brute", 12, 0.53])
print(x)
```

```
+------------------------+-------+--------------------+------+
|       Vectorizer       | Model | Hyper Parameter (k) | AUC  |
+------------------------+-------+--------------------+------+
|          BOW           | Brute |         13         | 0.57 |
|         TFIDF          | Brute |         11         | 0.53 |
| TFIDF--Top 2000 feature | Brute |         11         | 0.51 |
|        Avg_W2V         | Brute |         10         | 0.51 |
|       Tfidf_W2V        | Brute |         12         | 0.53 |
+------------------------+-------+--------------------+------+
```