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` • `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*

Feature	Description
`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."
- __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.

In [59]:

```
%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 TfidfVectorizer
```

```
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
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
from tqdm import tqdm
import os
import chart_studio.plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
from scipy.sparse import hstack,vstack
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import model_selection
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
from prettytable import PrettyTable
from sklearn.naive_bayes import MultinomialNB
from sklearn.preprocessing import Normalizer
import pdb
```

1.1 Reading Data

In [60]:

```
Project_data = pd.read_csv('train_data.csv')
Resource_data = pd.read_csv('resources.csv')
print(Project_data.shape)
print(Resource_data.shape)
```

```
(109248, 17)
(1541272, 4)
```

In [61]:

```
# 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]
Project_data.head(2)
```

Out[61]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_
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	

Splitting data into Train and cross validation(or test): Stratified Sampling

In [62]:

```
y = Project_data['project_is_approved'].values
Project_data.drop(['project_is_approved'], axis=1, inplace=True)
n_z = len(Project_data)
y_z = np.zeros(n_z, dtype=np.int32)

X = Project_data
# train test split
X_train, X_Test, y_train, y_Test = train_test_split(X, y, test_size=0.33, random_state=0, stratify=
y_z)
print('Shape of X_train: ',X_train.shape)
print('Shape of y_train: ',y_train.shape)
print('Shape of X_Test: ',X_Test.shape)
print('Shape of y_Test: ',y_Test.shape)
```

```
Shape of X_train:  (73196, 16)
Shape of y_train:  (73196,)
Shape of X_Test:  (36052, 16)
Shape of y_Test:  (36052,)
```

1.2 preprocessing of `project_subject_categories`

In [63]:

```
# 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
catogories_train = list(X_train['project_subject_categories'].values)
cat_list = []
for i in catogories_train:
    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())

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

from collections import Counter
my_counter = Counter()
for word in X_train['clean_categories'].values:
    my_counter.update(word.split())

cat_dict = dict(my_counter)
sorted_cat_dict_train = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))

print(len(sorted_cat_dict_train))


catogories_Test = list(X_Test['project_subject_categories'].values)
cat_list = []
for i in catogories_Test:
    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())

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

9

1.3 preprocessing of `project_subject_subcategories`

In [64]:

```
# 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_catogories_train = list(X_train['project_subject_subcategories'].values)
sub_cat_list = []
for i in sub_catogories_train:
    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())

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

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in X_train['clean_subcategories'].values:
    my_counter.update(word.split())

sub_cat_dict = dict(my_counter)
sorted_sub_cat_dict_train = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))

print(len(sorted_sub_cat_dict_train))

sub_catogories_Test = list(X_Test['project_subject_subcategories'].values)
sub_cat_list = []
for i in sub_catogories_Test:
    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())

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

30

1.3 Text preprocessing

In [65]:

```
# merge two column text dataframe:
X_train["essay"] = X_train["project_essay_1"].map(str) +\
                        X_train["project_essay_2"].map(str) + \
                        X_train["project_essay_3"].map(str) + \
                        X_train["project_essay_4"].map(str)


X_Test["essay"] = X_Test["project_essay_1"].map(str) +\
                        X_Test["project_essay_2"].map(str) + \
                        X_Test["project_essay_3"].map(str) + \
                        X_Test["project_essay_4"].map(str)
```

In [66]:

```
# 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 [67]:

```
# 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 [68]:

```
# Combining all the above stundents
# tqdm is for printing the status bar

preprocessed_essays_train = []

preprocessed_essays_Test = []

for sentance in tqdm(X_train['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.lower() not in stopwords)
    preprocessed_essays_train.append(sent.lower().strip())


for sentance in tqdm(X_Test['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.lower() not in stopwords)
    preprocessed_essays_Test.append(sent.lower().strip())

print("Shape of preprocessed_essays_train after preprocessing",len(preprocessed_essays_train))

print("Shape of preprocessed_essays_Test after preprocessing",len(preprocessed_essays_Test))
# pdb.set_trace()
```

```
100%|████████████████████████████████████████████████████████████████| 73196/73196 [01:
33<00:00, 786.44it/s]
100%|████████████████████████████████████████████████████████████████| 36052/36052 [00:
45<00:00, 783.83it/s]
```

```
Shape of preprocessed_essays_train after preprocessing 73196
Shape of preprocessed_essays_Test after preprocessing 36052
```

In [69]:

```
word_count_essay_train = []
for a in tqdm(X_train["essay"]) :
    b = len(a.split())
    word_count_essay_train.append(b)

X_train["word_count_essay_train"] = word_count_essay_train


word_count_essay_Test = []
for a in tqdm(X_Test["essay"]) :
    b = len(a.split())
    word_count_essay_Test.append(b)

X_Test["word_count_essay_Test"] = word_count_essay_Test
```

```
100%|████████████████████████████████████████████████████████████████| 73196/73196
[00:02<00:00, 25283.75it/s]
100%|████████████████████████████████████████████████████████████████| 36052/36052
[00:01<00:00, 24618.12it/s]
```

1.4 Preprocessing of `project_title`

In [70]:

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



preprocessed_titles_Test = []
for sentance in tqdm(X_Test['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)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_titles_Test.append(sent.lower().strip())

print("Shape of preprocessed_titles_train after preprocessing",len(preprocessed_titles_train))

print("Shape of preprocessed_titles_Test after preprocessing",len(preprocessed_titles_Test))
```

```
100%|████████████████████████████████████████████████████████████████| 73196/73196
[00:04<00:00, 15464.99it/s]
100%|████████████████████████████████████████████████████████████████| 36052/36052
[00:02<00:00, 14493.03it/s]
```

```
Shape of preprocessed_titles_train after preprocessing 73196
Shape of preprocessed_titles_Test after preprocessing 36052
```

In [71]:

```
word_count_title_train = []
for a in tqdm(X_train["project_title"]) :
    b = len(a.split())
    word_count_title_train.append(b)

X_train["word_count_title_train"] = word_count_title_train


word_count_title_Test = []
for a in tqdm(X_Test["project_title"]) :
    b = len(a.split())
    word_count_title_Test.append(b)

X_Test["word_count_title_Test"] = word_count_title_Test
```

```
100%|████████████████████████████████████████████████████████████████| 73196/73196
[00:00<00:00, 380650.78it/s]
100%|████████████████████████████████████████████████████████████████| 36052/36052
[00:00<00:00, 431499.75it/s]
```

Make Data Model Ready: encoding numerical, categorical features

1.5 Preparing data for models

1.5.1 Vectorizing Categorical data

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

In [72]:

```
# we use count vectorizer to convert the values into one
vectorizer_cat = CountVectorizer(vocabulary=list(sorted_cat_dict_train.keys()), lowercase=False, b
```

```
inary=True)
vectorizer_cat.fit(X_train['clean_categories'].values)
categories_one_hot_train = vectorizer_cat.transform(X_train['clean_categories'].values)

categories_one_hot_Test = vectorizer_cat.transform(X_Test['clean_categories'].values)
print(vectorizer_cat.get_feature_names())
print("Shape of categories_one_hot_train matrix after one hot encodig ",categories_one_hot_train.s
hape)

print("Shape of categories_one_hot_Test matrix after one hot encodig ",categories_one_hot_Test.sha
pe)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of categories_one_hot_train matrix after one hot encodig  (73196, 9)
Shape of categories_one_hot_Test matrix after one hot encodig  (36052, 9)
```

In [73]:

```
# we use count vectorizer to convert the values into one
vectorizer_sub_cat = CountVectorizer(vocabulary=list(sorted_sub_cat_dict_train.keys()), lowercase=
False, binary=True)
sub_categories_one_hot_train =
vectorizer_sub_cat.fit_transform(X_train['clean_subcategories'].values)

sub_categories_one_hot_Test = vectorizer_sub_cat.transform(X_Test['clean_subcategories'].values)
print(vectorizer_sub_cat.get_feature_names())
print("Shape of sub_categories_one_hot_train matrix after one hot encodig
",sub_categories_one_hot_train.shape)

print("Shape of sub_categories_one_hot_Test matrix after one hot encodig
",sub_categories_one_hot_Test.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Civics_Government', '
Extracurricular', 'ForeignLanguages', 'Warmth', 'Care_Hunger', 'NutritionEducation',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of sub_categories_one_hot_train matrix after one hot encodig  (73196, 30)
Shape of sub_categories_one_hot_Test matrix after one hot encodig  (36052, 30)
```

School State

In [74]:

```
sch1_catogories = list(X_train['school_state'].values)
school_list = []
for sent in sch1_catogories:
    school_list.append(sent.lower().strip())
X_train['school_categories'] = school_list
X_train.drop(['school_state'], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter_sch = Counter()
for word in X_train['school_categories'].values:
    my_counter_sch.update(word.split())

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
sch_dict = dict(my_counter_sch)
sorted_sch_dict = dict(sorted(sch_dict.items(), key=lambda kv: kv[1]))

vectorizer_sch = CountVectorizer(vocabulary=list(sorted_sch_dict.keys()), lowercase=False, binary=
True)
vectorizer_sch.fit(X_train['school_categories'].values)
#print(vectorizer.get_feature_names())

sch_one_hot_train = vectorizer_sch.transform(X_train['school_categories'].values)
print("Shape of sch_one_hot_train matrix after one hot encodig ",sch_one_hot_train.shape)
#-------------------------------------------------------------------------------------------


sch1_catogories_Test = list(X_Test['school_state'].values)
```

```
school_list_Test = []
for sent in sch1_catogories_Test:
    school_list_Test.append(sent.lower().strip())
X_Test['school_categories'] = school_list_Test
X_Test.drop(['school_state'], axis=1, inplace=True)

sch_one_hot_Test = vectorizer_sch.transform(X_Test['school_categories'].values)

print("Shape of sch_one_hot_Test matrix after one hot encodig ",sch_one_hot_Test.shape)
```

```
Shape of sch_one_hot_train matrix after one hot encodig  (73196, 51)
Shape of sch_one_hot_Test matrix after one hot encodig  (36052, 51)
```

Prefix

In [75]:

```
# 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
prefix_catogories_train = list(X_train['teacher_prefix'].values)
prefix_list_train = []
for sent in prefix_catogories_train:
    sent = re.sub('[^A-Za-z0-9]+', ' ', str(sent))
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split())
    prefix_list_train.append(sent.lower().strip())
X_train['prefix_catogories'] = prefix_list_train
X_train.drop(['teacher_prefix'], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter_prefix_train = Counter()
for word in X_train['prefix_catogories'].values:
    my_counter_prefix_train.update(word.split())

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
prefix_dict_train = dict(my_counter_prefix_train)
sorted_prefix_dict_train = dict(sorted(prefix_dict_train.items(), key=lambda kv: kv[1]))

vectorizer_prefix = CountVectorizer(vocabulary=list(sorted_prefix_dict_train.keys()), lowercase=Fa
lse, binary=True)
vectorizer_prefix.fit(X_train['prefix_catogories'].values)
#print(vectorizer.get_feature_names())

prefix_one_hot_train = vectorizer_prefix.transform(X_train['prefix_catogories'].values)
print("Shape of prefix_one_hot_train matrix after one hot encodig ",prefix_one_hot_train.shape)


#-------------------------------------------------------------------------------------------

prefix_catogories_Test = list(X_Test['teacher_prefix'].values)
prefix_list_Test = []
for sent in prefix_catogories_Test:
    sent = re.sub('[^A-Za-z0-9]+', ' ', str(sent))
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split())
    prefix_list_Test.append(sent.lower().strip())
X_Test['prefix_catogories'] = prefix_list_Test
X_Test.drop(['teacher_prefix'], axis=1, inplace=True)

prefix_one_hot_Test = vectorizer_prefix.transform(X_Test['prefix_catogories'])

print("Shape of prefix_one_hot_Test matrix after one hot encodig ",prefix_one_hot_Test.shape)
```

```
Shape of prefix_one_hot_train matrix after one hot encodig  (73196, 6)
Shape of prefix_one_hot_Test matrix after one hot encodig  (36052, 6)
```

project_grade_category

In [76]:

```
# 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
grade_catogories_train = list(X_train['project_grade_category'].values)
grade_list_train = []
for sent in grade_catogories_train:
    sent = sent.replace('-','_')
    sent = sent.replace(' ','_')
    # sent = re.sub('[^A-Za-z0-9]+', ' ', str(sent))
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split())
    grade_list_train.append(sent.lower().strip())

# temp = temp.replace('-','_')
X_train['new_grade_category'] = grade_list_train
X_train.drop(['project_grade_category'], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter_grade_train = Counter()
for word in X_train['new_grade_category'].values:
    my_counter_grade_train.update(word.split())

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
grade_dict_train = dict(my_counter_grade_train)
sorted_grade_dict_train = dict(sorted(grade_dict_train.items(), key=lambda kv: kv[1]))


vectorizer_grade = CountVectorizer(vocabulary=list(sorted_grade_dict_train.keys()), lowercase=Fals
e, binary=True)
vectorizer_grade.fit(X_train['new_grade_category'].values)
#print(vectorizer.get_feature_names())

grade_one_hot_train = vectorizer_grade.transform(X_train['new_grade_category'].values)
print("Shape of grade_one_hot_train matrix after one hot encodig ",grade_one_hot_train.shape)


#-------------------------------------------------------------------------------------------

grade_catogories_Test = list(X_Test['project_grade_category'].values)
grade_list_Test = []
for sent in grade_catogories_Test:
    sent = sent.replace('-','_')
    sent = sent.replace(' ','_')
    # sent = re.sub('[^A-Za-z0-9]+', ' ', str(sent))
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split())
    grade_list_Test.append(sent.lower().strip())

# temp = temp.replace('-','_')
X_Test['new_grade_category'] = grade_list_Test
X_Test.drop(['project_grade_category'], axis=1, inplace=True)

grade_one_hot_Test = vectorizer_grade.transform(X_Test['new_grade_category'].values)

print("Shape of grade_one_hot_Test matrix after one hot encodig ",grade_one_hot_Test.shape)
```

```
Shape of grade_one_hot_train matrix after one hot encodig  (73196, 4)
Shape of grade_one_hot_Test matrix after one hot encodig  (36052, 4)
```

1.5.2 Vectorizing Numerical features

Price and Quantity data

In [77]:

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

X_Test = pd.merge(X_Test, price_data, on='id', how='left')
```

In [78]:

```
price_norm = Normalizer(norm='l2', copy=False)
price_norm.fit(X_train['price'].values.reshape(1,-1))

price_norm.transform(X_train['price'].values.reshape(1,-1))

price_norm.transform(X_Test['price'].values.reshape(1,-1))

price_norm_train = (X_train['price'].values.reshape(-1,1))

price_norm_Test = (X_Test['price'].values.reshape(-1,1))

print("Shape of price_norm_train matrix after one hot encodig ",price_norm_train.shape)

print("Shape of price_norm_Test matrix after one hot encodig ",price_norm_Test.shape)
```

```
Shape of price_norm_train matrix after one hot encodig  (73196, 1)
Shape of price_norm_Test matrix after one hot encodig  (36052, 1)
```

In [79]:

```
quantity_norm = Normalizer(norm='l2', copy=False)
quantity_norm.fit(X_train['quantity'].values.reshape(1,-1))

quantity_norm_train = quantity_norm.transform(X_train['quantity'].values.reshape(1,-1))

quantity_norm_Test = quantity_norm.transform(X_Test['quantity'].values.reshape(1,-1))

quantity_norm_train = (X_train['quantity'].values.reshape(-1,1))

quantity_norm_Test = (X_Test['quantity'].values.reshape(-1,1))

print("Shape of quantity_norm_train matrix after one hot encodig ",quantity_norm_train.shape)

print("Shape of quantity_norm_Test matrix after one hot encodig ",quantity_norm_Test.shape)
```

```
Shape of quantity_norm_train matrix after one hot encodig  (73196, 1)
Shape of quantity_norm_Test matrix after one hot encodig  (36052, 1)
```

teacher_number_of_previously_posted_projects

In [80]:

```
teacher_prev_post_norm = Normalizer(norm='l2', copy=False)
teacher_prev_post_norm.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(
1,-1))

teacher_prev_post_norm_train =
teacher_prev_post_norm.transform(X_train['teacher_number_of_previously_posted_projects'].values.re
shape(1,-1))

teacher_prev_post_norm_Test =
teacher_prev_post_norm.transform(X_Test['teacher_number_of_previously_posted_projects'].values.res
hape(1,-1))

teacher_prev_post_norm_train =
(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

teacher_prev_post_norm_Test =
(X_Test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
print("Shape of teacher_prev_post_norm_train matrix after one hot encodig
",teacher_prev_post_norm_train.shape)

print("Shape of teacher_prev_post_norm_Test matrix after one hot encodig
",teacher_prev_post_norm_Test.shape)
```

```
Shape of teacher_prev_post_norm_train matrix after one hot encodig  (73196, 1)
Shape of teacher_prev_post_norm_Test matrix after one hot encodig  (36052, 1)
```

Title word count

Title word count

In [81]:

```
title_norm = Normalizer(norm='l2', copy=False)
title_norm.fit(X_train['word_count_title_train'].values.reshape(1,-1))
word_count_title_train = title_norm.transform(X_train['word_count_title_train'].values.reshape(1,-1
))

word_count_title_Test = title_norm.transform(X_Test['word_count_title_Test'].values.reshape(1,-1))

word_count_title_train = (X_train['word_count_title_train'].values.reshape(-1,1))

word_count_title_Test = (X_Test['word_count_title_Test'].values.reshape(-1,1))

print(word_count_title_train.shape)

print(word_count_title_Test.shape)
```

```
(73196, 1)
(36052, 1)
```

Essay word count

In [82]:

```
essay_norm = Normalizer(norm='l2', copy=False)
essay_norm.fit(X_train['word_count_essay_train'].values.reshape(1,-1))
word_count_essay_train = essay_norm.transform(X_train['word_count_essay_train'].values.reshape(1,-1
))

word_count_essay_Test = essay_norm.transform(X_Test['word_count_essay_Test'].values.reshape(1,-1))

word_count_essay_train = (X_train['word_count_essay_train'].values.reshape(-1,1))

word_count_essay_Test = (X_Test['word_count_essay_Test'].values.reshape(-1,1))

print(word_count_essay_train.shape)

print(word_count_essay_Test.shape)
```

```
(73196, 1)
(36052, 1)
```

Make Data Model Ready: encoding eassay, and project_title

1.5.3 Vectorizing Text data

1.5.3.1 Bag of words

In [83]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer_essays_bow = CountVectorizer(min_df=10)
text_bow_train = vectorizer_essays_bow.fit_transform(preprocessed_essays_train)

text_bow_Test = vectorizer_essays_bow.transform(preprocessed_essays_Test)
print("Shape of matrix after one hot encodig ",text_bow_train.shape)

print("Shape of text_bow_Test ",text_bow_Test.shape)
```

```
Shape of matrix after one hot encodig  (73196, 14144)
Shape of text_bow_Test  (36052, 14144)
```

Bag of Words for Project Title

In [84]:

In [84]:

```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
vectorizer_titles_bow = CountVectorizer(min_df=10)
title_bow_train = vectorizer_titles_bow.fit_transform(preprocessed_titles_train)

title_bow_Test = vectorizer_titles_bow.transform(preprocessed_titles_Test)
print("Shape of matrix (title) after one hot encoding ",title_bow_train.shape)

print("Shape of title_bow_test ",title_bow_Test.shape)
```

```
Shape of matrix (title) after one hot encoding  (73196, 2631)
Shape of title_bow_test  (36052, 2631)
```

1.5.2.2 TFIDF vectorizer

In [85]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_essays_tfidf = TfidfVectorizer(min_df=10)
text_tfidf_train = vectorizer_essays_tfidf.fit_transform(preprocessed_essays_train)

text_tfidf_Test = vectorizer_essays_tfidf.transform(preprocessed_essays_Test)
print("Shape of matrix after one hot encodig ",text_tfidf_train.shape)

print("Shape of text_tfidf_test ",text_tfidf_Test.shape)
```

```
Shape of matrix after one hot encodig  (73196, 14144)
Shape of text_tfidf_test  (36052, 14144)
```

TFIDF vectorizer for Project Title

In [86]:

```
vectorizer_titles_tfidf = TfidfVectorizer(min_df=10)
title_tfidf_train = vectorizer_titles_tfidf.fit_transform(preprocessed_titles_train)

title_tfidf_Test = vectorizer_titles_tfidf.transform(preprocessed_titles_Test)
print("Shape of matrix(title) after one hot encoding ",title_tfidf_train.shape)

print("Shape of title_tfidf_test ",title_tfidf_Test.shape)
```

```
Shape of matrix(title) after one hot encoding  (73196, 2631)
Shape of title_tfidf_test  (36052, 2631)
```

1.5.4 Merging all the above features

- we need to merge all the numerical vectors i.e catogorical, text, numerical vectors

In [87]:

```
print(categories_one_hot_train.shape)
print(sub_categories_one_hot_train.shape)
print(sch_one_hot_train.shape)
print(grade_one_hot_train.shape)
print(prefix_one_hot_train.shape)
print(price_norm_train.shape)
print(quantity_norm_train.shape)
print(teacher_prev_post_norm_train.shape)
print(word_count_essay_train.shape)
print(word_count_title_train.shape)
print(text_bow_train.shape)
print(title_bow_train.shape)
```

```
(73196, 9)
(73196, 30)
(73196, 51)
```

```
(73196, 51)
(73196, 4)
(73196, 6)
(73196, 1)
(73196, 1)
(73196, 1)
(73196, 1)
(73196, 1)
(73196, 14144)
(73196, 2631)
```

In [88]:

```
print(categories_one_hot_Test.shape)
print(sub_categories_one_hot_Test.shape)
print(sch_one_hot_Test.shape)
print(grade_one_hot_Test.shape)
print(prefix_one_hot_Test.shape)
print(price_norm_Test.shape)
print(quantity_norm_Test.shape)
print(teacher_prev_post_norm_Test.shape)
print(word_count_essay_Test.shape)
print(word_count_title_Test.shape)
print(text_bow_Test.shape)
print(title_bow_Test.shape)
```

```
(36052, 9)
(36052, 30)
(36052, 51)
(36052, 4)
(36052, 6)
(36052, 1)
(36052, 1)
(36052, 1)
(36052, 1)
(36052, 1)
(36052, 14144)
(36052, 2631)
```

Merging vectorised Test data

In [89]:

```
X1_Test =
hstack((categories_one_hot_Test,sub_categories_one_hot_Test,sch_one_hot_Test,grade_one_hot_Test,pr
efix_one_hot_Test,text_bow_Test,title_bow_Test,price_norm_Test, quantity_norm_Test,
teacher_prev_post_norm_Test, word_count_essay_Test, word_count_title_Test))
print(X1_Test.shape)
X2_Test =
hstack((categories_one_hot_Test,sub_categories_one_hot_Test,sch_one_hot_Test,grade_one_hot_Test,pr
efix_one_hot_Test,text_tfidf_Test,title_tfidf_Test,price_norm_Test, quantity_norm_Test,
teacher_prev_post_norm_Test, word_count_essay_Test, word_count_title_Test))
print(X2_Test.shape)
```

```
(36052, 16880)
(36052, 16880)
```

In [90]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
X1_train =
hstack((categories_one_hot_train,sub_categories_one_hot_train,sch_one_hot_train,grade_one_hot_trair
,prefix_one_hot_train, text_bow_train,title_bow_train, price_norm_train, quantity_norm_train, teach
er_prev_post_norm_train, word_count_essay_train, word_count_title_train))
print(X1_train.shape)
X2_train =
hstack((categories_one_hot_train,sub_categories_one_hot_train,sch_one_hot_train,grade_one_hot_trair
,prefix_one_hot_train, text_tfidf_train,title_tfidf_train, price_norm_train, quantity_norm_train, t
eacher_prev_post_norm_train, word_count_essay_train, word_count_title_train))
print(X2_train.shape)

print(y_train.shape)
```

```
(73196, 16880)
(73196, 16880)
(73196,)
```

Assignment 4: Naive Bayes

1. **Apply Multinomial NaiveBayes on these feature sets**
    - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (BOW)
    - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
2. **The hyper paramter tuning(find best Alpha)**
    - Find the best hyper parameter which will give the maximum AUC value
    - Consider a wide range of alpha values for hyperparameter tuning, start as low as 0.00001
    - Find the best hyper paramter using k-fold cross validation or simple cross validation data
    - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning
3. **Feature importance**
    - Find the top 10 features of positive class and top 10 features of negative class for both feature sets Set 1 and Set 2 using values of `feature_log_prob_` parameter of MultinomialNB and print their corresponding feature names
4. **Representation of results**
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure. Here on X-axis you will have alpha values, since they have a wide range, just to represent those alpha values on the graph, apply log function on those alpha values.
    - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
    - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.
5. **Conclusion**
    - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

2. Naive Bayes

2.4 Appling NB() on different kind of featurization as mentioned in the instructions

Apply Naive Bayes on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

2.4.1 Applying Naive Bayes on BOW, SET 1

In [106]:

```
%%time
alphas = [0.00001, 0.0001, 0.001, 0.01, 0.1,0.25,0.5,1,10,100,1000]
log_alphas = []
auc_scores_train = []
auc_scores_cv = []

nb = MultinomialNB(fit_prior=True,class_prior=[0.5,0.5])
parameters = {'alpha':[0.00001, 0.0001, 0.001, 0.01, 0.1,0.25,0.5,1,10,100,1000]}
GridSearch_nb = GridSearchCV(nb, parameters, cv= 10, n_jobs=-1, scoring='roc_auc',return_train_scor
e=True,verbose=1)
```

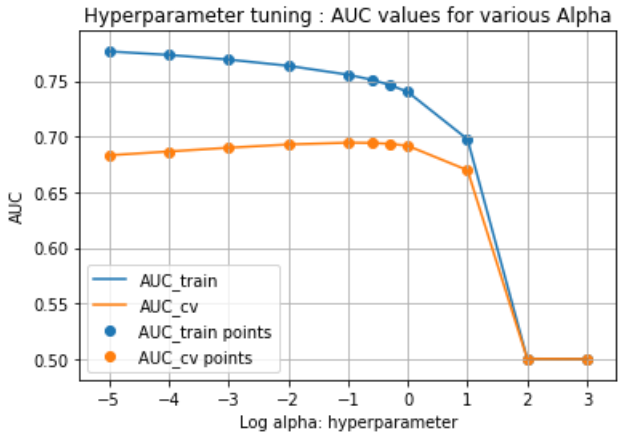
```
GridSearch_nb.fit(X1_train, y_train)
auc_scores_train= GridSearch_nb.cv_results_['mean_train_score']
auc_scores_cv = GridSearch_nb.cv_results_['mean_test_score']
print('GridSearch_nb.best_estimator_: ', GridSearch_nb.best_estimator_)
print('GridSearch_nb.best_params_: ', GridSearch_nb.best_params_)
print('GridSearch_nb.best_score_: ', GridSearch_nb.best_score_)

for av in tqdm(alphas):
    b = np.log10(av)
    log_alphas.append(b)

# Performance of model on Train data and Test data for each hyper parameter.
plt.plot(log_alphas, auc_scores_train, label='AUC_train')
plt.gca()
plt.plot(log_alphas, auc_scores_cv, label='AUC_cv')
plt.gca()
plt.scatter(log_alphas, auc_scores_train, label='AUC_train points')
plt.scatter(log_alphas, auc_scores_cv, label='AUC_cv points')
plt.legend()
plt.xlabel("Log alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyperparameter tuning : AUC values for various Alpha") # Alpha value: Hyperparameter vs
AUC
plt.grid()
plt.show()
```

```
Fitting 10 folds for each of 11 candidates, totalling 110 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   15.9s
[Parallel(n_jobs=-1)]: Done 110 out of 110 | elapsed:   28.6s finished
```

```
GridSearch_nb.best_estimator_:  MultinomialNB(alpha=0.1, class_prior=[0.5, 0.5], fit_prior=True)
GridSearch_nb.best_params_:  {'alpha': 0.1}
GridSearch_nb.best_score_:  0.6945325467830968
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 11/11
[00:00<00:00, 2755.13it/s]
```




```
Wall time: 30.6 s
```

Observations:

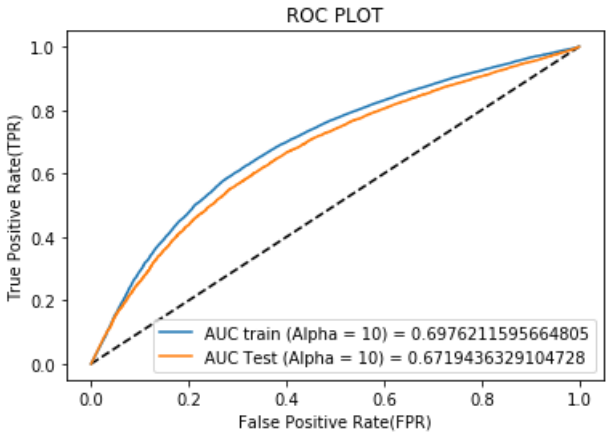
- Alpha values are chosen from 0.00001 to 1000 and for the sake of graphical representation the alpha values are scaled down by applying a log function on the alpha values without losing the relationship with their corresponding AUC values.
- At alpha = 10 the graph gains a curve downwards and at alpha = 10 the graph takes sharp dip and consequently the latter is considered as the optimal alpha value since it gives a good balance between overfitting and underfitting of the model.

In [107]:

```
#a1=GridSearch_nb.best_params_['alpha']
a1=10
nb_opt = MultinomialNB(alpha=a1,fit_prior=True,class_prior=[0.5,0.5])
```

```
nb_opt.fit(X1_train, y_train)
pred = nb_opt.predict(X1_Test)
acc = accuracy_score(y_Test, pred, normalize=True) * float(100)
print('\nThe optimal Alpha(a) = {0}'.format(a1))
print('\nTest accuracy for (Alpha = {0}) is {1}%'.format(a1,acc))

fpr_train, tpr_train, thresholds = roc_curve(y_train, nb_opt.predict_proba(X1_train.tocsr()) [:,1])
fpr_Test, tpr_Test, thresholds = roc_curve(y_Test, nb_opt.predict_proba(X1_Test.tocsr())[:,1 ])

#ROC plot
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr_train, tpr_train, label="AUC train (Alpha = {0}) = ".format(a1)+str(auc(fpr_train, tpr
_train)))
plt.plot(fpr_Test, tpr_Test, label="AUC Test (Alpha = {0}) = ".format(a1)+str(auc(fpr_Test, tpr_Tes
t)))
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("ROC PLOT")
plt.show()
print("AUC value for train data (Alpha = {0}) = ".format(a1), auc(fpr_train, tpr_train))
print("AUC value for Test data (Alpha = {0}) = ".format(a1), auc(fpr_Test, tpr_Test))
print("="*115)

AUC1 = auc(fpr_Test, tpr_Test)
pred0 = nb_opt.predict(X1_train)
pred2 = nb_opt.predict(X1_Test)
```

```
The optimal Alpha(a) = 10

Test accuracy for (Alpha = 10) is 72.83923222012648%
```



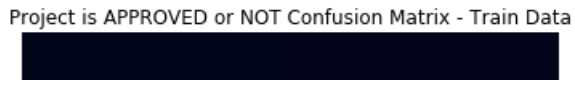

```
AUC value for train data (Alpha = 10) =  0.6976211595664805
AUC value for Test data (Alpha = 10) =  0.6719436329104728
=========================================================================================================
============
```

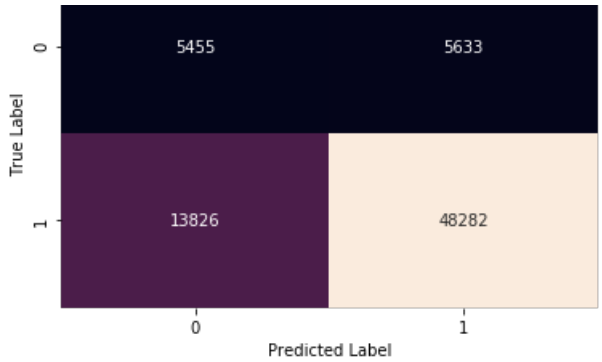
In [108]:

```
#https://seaborn.pydata.org/generated/seaborn.heatmap.html
#https://getaravind.com/blog/confusion-matrix-seaborn-heatmap/
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels
%matplotlib inline
Train = confusion_matrix(y_train, pred0)
sns.heatmap(Train,annot=True,cbar=False,fmt='g')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Train Data')
```

Out[108]:

```
Text(0.5, 1, 'Project is APPROVED or NOT Confusion Matrix - Train Data')
```





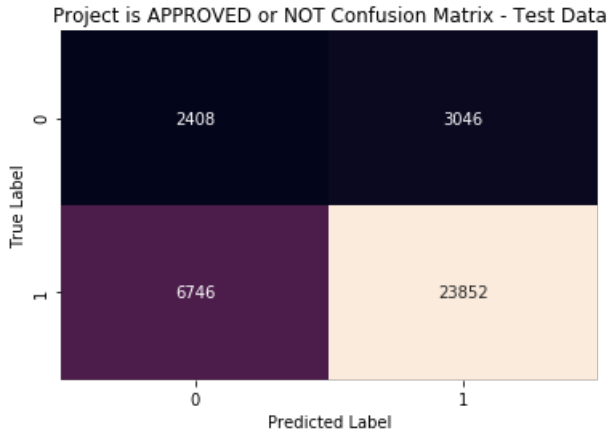
Observations for train data: Here we got 5455 - true positives, 48282 - true negatives, 13826 - false negatives, 5633 - false positives. But, false positives and false negatives are relatively high.

In [109]:

```
#https://seaborn.pydata.org/generated/seaborn.heatmap.html
#https://getaravind.com/blog/confusion-matrix-seaborn-heatmap/
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels
Test = confusion_matrix(y_Test, pred2)
sns.heatmap(Test,annot=True,cbar=False,fmt='g')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Test Data')
```

Out[109]:

```
Text(0.5, 1, 'Project is APPROVED or NOT Confusion Matrix - Test Data')
```



Observations for Test data: Here we got 2408 - true positives, 23852 - true negatives, 6746 - false negatives, 3046 - false positives. But, false positives and false negatives are relatively high.

Select feature names

In [110]:

```
%%time
nb_bow = MultinomialNB(alpha = a1,class_prior=[0.5,0.5])
nb_bow.fit(X1_train.tocsr(), y_train)
features_bow = []
for c in vectorizer_cat.get_feature_names() :
    features_bow.append(c)
for c in vectorizer_sub_cat.get_feature_names() :
    features_bow.append(c)
for c in vectorizer_sch.get_feature_names() :
    features_bow.append(c)
for c in vectorizer_grade.get_feature_names() :
    features_bow.append(c)
for c in vectorizer_prefix.get_feature_names() :
    features_bow.append(c)
for c in vectorizer_essays_bow.get_feature_names() :
```

```
    features_bow.append(c)
for c in vectorizer_titles_bow.get_feature_names() :
    features_bow.append(c)
features_bow.append("price")
features_bow.append("quantity")
features_bow.append("prev_proposed_projects")
features_bow.append("essay_word_count")
features_bow.append("title_word_count")
print(len(features_bow))
```

```
16880
Wall time: 692 ms
```

2.4.1.1 Top 10 important features of positive class from SET 1

In [111]:

```
%%time
bow_positive_features = nb_bow.feature_log_prob_[1, :].argsort()[::-1]
for i in bow_positive_features[:10]:
    print(features_bow[i])
```

```
essay_word_count
quantity
prev_proposed_projects
students
title_word_count
school
learning
classroom
not
learn
Wall time: 46.8 ms
```

2.4.1.2 Top 10 important features of negative class from SET 1

In [112]:

```
%%time
bow_negative_features = nb_bow.feature_log_prob_[0, :].argsort()[::-1]
for i in bow_negative_features[:10]:
    print(features_bow[i])
```

```
essay_word_count
quantity
prev_proposed_projects
students
title_word_count
school
learning
classroom
not
learn
Wall time: 39.6 ms
```

2.4.2 Applying Naive Bayes on TFIDF, SET 2

In [113]:

```
%%time
alphas = [0.00001, 0.0001, 0.001, 0.01, 0.1,0.25,0.5,1,10,100,1000]
log_alphas = []
auc_scores_train = []
auc_scores_cv = []

nb = MultinomialNB(fit_prior=True,class_prior=[0.5,0.5])
parameters = {'alpha':[0.00001, 0.0001, 0.001, 0.01, 0.1,0.25,0.5,1,10,100,1000]}
GridSearch_nb = GridSearchCV(nb, parameters, cv= 10, n_jobs=-1, scoring='roc_auc',return_train_scor
```

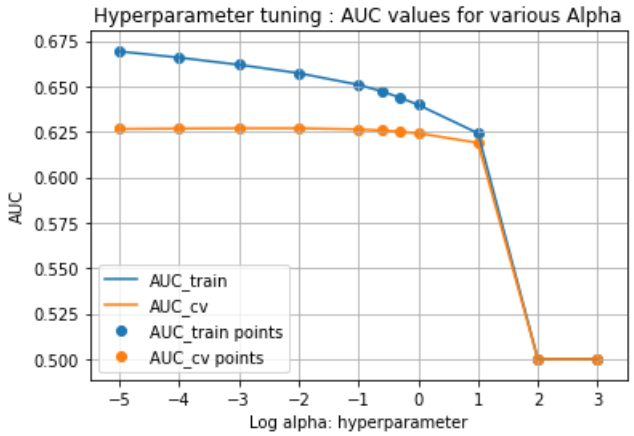
```
e=True,verbose=1)
GridSearch_nb.fit(X2_train, y_train)
auc_scores_train= GridSearch_nb.cv_results_['mean_train_score']
auc_scores_cv = GridSearch_nb.cv_results_['mean_test_score']
print('GridSearch_nb.best_estimator_: ', GridSearch_nb.best_estimator_)
print('GridSearch_nb.best_params_: ', GridSearch_nb.best_params_)
print('GridSearch_nb.best_score_: ', GridSearch_nb.best_score_)

for av in tqdm(alphas):
    b = np.log10(av)
    log_alphas.append(b)

# Performance of model on Train data and Test data for each hyper parameter.
plt.plot(log_alphas, auc_scores_train, label='AUC_train')
plt.gca()
plt.plot(log_alphas, auc_scores_cv, label='AUC_cv')
plt.gca()
plt.scatter(log_alphas, auc_scores_train, label='AUC_train points')
plt.scatter(log_alphas, auc_scores_cv, label='AUC_cv points')
plt.legend()
plt.xlabel("Log alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyperparameter tuning : AUC values for various Alpha") # Alpha value: Hyperparameter vs
AUC
plt.grid()
plt.show()
```

```
Fitting 10 folds for each of 11 candidates, totalling 110 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    7.9s
[Parallel(n_jobs=-1)]: Done 110 out of 110 | elapsed:   22.1s finished
```

```
GridSearch_nb.best_estimator_:  MultinomialNB(alpha=0.01, class_prior=[0.5, 0.5], fit_prior=True)
GridSearch_nb.best_params_:  {'alpha': 0.01}
GridSearch_nb.best_score_:  0.6268794924001962
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 11/11
[00:00<00:00, 11187.52it/s]
```




```
Wall time: 23.8 s
```

Observations:

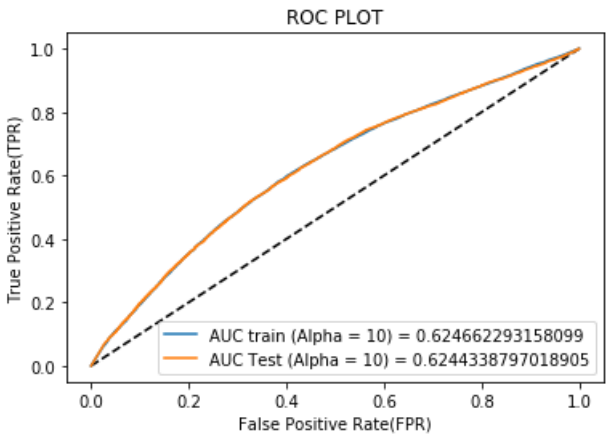
- Alpha values are chosen from 0.00001 to 1000 and for the sake of graphical representation the alpha values are scaled down by applying a log function on the alpha values without losing the relationship with their corresponding AUC values.
- At alpha = 10 the graph takes sharp dip and it is considered as the optimal alpha value since it gives a good balance between overfitting and underfitting of the model.

In [114]:

```
#a2=GridSearch_nb.best_params_['alpha']
a2=10
nb_opt = MultinomialNB(alpha=a2, fit_prior=True, class_prior=[0.5,0.5])
```

```
nb_opt = MultinomialNB(alpha=a2,fit_prior=True,class_prior=[0.5,0.5])
nb_opt.fit(X2_train, y_train)
pred = nb_opt.predict(X2_Test)
acc = accuracy_score(y_Test, pred, normalize=True) * float(100)
print('\nThe optimal Alpha(a) = {0}'.format(a2))
print('\nTest accuracy for (Alpha = {0}) is {1}%'.format(a2,acc))

fpr_train, tpr_train, thresholds = roc_curve(y_train, nb_opt.predict_proba(X2_train.tocsr()) [:,1])
fpr_Test, tpr_Test, thresholds = roc_curve(y_Test, nb_opt.predict_proba(X2_Test.tocsr())[:,1 ])

#ROC plot
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr_train, tpr_train, label="AUC train (Alpha = {0}) = ".format(a2)+str(auc(fpr_train, tpr
_train)))
plt.plot(fpr_Test, tpr_Test, label="AUC Test (Alpha = {0}) = ".format(a2)+str(auc(fpr_Test, tpr_Tes
t)))
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("ROC PLOT")
plt.show()
print("AUC value for train data (Alpha = {0}) =".format(a2), auc(fpr_train, tpr_train))
print("AUC value for Test data (Alpha = {0}) =".format(a2), auc(fpr_Test, tpr_Test))
print("="*115)

AUC2 = auc(fpr_Test, tpr_Test)
pred0 = nb_opt.predict(X2_train)
pred2 = nb_opt.predict(X2_Test)
```

```
The optimal Alpha(a) = 10

Test accuracy for (Alpha = 10) is 80.05658493287474%
```



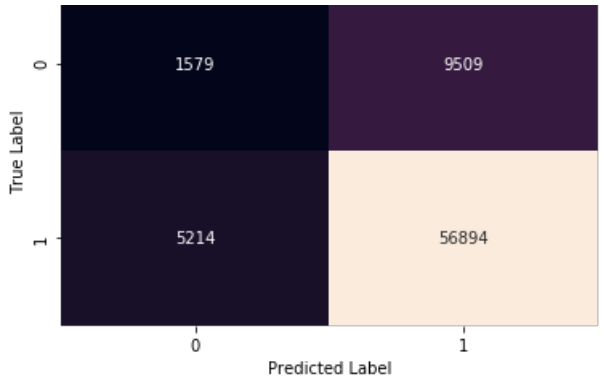

```
AUC value for train data (Alpha = 10) = 0.624662293158099
AUC value for Test data (Alpha = 10) = 0.6244338797018905
=============================================================================================================
============
```

In [115]:

```
#https://seaborn.pydata.org/generated/seaborn.heatmap.html
#https://getaravind.com/blog/confusion-matrix-seaborn-heatmap/
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels
%matplotlib inline
Train = confusion_matrix(y_train, pred0)
sns.heatmap(Train,annot=True,cbar=False,fmt='g')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Train Data')
```

Out[115]:

```
Text(0.5, 1, 'Project is APPROVED or NOT Confusion Matrix - Train Data')
```



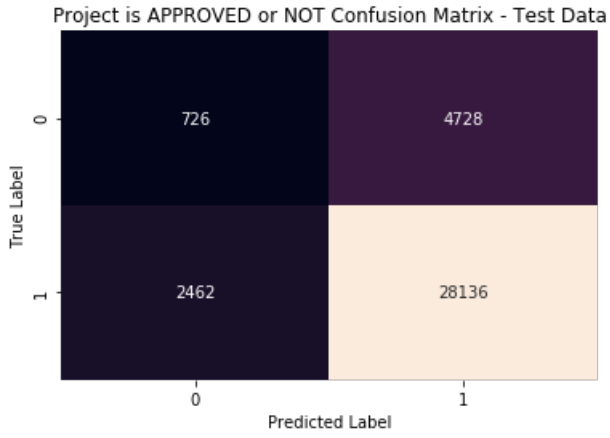
Observations for train data: Here we got 1579 - true positives, 56894 - true negatives, 5214 - false negatives, 9509 - false positives. But, false positives are relatively very high.

In [116]:

```
#https://seaborn.pydata.org/generated/seaborn.heatmap.html
#https://getaravind.com/blog/confusion-matrix-seaborn-heatmap/
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels
Test = confusion_matrix(y_Test, pred2)
sns.heatmap(Test,annot=True,cbar=False,fmt='g')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Test Data')
```

Out[116]:

```
Text(0.5, 1, 'Project is APPROVED or NOT Confusion Matrix - Test Data')
```



Observations for Test data: Here we got 726 - true positives, 28136 - true negatives, 2462 - false negatives, 4728 - false positives. But, false positives are relatively high.

Select feature names

In [117]:

```
%%time
nb_tfidf = MultinomialNB(alpha = a2,class_prior=[0.5,0.5])
nb_tfidf.fit(X2_train.tocsr(), y_train)
features_tfidf = []
for c in vectorizer_cat.get_feature_names() :
    features_tfidf.append(c)
for c in vectorizer_sub_cat.get_feature_names() :
    features_tfidf.append(c)
for c in vectorizer_sch.get_feature_names() :
    features_tfidf.append(c)
for c in vectorizer_grade.get_feature_names() :
    features_tfidf.append(c)
for c in vectorizer_prefix.get_feature_names() :
    features_tfidf.append(c)
for c in vectorizer_essays_bow.get_feature_names() :
```

```
for c in vectorizer_essays_bow.get_feature_names() :
    features_tfidf.append(c)
for c in vectorizer_titles_bow.get_feature_names() :
    features_tfidf.append(c)
features_tfidf.append("price")
features_tfidf.append("quantity")
features_tfidf.append("prev_proposed_projects")
features_tfidf.append("essay_word_count")
features_tfidf.append("title_word_count")
print(len(features_tfidf))
```

```
16880
Wall time: 634 ms
```

2.4.2.1 Top 10 important features of positive class from SET 2

In [118]:

```
%%time
tfidf_positive_features = nb_tfidf.feature_log_prob_[1, :].argsort()[::-1]
for i in tfidf_positive_features[:10]:
    print(features_tfidf[i])
```

```
essay_word_count
quantity
prev_proposed_projects
title_word_count
mrs
Literacy_Language
grades_prek_2
Math_Science
ms
grades_3_5
Wall time: 51.6 ms
```

2.4.2.2 Top 10 important features of negative class from SET 2

In [119]:

```
%%time
tfidf_negative_features = nb_tfidf.feature_log_prob_[0, :].argsort()[::-1]
for i in tfidf_negative_features[:10]:
    print(features_tfidf[i])
```

```
essay_word_count
quantity
prev_proposed_projects
title_word_count
mrs
Literacy_Language
grades_prek_2
Math_Science
ms
grades_3_5
Wall time: 77.4 ms
```

3. Conclusions

In [120]:

```
from prettytable import PrettyTable
pt = PrettyTable()
pt.field_names = ["Vectorizer", "Model", "Alpha:Hyper Parameter", " Test AUC"]
pt.add_row(["BOW", "Naive Bayes", a1, AUC1])
pt.add_row(["TFIDF", "Naive Bayes", a2, AUC2])
print(pt)
```

```
+------------+-------------+-----------------------+--------------------+
```

Vectorizer	Model	Alpha:Hyper Parameter	Test AUC
BOW	Naive Bayes	10	0.6719436329104728
TFIDF	Naive Bayes	10	0.6244338797018905

SUMMARY:

1. In the 'Multinomial Naive Bayes' model the AUC scores are higher for both BOW and TFIDF when compared to the KNN model.
2. Also'Multinomial Naive Bayes' is easier to understand and much faster and efficient in terms of space and time consumption than that of 'K-Nearest Neighbors(KNN)'.
3. Based on the confusion matrices of both BOW and TFIDF (Multinomial Naive Bayes models), TFIDF is better than BOW, because the False negatives are low and it is good since the projects that are supposed to be accepted are more likely to be marked correctly by the model (TFIDF).
4. With the right Alpha value a good balance between overfitting and underfitting can be reached where higher AUC scores and optimal performance can be achived.