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 [64]:

```
%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.linear_model import LogisticRegression
from sklearn.preprocessing import Normalizer
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')
import pdb
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\arjun\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

1.1 Reading Data

In [65]:

```
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 [66]:

```
# 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[66]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_s
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27	Grades PreK-2	

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	00:27:36 Date	project_grade_category	project_
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 [67]:

```
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 [68]:

```
# 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 = []
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 [69]:

```
# 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 [70]:

```
# 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 [71]:

```
# 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 [72]:

```
# 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', "do
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 [73]:

```
# 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
[01:09<00:00, 1046.47it/s]
100%|████████████████████████████████████████████████████████████████████████████| 36052/36052
[00:34<00:00, 1052.66it/s]
```

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

In [74]:

```
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, 31702.40it/s]
100%|███████████████████████████████████████████████████████████████████████████| 36052/36052
[00:00<00:00, 40302.47it/s]
```

1.4 Preprocessing of `project_title`

1.4 Preprocessing of project_title

In [75]:

```
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, 15051.65it/s]
100%|██████████████████████████████████████████████████████████████████████████| 36052/36052
[00:01<00:00, 21934.37it/s]
```

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

In [76]:

```
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, 509665.65it/s]
100%|█████████████████████████████████████████████████████████████████████████| 36052/36052
[00:00<00:00, 220414.97it/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 [77]:

```
# 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 [78]:

```
# 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 [79]:

```
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())
```

```
#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 [80]:

```
# 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_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 [81]:

```
# 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=False, 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 [82]:

```
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 [83]:

```
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 [84]:

```
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 [85]:

```
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

In [86]:

```
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 [87]:

```
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)
```

Sentiment Scores

In [88]:

```
# https://www.geeksforgeeks.org/python-sentiment-analysis-using-vader/
sid = SentimentIntensityAnalyzer()
essays = X_train['essay']
sentiment_essay_Train = []
for essay in tqdm(essays):
    res = sid.polarity_scores(essay)
    sentiment_essay_Train.append(res['compound']) #Considering compound as a criteria.
X_train['sentiment_essay'] = sentiment_essay_Train

essays = X_Test['essay']
sentiment_essay_Test = []
for essay in tqdm(essays):
    res = sid.polarity_scores(essay)
    sentiment_essay_Test.append(res['compound']) #Considering compound as a criteria.
X_Test['sentiment_essay'] = sentiment_essay_Test

sentiment_norm = Normalizer(norm='l2', copy=False)
```

```
sentiment_norm.fit(X_train['sentiment_essay'].values.reshape(1,-1))

sentiment_norm_train = sentiment_norm.transform(X_train['sentiment_essay'].values.reshape(1,-1))

sentiment_norm_Test = sentiment_norm.transform(X_Test['sentiment_essay'].values.reshape(1,-1))

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

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

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

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

```
100%|███████████████████████████████████████████████████████████████████████████| 73196/73196 [06:
36<00:00, 184.50it/s]
100%|███████████████████████████████████████████████████████████████████████████| 36052/36052 [03:
20<00:00, 179.55it/s]
```

```
Shape of sentiment_norm_train matrix after one hot encodig  (73196, 1)
Shape of sentiment_norm_Test matrix after one hot encodig  (36052, 1)
```

Make Data Model Ready: encoding essay and project_title

1.5.3 Vectorizing Text data

1.5.3.1 Bag of words

In [89]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer_essays_bow = CountVectorizer(min_df=10,max_features = 5000,ngram_range=(2, 2))
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, 5000)
Shape of text_bow_Test  (36052, 5000)
```

Bag of Words for Project Title

In [90]:

```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
vectorizer_titles_bow = CountVectorizer(min_df=10,max_features = 5000,ngram_range=(2, 2))
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, 2698)
Shape of title_bow_test  (36052, 2698)
```

1.5.2.2 TFIDF vectorizer

In [91]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_essays_tfidf = TfidfVectorizer(min_df=10,max_features = 5000,ngram_range=(2, 2))
```

```
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, 5000)
Shape of text_tfidf_test  (36052, 5000)
```

TFIDF vectorizer for Project Title

In [92]:

```
vectorizer_titles_tfidf = TfidfVectorizer(min_df=10,max_features = 5000,ngram_range=(2, 2))
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, 2698)
Shape of title_tfidf_test  (36052, 2698)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [93]:

```
'''
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
    return model
model = loadGloveModel('glove.42B.300d.txt')

# ============================
Output:

Loading Glove Model
1917495it [06:32, 4879.69it/s]
Done. 1917495  words loaded!

# ============================

words = []
for i in preproced_texts:
    words.extend(i.split(' '))

for i in preproced_titles:
    words.extend(i.split(' '))
print("all the words in the coupus", len(words))
words = set(words)
print("the unique words in the coupus", len(words))

inter_words = set(model.keys()).intersection(words)
print("The number of words that are present in both glove vectors and our coupus", \
      len(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")

words_courpus = {}
words_glove = set(model.keys())
for i in words:
    if i in words_glove:
        words_courpus[i] = model[i]
```

```
print("word 2 vec length", len(words_courpus))


# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
ve-and-load-variables-in-python/

import pickle
with open('glove_vectors', 'wb') as f:
    pickle.dump(words_courpus, f)


'''
```

Out[93]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef
loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\',
encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n        splitLine = line.split()\n
word = splitLine[0]\n        embedding = np.array([float(val) for val in splitLine[1:]])\n        m
odel[word] = embedding\n    print ("Done.",len(model)," words loaded!")\n    return model\nmodel =
loadGloveModel(\'glove.42B.300d.txt\')\n\n# ============================\nOutput:\n    \nLoading G
love Model\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n#
============================\n\nwords = []\nfor i in preproced_texts:\n    words.extend(i.split(\'
\'))\n\nfor i in preproced_titles:\n    words.extend(i.split(\' \'))\nprint("all the words in the
coupus", len(words))\nwords = set(words)\nprint("the unique words in the coupus",
len(words))\n\ninter_words = set(model.keys()).intersection(words)\nprint("The number of words tha
t are present in both glove vectors and our coupus",       len(inter_words),"
(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove =
set(model.keys())\nfor i in words:\n    if i in words_glove:\n        words_courpus[i] = model[i]\r
print("word 2 vec length", len(words_courpus))\n\n\n# stronging variables into pickle files python
: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport pic
kle\nwith open(\'glove_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n'
```

In [94]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
ve-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

Average Word2Vec for Project_Essays

In [95]:

```
avg_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_train): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train.append(vector)
#-----------------------------------------------------------------------------------------


avg_w2v_vectors_Test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_Test): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_Test.append(vector)

print(len(avg_w2v_vectors_Test))
print(len(avg_w2v_vectors_Test[1]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 73196/73196
[00:38<00:00, 1881.89it/s]
100%|██████████████████████████████████████████████████████████████████████████| 36052/36052
[00:18<00:00, 1929.38it/s]
```

```
36052
300
```

AVG W2V on project_title

In [96]:

```
# Similarly you can vectorize for title also
# compute average word2vec for each title.
avg_w2v_vectors_title_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles_train): # for each review/sentence
    vector_title = np.zeros(300) # as word vectors are of zero length
    cnt_title_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector_title += model[word]
            cnt_title_words += 1
    if cnt_title_words != 0:
        vector_title /= cnt_title_words
    avg_w2v_vectors_title_train.append(vector_title)


#-------------------------------------------------------------------------------------------
avg_w2v_vectors_title_Test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles_Test): # for each review/sentence
    vector_title = np.zeros(300) # as word vectors are of zero length
    cnt_title_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector_title += model[word]
            cnt_title_words += 1
    if cnt_title_words != 0:
        vector_title /= cnt_title_words
    avg_w2v_vectors_title_Test.append(vector_title)

print(len(avg_w2v_vectors_title_Test))
print(len(avg_w2v_vectors_title_Test[0]))
```

```
100%|█████████████████████████████████████████████████████████████████████████| 73196/73196
[00:02<00:00, 25263.65it/s]
100%|█████████████████████████████████████████████████████████████████████████| 36052/36052
[00:01<00:00, 21225.29it/s]
```

```
36052
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [97]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_essays = TfidfVectorizer()
tfidf_model_essays.fit(preprocessed_essays_train)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_essays.get_feature_names(), list(tfidf_model_essays.idf_)))
tfidf_words_essays = set(tfidf_model_essays.get_feature_names())
```

TFIDF weighted W2V for Project_Essays

In [98]:

```
tfidf_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_train): # for each review/sentence
for sentence in tqdm(preprocessed_essays_train): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_essays):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_train.append(vector)

#------------------------------------------------------------------------------------------
tfidf_w2v_vectors_Test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_Test): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_essays):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_Test.append(vector)

print(len(tfidf_w2v_vectors_Test))
print(len(tfidf_w2v_vectors_Test[0]))
```

```
100%|███████████████████████████████████████████████████████████████████████████| 73196/73196 [04:
20<00:00, 280.70it/s]
100%|███████████████████████████████████████████████████████████████████████████| 36052/36052 [02:
04<00:00, 289.69it/s]
```

```
36052
300
```

TFIDF weighted W2V on project_title

In [99]:

```
# Similarly you can vectorize for title also
tfidf_model_title = TfidfVectorizer()
tfidf_model_title.fit(preprocessed_titles_train)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_title.get_feature_names(), list(tfidf_model_title.idf_)))
tfidf_words_title = set(tfidf_model_title.get_feature_names())

# compute tfidf word2vec for each title.
tfidf_w2v_vectors_title_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles_train): # for each review/sentence
    vector_title = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_title):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector_title += (vector_title * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector_title /= tf_idf_weight
    tfidf_w2v_vectors_title_train.append(vector_title)

#-------------------------------------------------------------------------------------------------
```

```
------------------------------------


tfidf_w2v_vectors_title_Test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles_Test): # for each review/sentence
    vector_title = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_title):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector_title += (vector_title * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector_title /= tf_idf_weight
    tfidf_w2v_vectors_title_Test.append(vector_title)

print(len(tfidf_w2v_vectors_title_Test))
print(len(tfidf_w2v_vectors_title_Test[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████████| 73196/73196
[00:04<00:00, 17813.25it/s]
100%|████████████████████████████████████████████████████████████████████████████| 36052/36052
[00:02<00:00, 15074.43it/s]
```

```
36052
300
```

1.5.4 Merging all the above features

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

Merging vectorised Test data

```
In [100]:
```

```
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, sentiment_norm_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, sentiment_norm_Test))
print(X2_Test.shape)
X3_Test =
hstack((categories_one_hot_Test,sub_categories_one_hot_Test,sch_one_hot_Test,grade_one_hot_Test,pr
efix_one_hot_Test,avg_w2v_vectors_Test,avg_w2v_vectors_title_Test,price_norm_Test,
quantity_norm_Test, teacher_prev_post_norm_Test, word_count_essay_Test, word_count_title_Test,
sentiment_norm_Test))
print(X3_Test.shape)
X4_Test =
hstack((categories_one_hot_Test,sub_categories_one_hot_Test,sch_one_hot_Test,grade_one_hot_Test,pr
efix_one_hot_Test,tfidf_w2v_vectors_Test,tfidf_w2v_vectors_title_Test,price_norm_Test,
quantity_norm_Test, teacher_prev_post_norm_Test, word_count_essay_Test, word_count_title_Test,
sentiment_norm_Test))
print(X4_Test.shape)
X5_Test =
hstack((categories_one_hot_Test,sub_categories_one_hot_Test,sch_one_hot_Test,grade_one_hot_Test,pr
efix_one_hot_Test,price_norm_Test, quantity_norm_Test, teacher_prev_post_norm_Test,
word_count_essay_Test, word_count_title_Test, sentiment_norm_Test))
print(X5_Test.shape)

print(y_Test.shape)
```

```
(36052, 7804)
(36052, 7804)
```

```
(36052, 7804)
(36052, 706)
(36052, 706)
(36052, 106)
(36052,)
```

In [101]:

```
# 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, sentiment_norm_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, sentiment_norm_train)
)
print(X2_train.shape)
X3_train =
hstack((categories_one_hot_train,sub_categories_one_hot_train,sch_one_hot_train,grade_one_hot_trair
,prefix_one_hot_train, avg_w2v_vectors_train,avg_w2v_vectors_title_train, price_norm_train,
quantity_norm_train, teacher_prev_post_norm_train, word_count_essay_train, word_count_title_train,
sentiment_norm_train))
print(X3_train.shape)
X4_train =
hstack((categories_one_hot_train,sub_categories_one_hot_train,sch_one_hot_train,grade_one_hot_trair
,prefix_one_hot_train, tfidf_w2v_vectors_train,tfidf_w2v_vectors_title_train, price_norm_train,
quantity_norm_train, teacher_prev_post_norm_train, word_count_essay_train, word_count_title_train,
sentiment_norm_train))
print(X4_train.shape)
X5_train =
hstack((categories_one_hot_train,sub_categories_one_hot_train,sch_one_hot_train,grade_one_hot_trair
,prefix_one_hot_train, price_norm_train, quantity_norm_train, teacher_prev_post_norm_train,
word_count_essay_train, word_count_title_train, sentiment_norm_train))
print(X5_train.shape)
print(y_train.shape)
```

```
(73196, 7804)
(73196, 7804)
(73196, 706)
(73196, 706)
(73196, 106)
(73196,)
```

Assignment 5: Logistic Regression

1. **[Task-1] Logistic Regression(either SGDClassifier with log loss, or LogisticRegression) on these feature sets**

   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (`BOW with bi-grams` with `min_df=10` and `max_features=5000`)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (`TFIDF with bi-grams` with `min_df=10` and `max_features=5000`)
   - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
   - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

2. **Hyper paramter tuning (find best hyper parameters corresponding the algorithm that you choose)**

   - Find the best hyper parameter which will give the maximum AUC value
   - 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. **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.
   - 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.

4. **[Task-2] Apply Logistic Regression on the below feature set Set 5 by finding the best hyper parameter as suggested in step 2 and step 3.**
5. Consider these set of features Set 5 :
    - **school_state** : categorical data
    - **clean_categories** : categorical data
    - **clean_subcategories** : categorical data
    - **project_grade_category** :categorical data
    - **teacher_prefix** : categorical data
    - **quantity** : numerical data
    - **teacher_number_of_previously_posted_projects** : numerical data
    - **price** : numerical data
    - **sentiment score's of each of the essay** : numerical data
    - **number of words in the title** : numerical data
    - **number of words in the combine essays** : numerical data

    And apply the Logistic regression on these features by finding the best hyper paramter as suggested in step 2 and step 3

6. **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

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakage, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link.

2. Logistic Regression

2.4 Appling Logistic Regression on different kind of featurization as mentioned in the instructions

Apply Logistic Regression 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

Appling Logistic Regression on feature SET 1

In [102]:

```
%%time
C_val=[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
log_C_val = []
auc_scores_train = []
auc_scores_cv = []
lr = LogisticRegression(penalty='l2', class_weight='balanced', n_jobs=-1)
parameters = {'C':C_val}
GridSearch_lr = GridSearchCV(lr, parameters, cv = 10, n_jobs=-1, scoring='roc_auc',return_train_score=True,verbose=1)
GridSearch_lr.fit(X1_train, y_train)
auc_scores_train= GridSearch_lr.cv_results_['mean_train_score']
auc_scores_cv = GridSearch_lr.cv_results_['mean_test_score']
print('GridSearch_lr.best_estimator_: ', GridSearch_lr.best_estimator_)
print('GridSearch_lr.best_params_: ', GridSearch_lr.best_params_)
print('GridSearch_lr.best_score_: ', GridSearch_lr.best_score_)

for av in tqdm(C_val):
    b = np.log10(av)
    log_C_val.append(b)
```

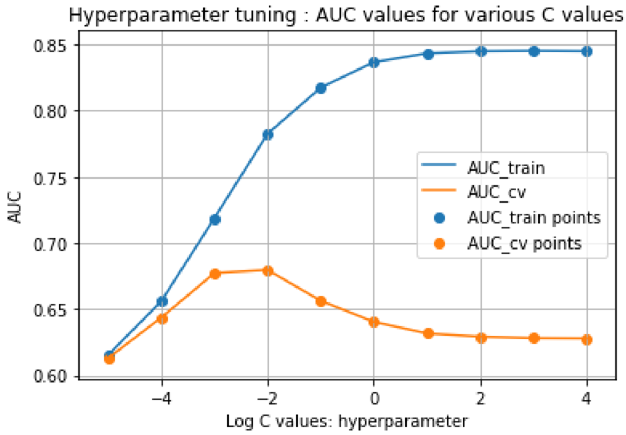
```
# Performance of model on Train data and Test data for each hyper parameter.
plt.plot(log_C_val, auc_scores_train, label='AUC_train')
plt.gca()
plt.plot(log_C_val, auc_scores_cv, label='AUC_cv')
plt.gca()
plt.scatter(log_C_val, auc_scores_train, label='AUC_train points')
plt.scatter(log_C_val, auc_scores_cv,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

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 27.4min finished
```

```
GridSearch_lr.best_estimator_:  LogisticRegression(C=0.01, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=-1, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
GridSearch_lr.best_params_:  {'C': 0.01}
GridSearch_lr.best_score_:  0.6796072346314381
```

```
100%|████████████████████████████████████████████████████████████████████████████████| 10/10
[00:00<00:00, 10029.42it/s]
```



```
Wall time: 27min 29s
Parser   : 110 ms
```

Observations:

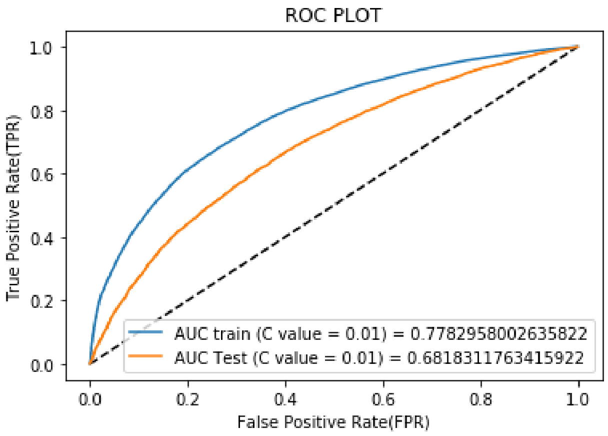
- C values are chosen from 0.00001 to 10000 and for the sake of graphical representation the C values are scaled down by applying a log function on them without losing the relationship with their corresponding AUC values.
- C = 0.01 gives the maximum cv score, hence it is considered as the optimal C value.

In [103]:

```
%%time
a1=GridSearch_lr.best_params_['C']
#a1=1e-05
lr_opt = LogisticRegression(penalty='l2', C=a1, class_weight='balanced', n_jobs=-1)
lr_opt.fit(X1_train, y_train)
pred = lr_opt.predict(X1_Test)
acc = accuracy_score(y_Test, pred, normalize=True) * float(100)
print('\nThe optimal C value = {0}'.format(a1))
print('\nTest accuracy for (C value = {0}) is {1}%'.format(a1,acc))
```

```
fpr_train, tpr_train, thresholds = roc_curve(y_train, lr_opt.predict_proba(X1_train.tocsr()) [:,1])
fpr_Test, tpr_Test, thresholds = roc_curve(y_Test, lr_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 (C value = {0}) = ".format(a1)+str(auc(fpr_train, t
pr_train)))
plt.plot(fpr_Test, tpr_Test, label="AUC Test (C value = {0}) = ".format(a1)+str(auc(fpr_Test, tpr_T
est)))
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 (C value = {0}) = ".format(a1), auc(fpr_train, tpr_train))
print("AUC value for Test data (C value = {0}) = ".format(a1), auc(fpr_Test, tpr_Test))
print("="*115)

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

```
The optimal C value = 0.01

Test accuracy for (C value = 0.01) is 65.3639187839787%
```



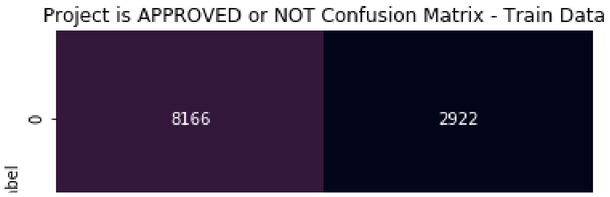
```
AUC value for train data (C value = 0.01) =  0.7782958002635822
AUC value for Test data (C value = 0.01) =  0.6818311763415922
=======================================================================================================
============
Wall time: 5.25 s
```

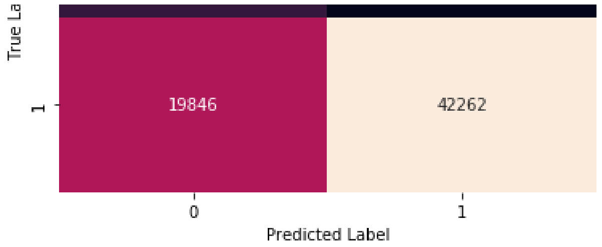
In [104]:

```
#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[104]:

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





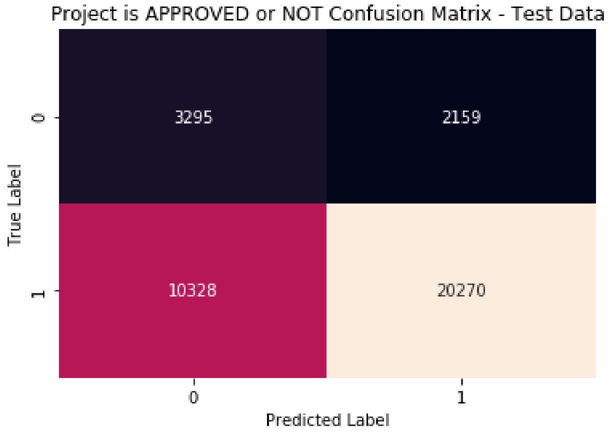
Observations for train data: Here we got 8166 - true positives, 42262 - true negatives, 19846 - false negatives, 2922 - false positives.

In [105]:

```
#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[105]:

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



Observations for Test data: Here we got 3295 - true positives, 20270 - true negatives, 10328 - false negatives, 2159 - false positives.

Appling Logistic Regression on feature SET 2

In [106]:

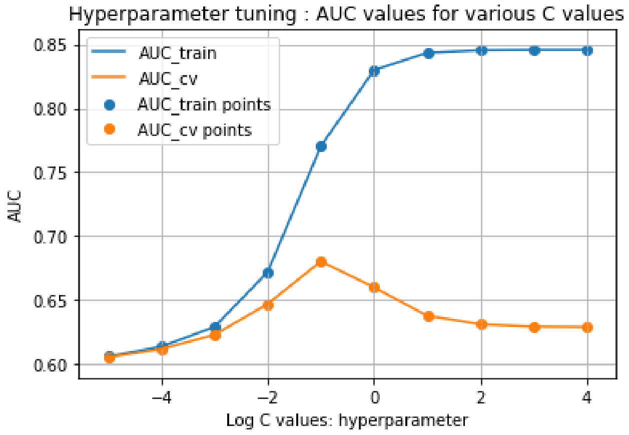
```
%%time
C_val=[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
log_C_val = []
auc_scores_train = []
auc_scores_cv = []
lr = LogisticRegression(penalty='l2', class_weight='balanced', n_jobs=-1)
parameters = {'C':C_val}
GridSearch_lr = GridSearchCV(lr, parameters, cv = 10, n_jobs=-1, scoring='roc_auc',return_train_sco
re=True,verbose=1)
GridSearch_lr.fit(X2_train, y_train)
auc_scores_train= GridSearch_lr.cv_results_['mean_train_score']
auc_scores_cv = GridSearch_lr.cv_results_['mean_test_score']
print('GridSearch_lr.best_estimator_: ', GridSearch_lr.best_estimator_)
print('GridSearch_lr.best_params_: ', GridSearch_lr.best_params_)
print('GridSearch_lr.best_score_: ', GridSearch_lr.best_score_)

for av in tqdm(C_val):
    b = np.log10(av)
    log_C_val.append(b)
    log_C_val.append(b)

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

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   53.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 22.2min finished
```

```
GridSearch_lr.best_estimator_:  LogisticRegression(C=0.1, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=-1, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
GridSearch_lr.best_params_:  {'C': 0.1}
GridSearch_lr.best_score_:  0.6802607998661343
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 10/10
[00:00<00:00, 2507.66it/s]
```



```
Wall time: 22min 18s
```

Observations:

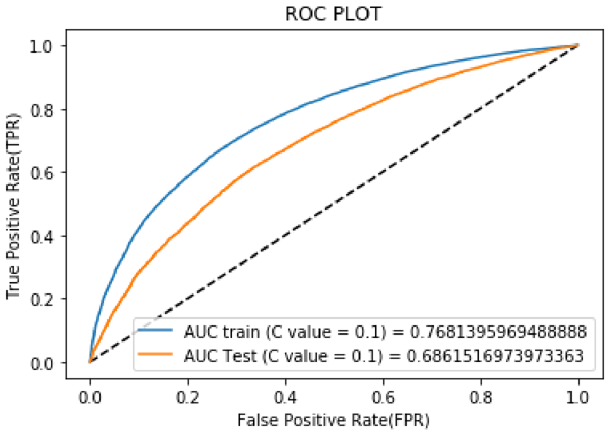
- C values are chosen from 0.00001 to 10000 and for the sake of graphical representation the C values are scaled down by applying a log function on them without losing the relationship with their corresponding AUC values.
- C = 0.1 gives the maximum cv score, hence it is considered as the optimal C value.

In [107]:

```
%%time
a2=GridSearch_lr.best_params_['C']
#a2=0.1
lr_opt = LogisticRegression(penalty='l2', C=a2, class_weight='balanced', n_jobs=-1)
lr_opt.fit(X2_train, y_train)
pred = lr_opt.predict(X2_Test)
acc = accuracy_score(y_Test, pred, normalize=True) * float(100)
print('\nThe optimal C value = {0}'.format(a2))
print('\nTest accuracy for (C value = {0}) is {1}%'.format(a2,acc))
```

```
fpr_train, tpr_train, thresholds = roc_curve(y_train, lr_opt.predict_proba(X2_train.tocsr()) [:,1])
fpr_Test, tpr_Test, thresholds = roc_curve(y_Test, lr_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 (C value = {0}) = ".format(a2)+str(auc(fpr_train, t
pr_train)))
plt.plot(fpr_Test, tpr_Test, label="AUC Test (C value = {0}) = ".format(a2)+str(auc(fpr_Test, tpr_T
est)))
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 (C value = {0}) = ".format(a2), auc(fpr_train, tpr_train))
print("AUC value for Test data (C value = {0}) = ".format(a2), auc(fpr_Test, tpr_Test))
print("="*115)

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

```
The optimal C value = 0.1

Test accuracy for (C value = 0.1) is 65.45822700543658%
```



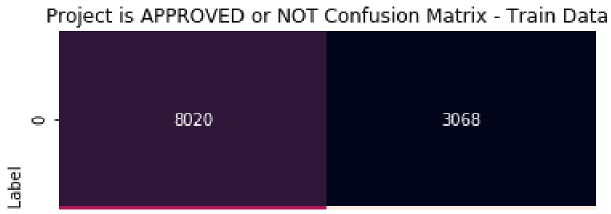
```
AUC value for train data (C value = 0.1) =  0.7681395969488888
AUC value for Test data (C value = 0.1) =  0.6861516973973363
===================================================================================================
============
Wall time: 7.61 s
```

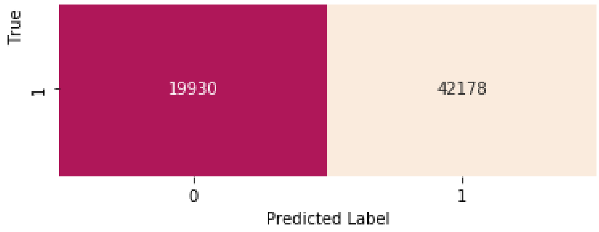
In [108]:

```
#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')
```





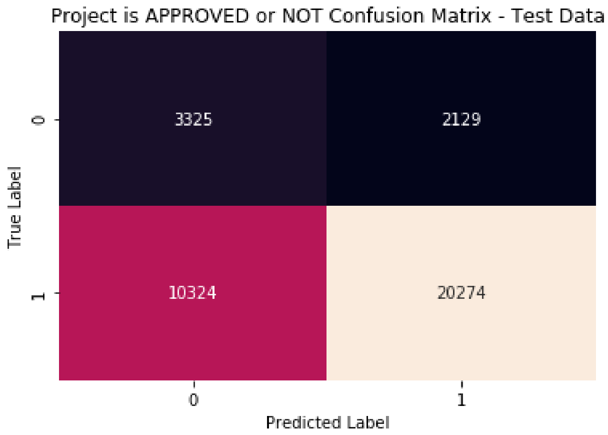
Observations for train data: Here we got 8020 - true positives, 42178 - true negatives, 19930 - false negatives, 3068 - false positives.

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 3325 - true positives, 20274 - true negatives, 10324 - false negatives, 2129 - false positives.

Appling Logistic Regression on feature SET 3

In [110]:

```
%%time
C_val=[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
log_C_val = []
auc_scores_train = []
auc_scores_cv = []
lr = LogisticRegression(penalty='l2', class_weight='balanced', n_jobs=-1)
parameters = {'C':C_val}
GridSearch_lr = GridSearchCV(lr, parameters, cv = 10, n_jobs=-1, scoring='roc_auc',return_train_sco
re=True,verbose=1)
GridSearch_lr.fit(X3_train, y_train)
auc_scores_train= GridSearch_lr.cv_results_['mean_train_score']
auc_scores_cv = GridSearch_lr.cv_results_['mean_test_score']
print('GridSearch_lr.best_estimator_: ', GridSearch_lr.best_estimator_)
print('GridSearch_lr.best_params_: ', GridSearch_lr.best_params_)
print('GridSearch_lr.best_score_: ', GridSearch_lr.best_score_)

for av in tqdm(C_val):
    b = np.log10(av)
    log_C_val.append(b)
```

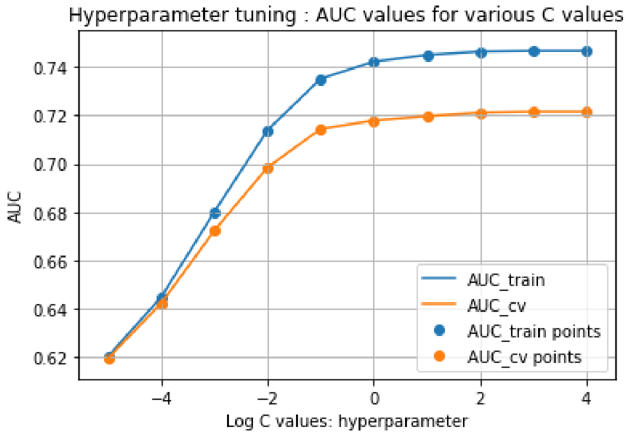
```
# Performance of model on Train data and Test data for each hyper parameter.
plt.plot(log_C_val, auc_scores_train, label='AUC_train')
plt.gca()
plt.plot(log_C_val, auc_scores_cv, label='AUC_cv')
plt.gca()
plt.scatter(log_C_val, auc_scores_train, label='AUC_train points')
plt.scatter(log_C_val, auc_scores_cv,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

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed: 16.5min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 176.8min finished
```

```
GridSearch_lr.best_estimator_:  LogisticRegression(C=1000, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=-1, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
GridSearch_lr.best_params_:  {'C': 1000}
GridSearch_lr.best_score_:  0.721570352056063
```

```
100%|██████████████████████████████████████████████████████████████████████████████████| 10/10
[00:00<00:00, 73.68it/s]
```



```
Wall time: 3h 3min 41s
```

Observations:

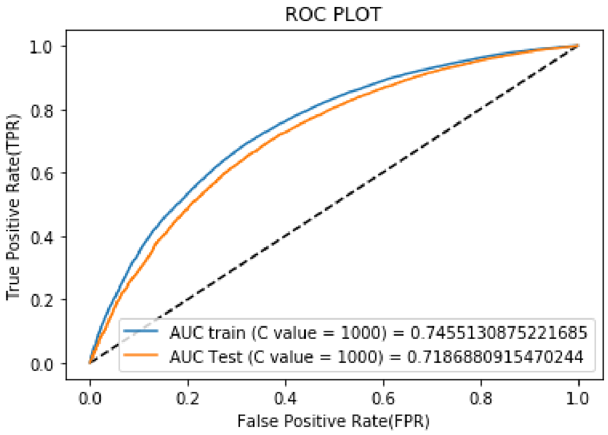
- C values are chosen from 0.00001 to 10000 and for the sake of graphical representation the C values are scaled down by applying a log function on them without losing the relationship with their corresponding AUC values.
- C = 1000 gives the maximum cv score, hence it is considered as the optimal C value.

In [111]:

```
%%time
a3=GridSearch_lr.best_params_['C']
#a3=1000
lr_opt = LogisticRegression(penalty='l2', C=a3, class_weight='balanced', n_jobs=-1)
lr_opt.fit(X3_train, y_train)
pred = lr_opt.predict(X3_Test)
acc = accuracy_score(y_Test, pred, normalize=True) * float(100)
print('\nThe optimal C value = {0}'.format(a3))
print('\nTest accuracy for (C value = {0}) is {1}%'.format(a3,acc))

fpr_train, tpr_train, thresholds = roc_curve(y_train, lr_opt.predict_proba(X3_train.tocsr()) [:,1])
```

```
fpr_train, tpr_train, thresholds = roc_curve(y_train, lr_opt.predict_proba(X3_train.tocsr())[:,1])
fpr_Test, tpr_Test, thresholds = roc_curve(y_Test, lr_opt.predict_proba(X3_Test.tocsr())[:,1 ])

#ROC plot
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr_train, tpr_train, label="AUC train (C value = {0}) = ".format(a3)+str(auc(fpr_train, t
pr_train)))
plt.plot(fpr_Test, tpr_Test, label="AUC Test (C value = {0}) = ".format(a3)+str(auc(fpr_Test, tpr_T
est)))
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 (C value = {0}) = ".format(a3), auc(fpr_train, tpr_train))
print("AUC value for Test data (C value = {0}) = ".format(a3), auc(fpr_Test, tpr_Test))
print("="*115)

AUC3 = auc(fpr_Test, tpr_Test)
pred0 = lr_opt.predict(X3_train)
pred2 = lr_opt.predict(X3_Test)
```

```
The optimal C value = 1000

Test accuracy for (C value = 1000) is 65.20026628203706%
```



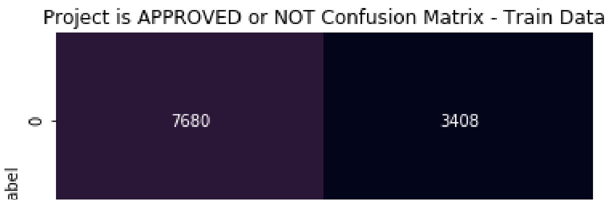
```
AUC value for train data (C value = 1000) =  0.7455130875221685
AUC value for Test data (C value = 1000) =  0.7186880915470244
===================================================================================================
============
Wall time: 6min 47s
Parser   : 295 ms
```

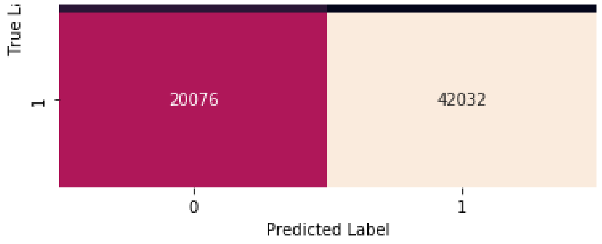
In [112]:

```
#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[112]:

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





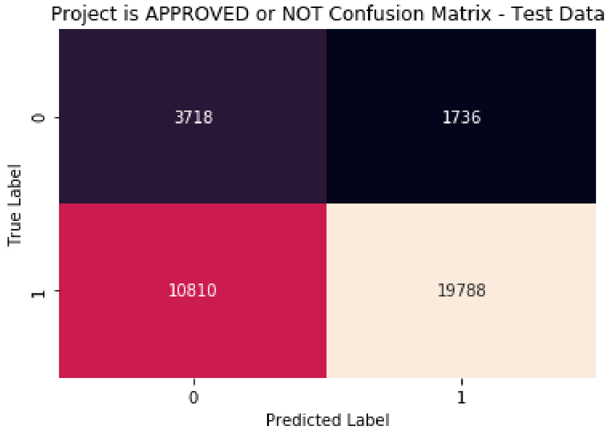
Observations for train data: Here we got 7680 - true positives, 42032 - true negatives, 20076 - false negatives, 3408 - false positives.

In [113]:

```
#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[113]:

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



Observations for Test data: Here we got 3718 - true positives, 19788 - true negatives, 10810 - false negatives, 1736 - false positives.

Appling Logistic Regression on feature SET 4

In [114]:

```
%%time
C_val=[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
log_C_val = []
auc_scores_train = []
auc_scores_cv = []
lr = LogisticRegression(penalty='l2', class_weight='balanced', n_jobs=-1)
parameters = {'C':C_val}
GridSearch_lr = GridSearchCV(lr, parameters, cv = 10, n_jobs=-1, scoring='roc_auc',return_train_sco
re=True,verbose=1)
GridSearch_lr.fit(X4_train, y_train)
auc_scores_train= GridSearch_lr.cv_results_['mean_train_score']
auc_scores_cv = GridSearch_lr.cv_results_['mean_test_score']
print('GridSearch_lr.best_estimator_: ', GridSearch_lr.best_estimator_)
print('GridSearch_lr.best_params_: ', GridSearch_lr.best_params_)
print('GridSearch_lr.best_score_: ', GridSearch_lr.best_score_)

for av in tqdm(C_val):
    b = np.log10(av)
    log_C_val.append(b)
```

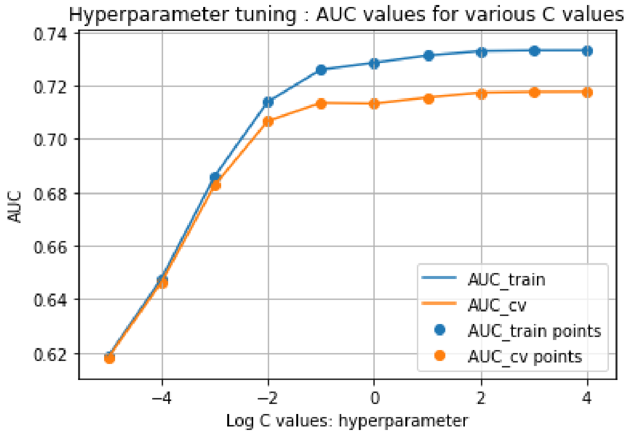
```
# Performance of model on Train data and Test data for each hyper parameter.
plt.plot(log_C_val, auc_scores_train, label='AUC_train')
plt.gca()
plt.plot(log_C_val, auc_scores_cv, label='AUC_cv')
plt.gca()
plt.scatter(log_C_val, auc_scores_train, label='AUC_train points')
plt.scatter(log_C_val, auc_scores_cv,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

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  3.3min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 74.0min finished
```

```
GridSearch_lr.best_estimator_:  LogisticRegression(C=10000, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=-1, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
GridSearch_lr.best_params_:  {'C': 10000}
GridSearch_lr.best_score_:  0.7176734207209854
```

```
100%|██████████████████████████████████████████████████████████████████████| 10/10
[00:00<00:00, 358.09it/s]
```



```
Wall time: 1h 16min 30s
```

Observations:

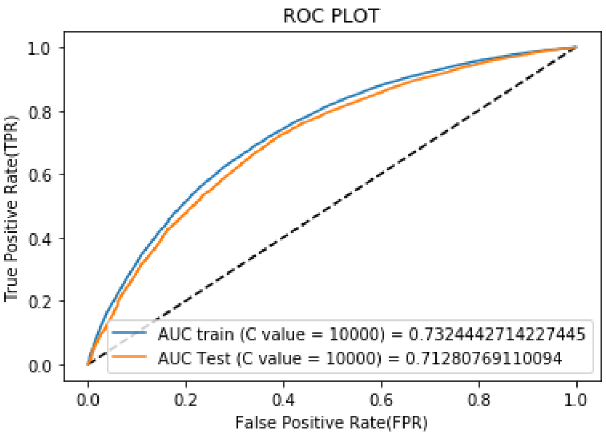
- C values are chosen from 0.00001 to 10000 and for the sake of graphical representation the C values are scaled down by applying a log function on them without losing the relationship with their corresponding AUC values.
- C = 10000 gives the maximum cv score, hence it is considered as the optimal C value.

In [115]:

```
%%time
a4=GridSearch_lr.best_params_['C']
#a4=0.01
lr_opt = LogisticRegression(penalty='l2', C=a4, class_weight='balanced', n_jobs=-1)
lr_opt.fit(X4_train, y_train)
pred = lr_opt.predict(X4_Test)
acc = accuracy_score(y_Test, pred, normalize=True) * float(100)
print('\nThe optimal C value = {0}'.format(a4))
print('\nTest accuracy for (C value = {0}) is {1}%'.format(a4,acc))
```

```
fpr_train, tpr_train, thresholds = roc_curve(y_train, lr_opt.predict_proba(X4_train.tocsr()) [:,1])
fpr_Test, tpr_Test, thresholds = roc_curve(y_Test, lr_opt.predict_proba(X4_Test.tocsr())[:,1 ])

#ROC plot
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr_train, tpr_train, label="AUC train (C value = {0}) = ".format(a4)+str(auc(fpr_train, t
pr_train)))
plt.plot(fpr_Test, tpr_Test, label="AUC Test (C value = {0}) = ".format(a4)+str(auc(fpr_Test, tpr_T
est)))
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 (C value = {0}) = ".format(a4), auc(fpr_train, tpr_train))
print("AUC value for Test data (C value = {0}) = ".format(a4), auc(fpr_Test, tpr_Test))
print("="*115)

AUC4 = auc(fpr_Test, tpr_Test)
pred0 = lr_opt.predict(X4_train)
pred2 = lr_opt.predict(X4_Test)
```

```
The optimal C value = 10000

Test accuracy for (C value = 10000) is 62.45978031731943%
```



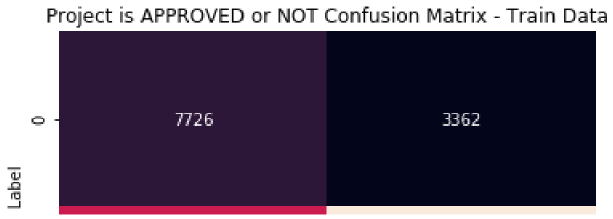
```
AUC value for train data (C value = 10000) =  0.7324442714227445
AUC value for Test data (C value = 10000) =  0.71280769110094
=======================================================================================================
============
Wall time: 2min 25s
```

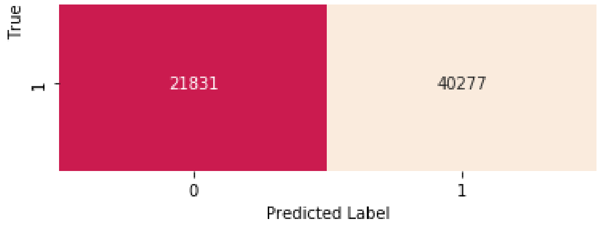
In [116]:

```
#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
%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[116]:

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





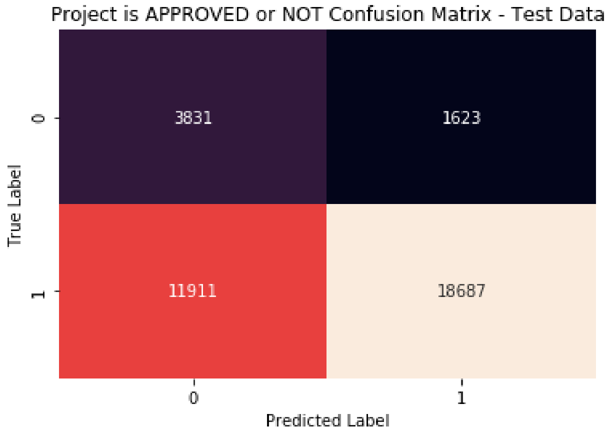
Observations for train data: Here we got 7726 - true positives, 40277 - true negatives, 21831 - false negatives, 3362 - false positives.

In [117]:

```
#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[117]:

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



Observations for Test data: Here we got 3831 - true positives, 18687 - true negatives, 11911 - false negatives, 1623 - false positives.

Appling Logistic Regression on feature SET 5

In [118]:

```
%%time
C_val=[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
log_C_val = []
auc_scores_train = []
auc_scores_cv = []
lr = LogisticRegression(penalty='l2', class_weight='balanced', n_jobs=-1)
parameters = {'C':C_val}
GridSearch_lr = GridSearchCV(lr, parameters, cv = 10, n_jobs=-1, scoring='roc_auc',return_train_sco
re=True,verbose=1)
GridSearch_lr.fit(X5_train, y_train)
auc_scores_train= GridSearch_lr.cv_results_['mean_train_score']
auc_scores_cv = GridSearch_lr.cv_results_['mean_test_score']
print('GridSearch_lr.best_estimator_: ', GridSearch_lr.best_estimator_)
print('GridSearch_lr.best_params_: ', GridSearch_lr.best_params_)
print('GridSearch_lr.best_score_: ', GridSearch_lr.best_score_)

for av in tqdm(C_val):
    b = np.log10(av)
    log_C_val.append(b)
```

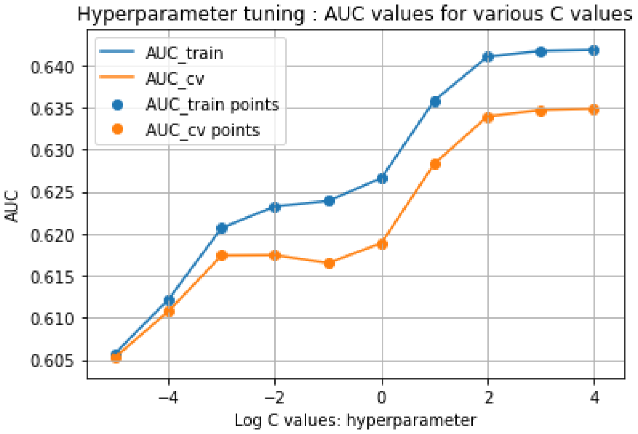
```
# Performance of model on Train data and Test data for each hyper parameter.
plt.plot(log_C_val, auc_scores_train, label='AUC_train')
plt.gca()
plt.plot(log_C_val, auc_scores_cv, label='AUC_cv')
plt.gca()
plt.scatter(log_C_val, auc_scores_train, label='AUC_train points')
plt.scatter(log_C_val, auc_scores_cv,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

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   15.8s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  5.5min finished
```

```
GridSearch_lr.best_estimator_ :  LogisticRegression(C=10000, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=-1, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
GridSearch_lr.best_params_ :  {'C': 10000}
GridSearch_lr.best_score_ :  0.634828913124095
```

```
100%|████████████████████████████████████████████████████████████████████████████████| 10/10
[00:00<00:00, 10000.72it/s]
```



```
Wall time: 5min 47s
```

Observations:

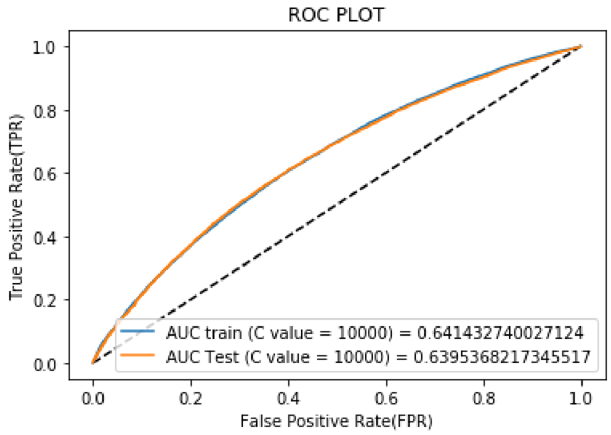
- C values are chosen from 0.00001 to 10000 and for the sake of graphical representation the C values are scaled down by applying a log function on them without losing the relationship with their corresponding AUC values.
- C = 10000 gives the maximum cv score, hence it is considered as the optimal C value.

In [119]:

```
a5=GridSearch_lr.best_params_['C']
#a5=10000
lr_opt = LogisticRegression(penalty='l2', C=a5, class_weight='balanced', n_jobs=-1)
lr_opt.fit(X5_train, y_train)
pred = lr_opt.predict(X5_Test)
acc = accuracy_score(y_Test, pred, normalize=True) * float(100)
print('\nThe optimal C value = {0}'.format(a5))
print('\nTest accuracy for (C value = {0}) is {1}%'.format(a5,acc))

fpr_train, tpr_train, thresholds = roc_curve(y_train, lr_opt.predict_proba(X5_train.tocsr()) [:,1])
fpr_Test, tpr_Test, thresholds = roc_curve(y_Test, lr_opt.predict_proba(X5_Test.tocsr())[:,1 ])
```

```
fpr_Test, tpr_Test, thresholds = roc_curve(y_Test, lr_opt.predict_proba(X5_Test.tocsr())[:,1])

#ROC plot
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr_train, tpr_train, label="AUC train (C value = {0}) = ".format(a5)+str(auc(fpr_train, t
pr_train)))
plt.plot(fpr_Test, tpr_Test, label="AUC Test (C value = {0}) = ".format(a5)+str(auc(fpr_Test, tpr_T
est)))
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 (C value = {0}) = ".format(a5), auc(fpr_train, tpr_train))
print("AUC value for Test data (C value = {0}) = ".format(a5), auc(fpr_Test, tpr_Test))
print("="*115)

AUC5 = auc(fpr_Test, tpr_Test)
pred0 = lr_opt.predict(X5_train)
pred2 = lr_opt.predict(X5_Test)
```

```
The optimal C value = 10000

Test accuracy for (C value = 10000) is 55.6002440918673%
```



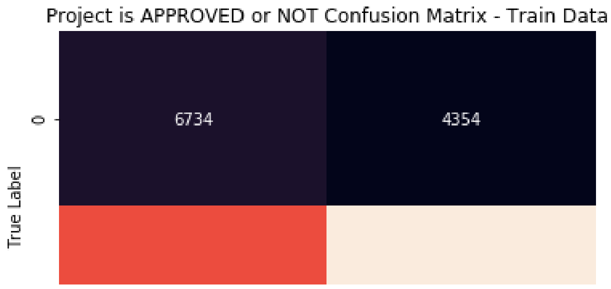
```
AUC value for train data (C value = 10000) =  0.641432740027124
AUC value for Test data (C value = 10000) =  0.6395368217345517
===================================================================================================
============
```

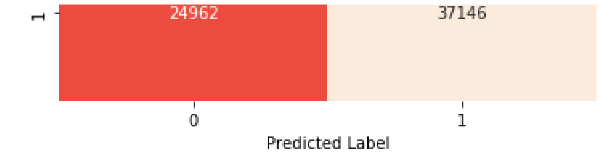
In [120]:

```
#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[120]:

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





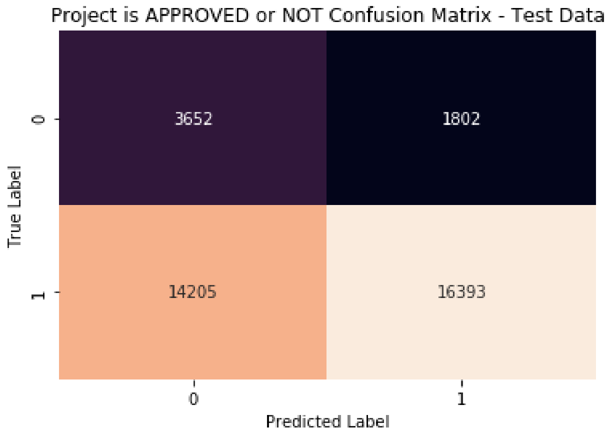
Observations for train data: Here we got 6734 - true positives, 37146 - true negatives, 24962 - false negatives, 4354 - false positives.

In [121]:

```
#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[121]:

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



Observations for Test data: Here we got 3652 - true positives, 16393 - true negatives, 14205 - false negatives, 1802 - false positives.

3. Conclusions

In [122]:

```
from prettytable import PrettyTable
pt = PrettyTable()
pt.field_names = ["Vectorizer", "Model", "C : Hyper Parameter", " Test AUC"]
pt.add_row(["BOW", "Logistic Regression", a1, AUC1])
pt.add_row(["TFIDF", "Logistic Regression", a2, AUC2])
pt.add_row(["AVGW2V", "Logistic Regression", a3, AUC3])
pt.add_row(["TFIDFW2V", "Logistic Regression", a4, AUC4])
pt.add_row(["Set 5", "Logistic Regression", a5, AUC5])
print(pt)
```

```
+------------+---------------------+---------------------+--------------------+
| Vectorizer |        Model        | C : Hyper Parameter |      Test AUC      |
+------------+---------------------+---------------------+--------------------+
|    BOW     | Logistic Regression |         0.01        | 0.6818311763415922 |
|   TFIDF    | Logistic Regression |         0.1         | 0.6861516973973363 |
|   AVGW2V   | Logistic Regression |         1000        | 0.7186880915470244 |
|  TFIDFW2V  | Logistic Regression |        10000        |  0.71280769110094  |
|   Set 5    | Logistic Regression |        10000        | 0.6395368217345517 |
+------------+---------------------+---------------------+--------------------+
```

SUMMARY:

1. In the 'Logistic Regression' model the AUC scores are higher for both 'Average Word 2 Vec' and 'TFIDF weighted Word 2 Vec' when compared to KNN and Naive Bayes model.
2. But 'Logistic Regression' model's space and time consumption is higher.
3. With the right C value and 'Word 2 Vec' a higher AUC score and better model performance can be achieved.