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`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy`

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_3:__ "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:

- __project_essay_1:__ "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."

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.

Importing Required Libraries

In [1]:

```
# importing required libraries

%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.model_selection import GridSearchCV
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

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 sklearn.model_selection import GridSearchCV
```

```
C:\Users\parik\AppData\Local\Continuum\anaconda3\lib\site-packages\smart_open\ssh.py:34:
UserWarning: paramiko missing, opening SSH/SCP/SFTP paths will be disabled.  `pip install
paramiko` to suppress
  warnings.warn('paramiko missing, opening SSH/SCP/SFTP paths will be disabled.  `pip install
paramiko` to suppress')
```

1.1 Reading Data

In [2]:

```
# Reading data from project and resources data file

project_data = pd.read_csv('E:/Applied AI/Donors choose
dataset/DC_data/Assignments_DonorsChoose_2018/train_data.csv')
resource_data = pd.read_csv('E:/Applied AI/Donors choose
dataset/DC_data/Assignments_DonorsChoose_2018/resources.csv')
```

In [3]:

```
# Getting basic information about the data

print("Number of data points in Project_train data", project_data.shape)
print('-'*100)
print("The attributes of Project_train data :", project_data.columns.values)
print('='*100)
print("Number of data points in Resource_train data", resource_data.shape)
print('-'*100)
print("The attributes of Resource_train data :", resource_data.columns.values)
```

```
Number of data points in Project_train data (109248, 17)
----------------------------------------------------------------------------------------------------
The attributes of Project_train data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix'
'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
====================================================================================================

Number of data points in Resource_train data (1541272, 4)
----------------------------------------------------------------------------------------------------
The attributes of Resource_train data : ['id' 'description' 'quantity' 'price']
```

1.2 Data Pre-Processing

In [4]:

```
# Merge two column text dataframe:
# Merge 4 essays into one:
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)

# Merge Price information from resource data to project data
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')

# find how many digits are present in each project_resource_summary coloumn
summary = list(project_data['project_resource_summary'].values)
presence_of_numeric_data=[]
for i in summary:
    count = 0
    for j in i.split(' '):
        if j.isdigit():
            count+=1
    presence_of_numeric_data.append(count)

# Replace Text summary coloumn with new numerical coloumn presence_of_numeric_data
project_data['numerical_data_in_resource_summary'] = presence_of_numeric_data
project_data.drop(['project_resource_summary'], axis=1, inplace=True)

# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.columns)]

#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
project_data.drop('project_submitted_datetime', axis=1, inplace=True)
project_data.sort_values(by=['Date'], inplace=True)

# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
project_data = project_data[cols]

# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dropna.html
# Here we drop 3 rows where teacher_prefix is having np.nan value
project_data.dropna(axis=0,subset=['teacher_prefix'], inplace=True)
```

```
project_data.head(2)
```

Out[4]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cate
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5

1.2.1 Pre-Processing Essay Text

In [5]:

```
# printing some random essays.
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
```

```
I have been fortunate enough to use the Fairy Tale STEM kits in my classroom as well as the STEM j
ournals, which my students really enjoyed.  I would love to implement more of the Lakeshore STEM k
its in my classroom for the next school year as they provide excellent and engaging STEM
lessons.My students come from a variety of backgrounds, including language and socioeconomic statu
s.  Many of them don't have a lot of experience in science and engineering and these kits give me
the materials to provide these exciting opportunities for my students.Each month I try to do
several science or STEM/STEAM projects.  I would use the kits and robot to help guide my science i
nstruction in engaging and meaningful ways.  I can adapt the kits to my current language arts paci
ng guide where we already teach some of the material in the kits like tall tales (Paul Bunyan) or
Johnny Appleseed.  The following units will be taught in the next school year where I will
implement these kits: magnets, motion, sink vs. float, robots.  I often get to these units and don
't know If I am teaching the right way or using the right materials.    The kits will give me
additional ideas, strategies, and lessons to prepare my students in science.It is challenging to d
evelop high quality science activities.  These kits give me the materials I need to provide my
students with science activities that will go along with the curriculum in my classroom.  Although
I have some things (like magnets) in my classroom, I don't know how to use them effectively.  The
kits will provide me with the right amount of materials and show me how to use them in an
appropriate way.
==================================================
I teach high school English to students with learning and behavioral disabilities. My students all
vary in their ability level. However, the ultimate goal is to increase all students literacy level
s. This includes their reading, writing, and communication levels.I teach a really dynamic group o
f students. However, my students face a lot of challenges. My students all live in poverty and in
a dangerous neighborhood. Despite these challenges, I have students who have the the desire to def
eat these challenges. My students all have learning disabilities and currently all are performing
below grade level. My students are visual learners and will benefit from a classroom that fulfills
their preferred learning style.The materials I am requesting will allow my students to be prepared
for the classroom with the necessary supplies.  Too often I am challenged with students who come t
o school unprepared for class due to economic challenges.  I want my students to be able to focus
on learning and not how they will be able to get school supplies.  The supplies will last all year
.  Students will be able to complete written assignments and maintain a classroom journal.  The ch
art paper will be used to make learning more visual in class and to create posters to aid students
in their learning.  The students have access to a classroom printer.  The toner will be used to pr
int student work that is completed on the classroom Chromebooks.I want to try and remove all barri
ers for the students learning and create opportunities for learning. One of the biggest barriers i
s the students not having the resources to get pens, paper, and folders. My students will be able
to increase their literacy skills because of this project.
==================================================
```

In [6]:

```
# 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 [7]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
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',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'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 [8]:

```
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(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)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essays.append(sent.lower().strip())

# Adding preprocessed_essays coloumn to our data matrix

project_data['preprocessed_essays']=preprocessed_essays
```

```
100%|██████████████████████████████████████████████████████████████| 109245/109245
[01:35<00:00, 1147.73it/s]
```

In [9]:

```
# after preprocesing
preprocessed_essays[100]
```

Out[9]:

```
'a typical day campus exciting my students love learning always put smile face they big personalit
ies even bigger dedication learning they need hero someone willing change future every child
deserves champion adult never give understand power connection insists become best rita pierson we
school 610 low income students grades k 6 we eager bunch love learn we high expectations learning
low resources impact learning with new technology aim high rise occasion i would love incorporate
technology intervention time i empower students empowering students starts able give goals tools n
eed successful school my students use ipads foster love learning remediation enrichment students l
ow income families need engagement motivation succeed ipads bring closer achieving success
classroom real world my project make difference allowing students access programs reinforce classr
oom learning motivating stay focused sustained engagement'
```

1.2.2 Pre-Processing Project Title Text

In [10]:

```
from tqdm import tqdm
preprocessed_titles = []
# tqdm is for printing the status bar
for title in tqdm(project_data['project_title'].values):
    title = decontracted(title)
    title = title.replace('\\r', ' ')
    title = title.replace('\\"', ' ')
    title = title.replace('\\n', ' ')
    title = re.sub('[^A-Za-z0-9]+', ' ', title)
    # https://gist.github.com/sebleier/554280
    title = ' '.join(e for e in title.split() if e not in stopwords)
    preprocessed_titles.append(title.lower().strip())

# Adding preprocessed_titles coloumn to our data matrix

project_data['preprocessed_titles']=preprocessed_titles
preprocessed_titles[1000]
```

```
100%|██████████████████████████████████████████████████████████████| 109245/109245
[00:03<00:00, 31997.38it/s]
```

Out[10]:

```
'empowering students through art learning about then now'
```

1.2.3 Pre-Processing Project Grades

In [11]:

```
# Remove special characters from grades
from tqdm import tqdm
preprocessed_grade_categories = []
# tqdm is for printing the status bar
for categories in tqdm(project_data['project_grade_category'].values):
    categories = decontracted(categories)
    # https://gist.github.com/sebleier/554280
    categories = '_'.join(e for e in categories.split(' ') if e not in stopwords)
    categories = '_'.join(e for e in categories.split('-') if e not in stopwords)
    preprocessed_grade_categories.append(categories.lower().strip())

# Adding preprocessed_titles coloumn to our data matrix

project_data['preprocessed_grade_category']=preprocessed_grade_categories

project_data.head(5)
```

```
100%|██████████████████████████████████████████████████████████████| 109245/109245
[00:01<00:00, 55420.17it/s]
```

Out[11]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cate
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	2016-04-27 00:46:53	Grades PreK-2
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5

5 rows × 23 columns

1.2.4 preprocessing of `project_subject_categories`

In [12]:

```
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())

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

1.2.5 preprocessing of `project_subject_subcategories`

In [13]:

```
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())

project_data['clean_subcategories'] = sub_cat_list
project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)
```

In [14]:

```
# Drop all unnecessary featurs like project_grade_category, project_essay_1, etc.
project_data.drop(['project_grade_category'], axis=1, inplace=True)
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
project_data.drop(['essay'], axis=1, inplace=True)
```

In [15]:

```
project_data.head(5)
```

Out[15]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_title	teache
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Engineering STEAM into the Primary Classroom	53
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Sensory Tools for Focus	4
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	2016-04-27 00:46:53	Mobile Learning with a Mobile	10

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_title	teache
							Listening project	
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Flexible Seating for Flexible Learning	2
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Going Deep: The Art of Inner Thinking!	2

1.3 Sampling data for Naive Bayes Assignment

In [16]:

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

Out[16]:

```
1    92703
0    16542
Name: project_is_approved, dtype: int64
```

In [17]:

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

Out[17]:

```
1    92703
0    16542
Name: project_is_approved, dtype: int64
```

In [18]:

```
data.head(5)
```

Out[18]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_title	teache
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Engineering STEAM into the Primary Classroom	53
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Sensory Tools for Focus	4
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	2016-04-27 00:46:53	Mobile Learning with a Mobile Listening Center	10
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Flexible Seating for Flexible Learning	2
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27	Going Deep: The Art of	2

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_title	teache
						01:05:25	Inner Thinking	

In [19]:

```
# This list contains the feature names in the same order in which final data matrix is created

List_of_imp_features=[]
```

In [20]:

```
# Split the class label from data
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
X.head(1)
```

Out[20]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_title	teache
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Engineering STEAM into the Primary Classroom	53

In [21]:

```
X.shape
```

Out[21]:

```
(109245, 16)
```

2.1 Splitting data into Train and cross validation(or test): Stratified Sampling And Simple Upsampling

In [22]:

```
# train test split
# Not using CV data as it will be done by the GridsearchCV internally
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)
```

In [23]:

```
X_train['project_is_approved']=y_train

print(X_train.project_is_approved.value_counts())
Accepted, Rejected = X_train.project_is_approved.value_counts()
```

```
1    62111
0    11083
Name: project_is_approved, dtype: int64
```

In [24]:

```
y_train = X_train.project_is_approved
X_train = X_train.drop('project_is_approved', axis=1)
X_train.shape
```

Out[24]:

```
(73194, 16)
```

(73194, 16)

2.2 Make Data Model Ready:

2.2.1 Encoding numerical, categorical features

2.2.1.1 Encoding School State

In [25]:

```
# Encoding School State

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

for i in features:
    List_of_imp_features.append(i)


# 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
(73194, 51) (73194,)
(36051, 51) (36051,)
['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']
====================================================================================================
```

2.2.1.2 Encoding Teacher Prefix

In [26]:

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

for i in features:
    List_of_imp_features.append(i)


# 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(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(73194, 5) (73194,)
(36051, 5) (36051,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
==========================================================================================================
```

2.2.1.3 Encoding preprocessed_grade_category

In [27]:

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

for i in features:
    List_of_imp_features.append(i)


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

X_test_grade_ohe = vectorizer.transform(X_test['preprocessed_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
(73194, 4) (73194,)
(36051, 4) (36051,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
==========================================================================================================
```

2.2.1.4 Encoding numerical feature Price

In [48]:

```
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))
List_of_imp_features.append('price')
X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(1,-1))
#X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1))

X_train_price_norm = X_train_price_norm.reshape(-1,1)
X_test_price_norm = X_test_price_norm.reshape(-1,1)


print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_train_price_norm)
#print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
```

```
(73194, 1) (73194,)
[[4.25607639e-03]
 [2.48972645e-03]
 [3.76866176e-04]
 ...
 [3.74753785e-03]
 [4.88196998e-05]
 [2.19829475e-03]]
(36051, 1) (36051,)
====================================================================================================
```

2.2.1.5 Encoding numeric feature Quantity

In [44]:

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

```
[[0.00118801]
 [0.00415805]
 [0.00047521]
 ...
 [0.0001188 ]
 [0.00594007]
 [0.00142562]]
After vectorizations
(73194, 1) (73194,)
(36051, 1) (36051,)
====================================================================================================
```

2.2.1.6 Encoding numeric feature teacher_number_of_previously_posted_projects

In [45]:

```
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['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
List_of_imp_features.append('teacher_number_of_previously_posted_projects')
X_train_teacher_number_of_previously_posted_projects_norm =
normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
#X_cv_teacher_number_of_previously_posted_projects_norm =
normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
X_test_teacher_number_of_previously_posted_projects_norm =
normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
X_train_teacher_number_of_previously_posted_projects_norm =
```

```
X_train_teacher_number_of_previously_posted_projects_norm.reshape(-1,1)
X_test_teacher_number_of_previously_posted_projects_norm =
X_test_teacher_number_of_previously_posted_projects_norm.reshape(-1,1)

print(X_test_teacher_number_of_previously_posted_projects_norm)
print("After vectorizations")
print(X_train_teacher_number_of_previously_posted_projects_norm.shape, y_train.shape)
#print(X_cv_teacher_number_of_previously_posted_projects_norm.shape, y_cv.shape)
print(X_test_teacher_number_of_previously_posted_projects_norm.shape, y_test.shape)
print("="*100)
```

```
[[0.00017734]
 [0.00035468]
 [0.        ]
 ...
 [0.00017734]
 [0.00070935]
 [0.00461078]]
After vectorizations
(73194, 1) (73194,)
(36051, 1) (36051,)
====================================================================================================
```

2.2.1.7 Encoding numeric feature numerical_data_in_resource_summary

In [46]:

```
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['numerical_data_in_resource_summary'].values.reshape(1,-1))
List_of_imp_features.append('numerical_data_in_resource_summary')
X_train_numerical_data_in_resource_summary_norm =
normalizer.transform(X_train['numerical_data_in_resource_summary'].values.reshape(1,-1))
#X_cv_numerical_data_in_resource_summary_norm =
normalizer.transform(X_cv['numerical_data_in_resource_summary'].values.reshape(1,-1))
X_test_numerical_data_in_resource_summary_norm =
normalizer.transform(X_test['numerical_data_in_resource_summary'].values.reshape(1,-1))

X_train_numerical_data_in_resource_summary_norm = X_train_numerical_data_in_resource_summary_norm.
reshape(-1,1)
X_test_numerical_data_in_resource_summary_norm = X_test_numerical_data_in_resource_summary_norm.re
shape(-1,1)

print(X_test_numerical_data_in_resource_summary_norm)
print("After vectorizations")
print(X_train_numerical_data_in_resource_summary_norm.shape, y_train.shape)
#print(X_cv_numerical_data_in_resource_summary_norm.shape, y_cv.shape)
print(X_test_numerical_data_in_resource_summary_norm.shape, y_test.shape)
print("="*100)
```

```
[[0.        ]
 [0.01176471]
 [0.        ]
 ...
 [0.        ]
 [0.        ]
 [0.        ]]
After vectorizations
(73194, 1) (73194,)
(36051, 1) (36051,)
====================================================================================================
```

In [34]:

```
print(List_of_imp_features)
```

```
['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', 'dr', 'mr', 'mrs', 'ms', 'teacher', 'grades_3_5', 'grades_6_8', 'grades_9_12',
'grades_prek_2', 'price', 'price', 'quantity', 'teacher_number_of_previously_posted_projects', 'te
acher_number_of_previously_posted_projects', 'numerical_data_in_resource_summary']
```

2.3 Appling Naive Bayes on different kind of featurizations

In [136]:

```
#Setting the values for alpha
import math
a=[]
for i in range(-4,5):
    a.append(10**i)

log_alpha = []
for i in a:
    log_alpha.append(math.log(i,10))
```

In [120]:

```
# Define Functions for Train NB model, Test NB Model and Plot the graphs for different featurizati
ons

import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score


def train_MNB(X_tr,y_train):

    train_score=[]
    test_score=[]

    #create new a knn model


    nb = MultinomialNB(class_prior = [0.5, 0.5])
    #create a dictionary of all values we want to test for n_neighbors
    param_grid = {'alpha': a}
#use gridsearch to test all values for n_neighbors
    NB_gscv = GridSearchCV(nb, param_grid, cv=10, scoring='roc_auc', return_train_score=True)
    NB_gscv.fit(X_tr, y_train)
    print(NB_gscv.best_params_)

    print(NB_gscv.cv_results_.keys())
    for key, value in NB_gscv.cv_results_.items():
        if key == "mean_train_score":
            train_score = value
        if key == "mean_test_score":
            test_score = value


    plt.plot(log_alpha, train_score, label='Train AUC')
    plt.plot(log_alpha, test_score, label='CV AUC')

    plt.scatter(log_alpha, train_score, label='Train AUC points')
    plt.scatter(log_alpha, test_score, label='CV AUC points')

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

# Test the model with optimal alpha found out using training data. Plot FPR vs TPR(ROC curves) for
training and testing data
```

```
def Testing_MNB(X_tr,X_te,best_a):

    y_train_pred=[]
    y_test_pred=[]

    from sklearn.metrics import roc_curve, auc

    nb = MultinomialNB(alpha=best_a,class_prior = [0.5, 0.5])
    nb=nb.fit(X_tr, y_train)

    y_train_pred_raw = nb.predict_proba(X_tr)

    y_test_pred_raw = nb.predict_proba(X_te)


    for i in y_train_pred_raw:
        y_train_pred.append(i[1])
    for i in y_test_pred_raw:
        y_test_pred.append(i[1])




    train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
    test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

    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.legend()
    plt.xlabel("tpr")
    plt.ylabel("fpr")
    plt.title("ERROR PLOTS")
    plt.grid()
    plt.show()
    return train_fpr,train_tpr,tr_thresholds,y_train_pred,y_test_pred

# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

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

2.3.1 Applying Naive Bayes on BOW encoding essay, and project_title

2.3.1.1 Encoding preprocessed_essays BoW

In [37]:

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

print("="*100)


vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=10000)
vectorizer.fit(X_train['preprocessed_essays'].values) # fit has to happen only on train data
features = vectorizer.get_feature_names()

for i in features:
    List_of_imp_features.append(i)
```

```
# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['preprocessed_essays'].values)
#X_cv_essay_bow = vectorizer.transform(X_cv['preprocessed_essays'].values)
X_test_essay_bow = vectorizer.transform(X_test['preprocessed_essays'].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)
print("="*100)


# print("YOU SHOULD NOT DO SOMETHING LIKE THIS")
# vectorizer = CountVectorizer()
# x_train_bow = vectorizer.fit_transform(X_train['essay'].values)
# x_cv_bow = vectorizer.fit_transform(X_cv['essay'].values)
# x_test_bow = vectorizer.fit_transform(X_test['essay'].values)

# print(x_train_bow.shape, y_train.shape)
# print(x_cv_bow.shape, y_cv.shape)
# print(x_test_bow.shape, y_test.shape)

print("NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME")
```

```
(73194, 16) (73194,)
(36051, 16) (36051,)
==================================================================================================

After vectorizations
(73194, 10000) (73194,)
(36051, 10000) (36051,)
==================================================================================================

NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

2.3.1.2 Encoding preprocessed_titles BoW

In [39]:

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=10000)
vectorizer.fit(X_train['preprocessed_titles'].values) # fit has to happen only on train data

features = vectorizer.get_feature_names()

for i in features:
    List_of_imp_features.append(i)



# we use the fitted CountVectorizer to convert the text to vector
X_train_titles_bow = vectorizer.transform(X_train['preprocessed_titles'].values)
#X_cv_titles_bow = vectorizer.transform(X_cv['preprocessed_titles'].values)
X_test_titles_bow = vectorizer.transform(X_test['preprocessed_titles'].values)

print("After vectorizations")
print(X_train_titles_bow.shape, y_train.shape)
#print(X_cv_titles_bow.shape, y_cv.shape)
print(X_test_titles_bow.shape, y_test.shape)
print("="*100)


# print("YOU SHOULD NOT DO SOMETHING LIKE THIS")
# vectorizer = CountVectorizer()
# x_train_bow = vectorizer.fit_transform(X_train['essay'].values)
# x_cv_bow = vectorizer.fit_transform(X_cv['essay'].values)
# x_test_bow = vectorizer.fit_transform(X_test['essay'].values)

# print(x_train_bow.shape, y_train.shape)
# print(x_cv_bow.shape, y_cv.shape)
# print(x_test_bow.shape, y_test.shape)

print("NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME")
```

```
After vectorizations
(73194, 5898) (73194,)
(36051, 5898) (36051,)
==========================================================================================================

NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

2.3.1.3 Merge all the features and obtain final data matrix

In [121]:

```
# Merge all the features:
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
# Matrix are merged in such a way that the order is preserved in # Matrix are merged in such a way
that the order is preserved in List_of_imp_features list


from scipy.sparse import hstack
X_tr = hstack((X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe,
X_train_price_norm,X_train_quantity_norm,
X_train_teacher_number_of_previously_posted_projects_norm,X_train_numerical_data_in_resource_summar
_norm,X_train_essay_bow,X_train_titles_bow )).tocsr()
#X_cr = hstack((X_cv_titles_bow,X_cv_essay_bow, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe,
X_cv_price_norm, X_cv_numerical_data_in_resource_summary_norm,
X_cv_teacher_number_of_previously_posted_projects_norm, X_cv_quantity_norm)).tocsr()
X_te = hstack((X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe,
X_test_price_norm,X_test_quantity_norm, X_test_teacher_number_of_previously_posted_projects_norm,X
_test_numerical_data_in_resource_summary_norm,X_test_essay_bow,X_test_titles_bow )).tocsr()

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
(73194, 15962) (73194,)
(36051, 15962) (36051,)
==========================================================================================================
```

In [122]:

```
print(len(List_of_imp_features))
```

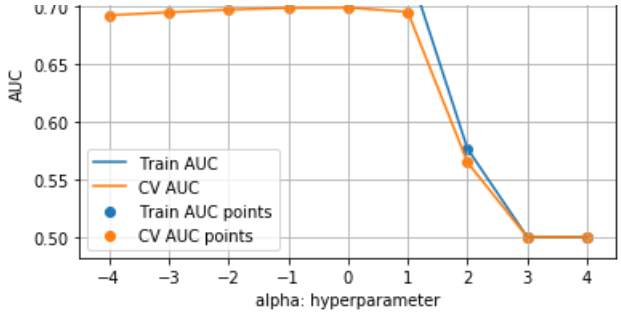
```
21867
```

2.3.1.4 Training the data model and find best hyperparameter using ROC-AUC

In [123]:

```
# Call train_KNN function on above data

train_MNB(X_tr,y_train)
```

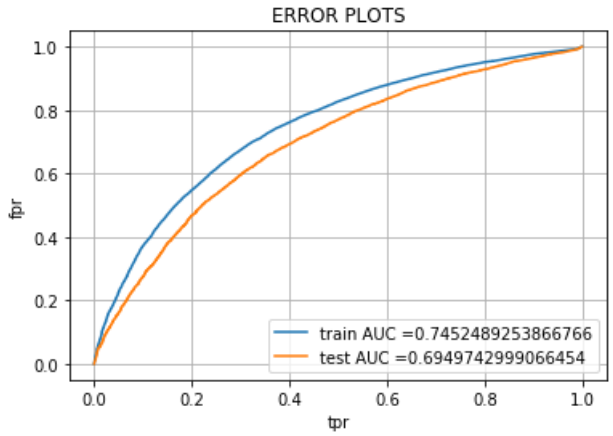
```
{'alpha': 1}
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time', 'param_alpha', 'p
arams', 'split0_test_score', 'split1_test_score', 'split2_test_score', 'split3_test_score',
'split4_test_score', 'split5_test_score', 'split6_test_score', 'split7_test_score',
'split8_test_score', 'split9_test_score', 'mean_test_score', 'std_test_score', 'rank_test_score',
'split0_train_score', 'split1_train_score', 'split2_train_score', 'split3_train_score',
'split4_train_score', 'split5_train_score', 'split6_train_score', 'split7_train_score',
'split8_train_score', 'split9_train_score', 'mean_train_score', 'std_train_score'])
```



2.3.1.5 Testing the performance of the model on test data, plotting ROC Curves

In [124]:

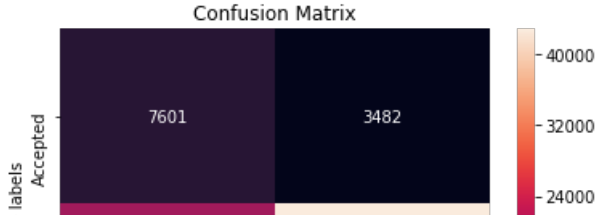
```
# Call test_KNN for K obtained by training the data

best_a=1
train_fpr,train_tpr,tr_thresholds,y_train_pred,y_test_pred=Testing_MNB(X_tr,X_te,best_a)
```

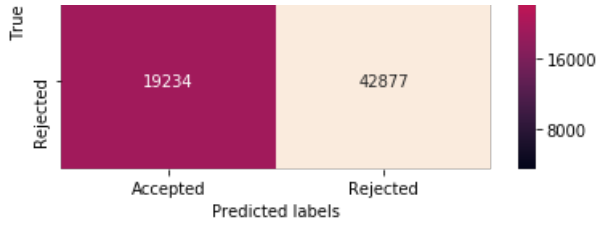


In [125]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")

ax= plt.subplot()
cm=confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
print(cm)
sns.heatmap(cm, annot=True, ax = ax,fmt='d'); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['Accepted', 'Rejected']); ax.yaxis.set_ticklabels(['Accepted', 'Rejected'
]);
```

```
====================================================================================================

the maximum value of tpr*(1-fpr) 0.4734447106759629 for threshold 0.447
Train confusion matrix
[[ 7601  3482]
 [19234 42877]]
```

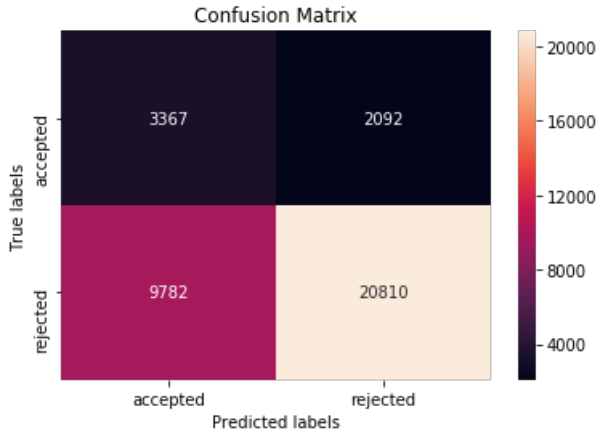




In [126]:

```
print("Test confusion matrix")


cm_test = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
print(cm_test)
ax= plt.subplot()
sns.heatmap(cm_test, annot=True, ax = ax,fmt='d'); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['accepted', 'rejected']); ax.yaxis.set_ticklabels(['accepted', 'rejected'
]);
```

```
Test confusion matrix
[[ 3367  2092]
 [ 9782 20810]]
```



Print Top 20 features with Bag of Words Featurization

In [127]:

```
nb = MultinomialNB()
nb.fit(X_tr, y_train)
neg_class_prob_values = nb.feature_log_prob_[0, :]
pos_class_prob_values = nb.feature_log_prob_[1, :]

neg_class_prob_values=sorted(neg_class_prob_values, reverse=True)
pos_class_prob_values=sorted(pos_class_prob_values, reverse=True)

print(neg_class_prob_values[:20])
print(pos_class_prob_values[:20])
print("-"*100)
neg_class_prob_feature_indices = nb.feature_log_prob_[0, :].argsort()
pos_class_prob_feature_indices = nb.feature_log_prob_[1, :].argsort()


print(neg_class_prob_feature_indices[len(neg_class_prob_feature_indices)-1:len(neg_class_prob_featu
re_indices)-21:-1])
print(pos_class_prob_feature_indices[len(pos_class_prob_feature_indices)-1:len(pos_class_prob_featu
re_indices)-21:-1])
print("-"*100)
top_20_neg_features=[]
```

```
for i in range((len(neg_class_prob_feature_indices)-1),(len(neg_class_prob_feature_indices)-21),-1)
:
    index = neg_class_prob_feature_indices[i]
    top_20_neg_features.append(List_of_imp_features[index])

top_20_pos_features=[]

for i in range((len(pos_class_prob_feature_indices)-1),(len(pos_class_prob_feature_indices)-21),-1)
:
    index = pos_class_prob_feature_indices[i]
    top_20_pos_features.append(List_of_imp_features[index])

print(top_20_neg_features)
print(top_20_pos_features)
print("-"*100)
```

```
[-3.3447012838366845, -4.437931428188566, -4.757200463495089, -4.808033031918269, -
4.92448059349605, -5.104606759726568, -5.115065175569409, -5.1418587877776645, -5.150471281204663,
-5.162233279366973, -5.17923270801788, -5.3148858922298565, -5.355047291246448, -
5.391371260619858, -5.462090738661926, -5.5148645461418955, -5.6530020735539495, -
5.686931014610771, -5.722761152493572, -5.72859788342061]
[-3.327294283893213, -4.4711246440053305, -4.784735658861376, -4.838013780066371, -
4.861694809193336, -5.090700927379986, -5.128775098023322, -5.1335608835095154, -5.16174630932224,
-5.172485764993146, -5.205399590655821, -5.346386131767806, -5.362766240604831, -
5.400335470213474, -5.475229779530904, -5.47634301869981, -5.477986839156689, -5.541756648101302,
-5.635771687923009, -5.652923004983217]
----------------------------------------------------------------------------------------------------
[7897 7083 4544 5461 1430 5780 4428 8911 3689 8777 5505 5595 5075 9619
 5617 9825 1609 4890 6728  219]
[7897 7083 5461 4544 1430 8777 8911 5780 5505 4428 3689 5075 5595 9619
 9825 6728 5617 9378 4890 1998]
----------------------------------------------------------------------------------------------------
['student work', 'scholastic', 'learners want', 'musicians', 'classified', 'north carolina',
'league', 'these tools', 'held', 'thanks', 'my special', 'name', 'manipulatives', 'ways the', 'nec
essities', 'word work', 'combine', 'loud', 'readily available', 'ability read']
['student work', 'scholastic', 'musicians', 'learners want', 'classified', 'thanks', 'these
tools', 'north carolina', 'my special', 'league', 'held', 'manipulatives', 'name', 'ways the', 'wo
rd work', 'readily available', 'necessities', 'us we', 'loud', 'data']
----------------------------------------------------------------------------------------------------
```

2.3.2 Applying KNN brute force on TFIDF encoding eassay, and project_title

2.3.2.2 Encoding preprocessed_essays TFIDF

In [69]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer.transform(X_train['preprocessed_essays'].values)
#X_cv_essay_tfidf = vectorizer.transform(X_cv['preprocessed_essays'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['preprocessed_essays'].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)
print("="*100)
```

```
After vectorizations
(73194, 14222) (73194,)
(36051, 14222) (36051,)
====================================================================================================
```

2.3.2.1 Encoding preprocessed_titles TFIDF

In [70]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)

#vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=10000)
vectorizer.fit(X_train['preprocessed_titles'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_titles_tfidf = vectorizer.transform(X_train['preprocessed_titles'].values)
#X_cv_titles_tfidf = vectorizer.transform(X_cv['preprocessed_titles'].values)
X_test_titles_tfidf = vectorizer.transform(X_test['preprocessed_titles'].values)

print("After vectorizations")
print(X_train_titles_tfidf.shape, y_train.shape)
#print(X_cv_titles_tfidf.shape, y_cv.shape)
print(X_test_titles_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(73194, 2632) (73194,)
(36051, 2632) (36051,)
====================================================================================================
```

2.3.2.3 Merge all the features and obtain final data matrix

In [128]:

```
from scipy.sparse import hstack
X_tr = hstack((X_train_titles_tfidf,X_train_essay_tfidf, X_train_state_ohe, X_train_teacher_ohe,
X_train_grade_ohe, X_train_price_norm, X_train_numerical_data_in_resource_summary_norm,
X_train_teacher_number_of_previously_posted_projects_norm, X_train_quantity_norm)).tocsr()
#X_cr = hstack((X_cv_titles_tfidf,X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_ohe,
X_cv_grade_ohe, X_cv_price_norm, X_cv_numerical_data_in_resource_summary_norm,
X_cv_teacher_number_of_previously_posted_projects_norm, X_cv_quantity_norm)).tocsr()
X_te = hstack((X_test_titles_tfidf,X_test_essay_tfidf, X_test_state_ohe, X_test_teacher_ohe, X_test
_grade_ohe, X_test_price_norm,X_test_numerical_data_in_resource_summary_norm,
X_test_teacher_number_of_previously_posted_projects_norm, X_test_quantity_norm)).tocsr()

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)
```

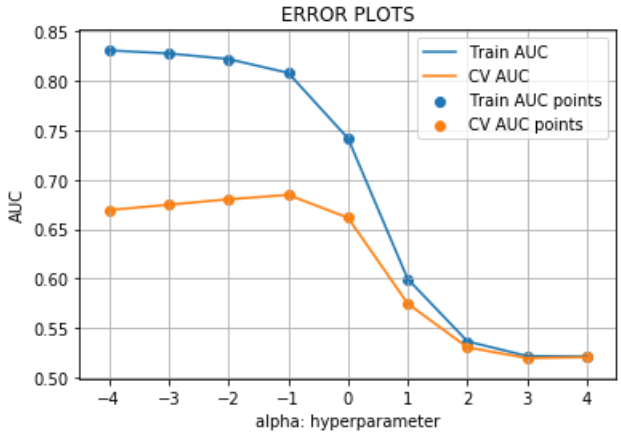
```
Final Data matrix
(73194, 16918) (73194,)
(36051, 16918) (36051,)
====================================================================================================
```

2.3.2.4 Training the data model and find best hyperparameter using ROC-AUC

In [129]:

```
# Call train_KNN function on above data

train_MNB(X_tr,y_train)
```

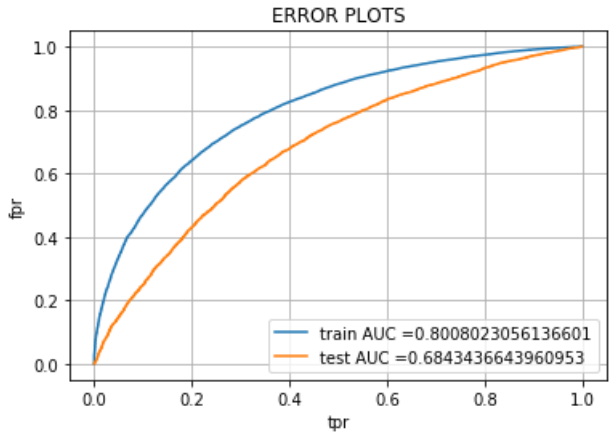
```
{'alpha': 0.1}
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time', 'param_alpha', 'p
arams', 'split0_test_score', 'split1_test_score', 'split2_test_score', 'split3_test_score',
'split4_test_score', 'split5_test_score', 'split6_test_score', 'split7_test_score',
'split8_test_score', 'split9_test_score', 'mean_test_score', 'std_test_score', 'rank_test_score',
'split0_train_score', 'split1_train_score', 'split2_train_score', 'split3_train_score',
'split4_train_score', 'split5_train_score', 'split6_train_score', 'split7_train_score',
'split8_train_score', 'split9_train_score', 'mean_train_score', 'std_train_score'])
```



2.3.2.5 Testing the performance of the model on test data, plotting ROC Curves

In [132]:

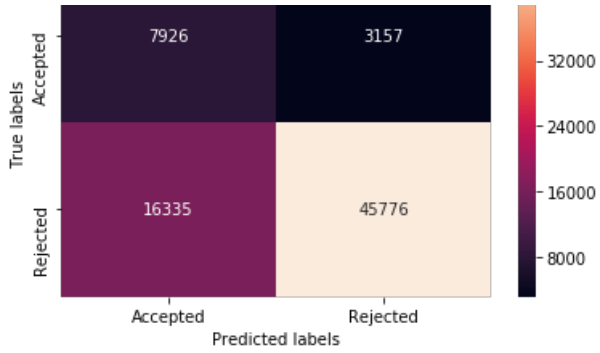
```
# Call test_KNN for K obtained by training the data

best_a=0.1
train_fpr,train_tpr,tr_thresholds,y_train_pred,y_test_pred=Testing_MNB(X_tr,X_te,best_a)
```



In [134]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")

ax= plt.subplot()
cm=confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
print(cm)
sns.heatmap(cm, annot=True, ax = ax,fmt='d'); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['Accepted', 'Rejected']); ax.yaxis.set_ticklabels(['Accepted', 'Rejected'
]);
```

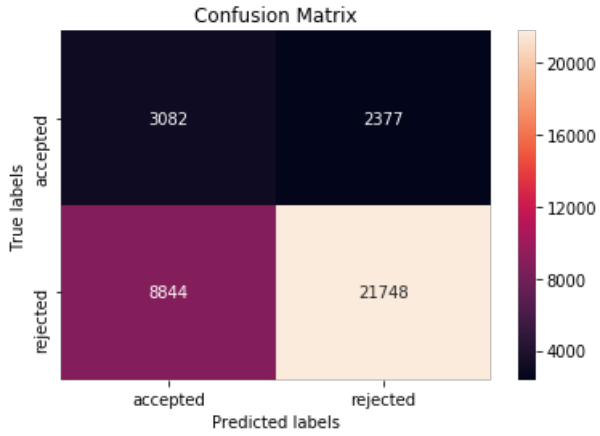
```
====================================================================================================

the maximum value of tpr*(1-fpr) 0.5270672768002808 for threshold 0.485
Train confusion matrix
[[ 7926  3157]
 [16335 45776]]
```



In [135]:

```
print("Test confusion matrix")


cm_test = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
print(cm_test)
ax= plt.subplot()
sns.heatmap(cm_test, annot=True, ax = ax,fmt='d'); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['accepted', 'rejected']); ax.yaxis.set_ticklabels(['accepted', 'rejected'
]);
```

```
Test confusion matrix
[[ 3082  2377]
 [ 8844 21748]]
```



2.4 Summary of Naive Bayes

In [133]:

```
# To summarize the results:
# summary table in jupyter notebook
# http://zetcode.com/python/prettytable/
# https://stackoverflow.com/questions/35160256/how-do-i-output-lists-as-a-table-in-jupyter-noteboo
k

from prettytable import PrettyTable

x = PrettyTable(header_color='\033[40m')

x.field_names = ["Vectorizer", "Model", "Hyperparameter", "Train_AUC","Test_AUC"]

x.add_row(["Bag of Words", "Naive Bayes",1.0 ,0.74, 0.69])
x.add_row(["TF-IDF", "Naive Bayes",0.1 , 0.80,0.68])

print(x)
```

Vectorizer	Model	Hyperparameter	Train_AUC	Test_AUC
Bag of Words	Naive Bayes	1.0	0.74	0.69
TF-IDF	Naive Bayes	0.1	0.8	0.68