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 [1]:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%
b&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2
www.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly
ttps%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
```

.....
Mounted at /content/gdrive

In [2]:

```
%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 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.preprocessing import Normalizer
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')
import pdb
from sklearn.tree import DecisionTreeClassifier
import graphviz
from sklearn import tree
from graphviz import Source
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
```

Output hidden; open in https://colab.research.google.com to view.

1.1 Reading Data

In [3]:

```
Project_data = pd.read_csv('/content/gdrive/My Drive/Colab Notebooks/train_data.csv')
Resource_data = pd.read_csv('/content/gdrive/My Drive/Colab Notebooks/resources.csv')
print(Project_data.shape)
print(Resource_data.shape)
```

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

In [4]:

```
# 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[4]:

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

2. Decision Tree

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

In [5]:

```
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 [6]:

```
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
catogories_train = list(X_train['project_subject_categories'].values)
cat_list = []
for i in catogories_train:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
```

```
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

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

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

print(len(sorted_cat_dict_train))


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

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

```
9
```

1.3 preprocessing of `project_subject_subcategories`

In [7]:

```
# 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 [0]:

```
# 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 [0]:

```
# 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 [0]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've"
,\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "d
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [11]:

```
# 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 [00:39<00:00, 1853.47it/s]
100%|██████████| 36052/36052 [00:19<00:00, 1842.89it/s]
```

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

In [12]:

```
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 [00:01<00:00, 65144.86it/s]
100%|██████████| 36052/36052 [00:00<00:00, 65416.47it/s]
```

1.4 Preprocessing of `project_title`

In [13]:

```
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 [00:01<00:00, 43234.28it/s]
100%|██████████| 36052/36052 [00:00<00:00, 42955.58it/s]
```

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

In [14]:

```
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, 880981.73it/s]
100%|██████████| 36052/36052 [00:00<00:00, 914757.34it/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 [15]:

```
# 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 [16]:

```
# 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 [17]:

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

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

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

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

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


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

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

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

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

Prefix

In [18]:

```
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
prefix_catogories_train = list(X_train['teacher_prefix'].values)
prefix_list_train = []
for sent in prefix_catogories_train:
    sent = re.sub('[^A-Za-z0-9]+', ' ', str(sent))
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split())
    prefix_list_train.append(sent.lower().strip())
X_train['prefix_catogories'] = prefix_list_train
X_train.drop(['teacher_prefix'], axis=1, inplace=True)

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

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

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

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


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

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

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

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

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

project_grade_category

In [19]:

```
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
grade_catogories_train = list(X_train['project_grade_category'].values)
grade_list_train = []
for sent in grade_catogories_train:
    sent = sent.replace('-','_')
    sent = sent.replace(' ','_')
    # sent = re.sub('[^A-Za-z0-9]+', ' ', str(sent))
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split())
    grade_list_train.append(sent.lower().strip())

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

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

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


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

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


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

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

```
    # 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)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

1.5.2 Vectorizing Numerical features

Price and Quantity data

In [0]:

```
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 [21]:

```
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 [22]:

```
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 [23]:

```
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 [24]:

```
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 [25]:

```
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 [26]:

```
# https://www.geeksforgeeks.org/python-sentiment-analysis-using-vader/
sid = SentimentIntensityAnalyzer()
essays = X_train['essay']

sentiment_pos_essay_Train = []
sentiment_neg_essay_Train = []
sentiment_neut_essay_Train = []
sentiment_com_essay_Train = []

for essay in tqdm(essays):
    res = sid.polarity_scores(essay)
    sentiment_pos_essay_Train.append(res['pos'])
    sentiment_neg_essay_Train.append(res['neg'])
    sentiment_neut_essay_Train.append(res['neu'])
    sentiment_com_essay_Train.append(res['compound'])
X_train['sentiment_pos_essay_Train'] = sentiment_pos_essay_Train
X_train['sentiment_neg_essay_Train'] = sentiment_neg_essay_Train
X_train['sentiment_neut_essay_Train'] = sentiment_neut_essay_Train
X_train['sentiment_com_essay_Train'] = sentiment_com_essay_Train

essays = X_Test['essay']

sentiment_pos_essay_Test = []
sentiment_neg_essay_Test = []
sentiment_neut_essay_Test = []
sentiment_com_essay_Test = []

for essay in tqdm(essays):
    res = sid.polarity_scores(essay)
    sentiment_pos_essay_Test.append(res['pos'])
    sentiment_neg_essay_Test.append(res['neg'])
    sentiment_neut_essay_Test.append(res['neu'])
    sentiment_com_essay_Test.append(res['compound'])
X_Test['sentiment_pos_essay_Test'] = sentiment_pos_essay_Test
X_Test['sentiment_neg_essay_Test'] = sentiment_neg_essay_Test
X_Test['sentiment_neut_essay_Test'] = sentiment_neut_essay_Test
X_Test['sentiment_com_essay_Test'] = sentiment_com_essay_Test

sentiment_norm_pos = Normalizer(norm='l2', copy=False)
sentiment_norm_neg = Normalizer(norm='l2', copy=False)
sentiment_norm_neut = Normalizer(norm='l2', copy=False)
sentiment_norm_com = Normalizer(norm='l2', copy=False)

sentiment_norm_pos.fit(X_train['sentiment_pos_essay_Train'].values.reshape(1,-1))
sentiment_norm_neg.fit(X_train['sentiment_neg_essay_Train'].values.reshape(1,-1))
sentiment_norm_neut.fit(X_train['sentiment_neut_essay_Train'].values.reshape(1,-1))
sentiment_norm_com.fit(X_train['sentiment_com_essay_Train'].values.reshape(1,-1))

senti_pos_ess_Tr_norm = sentiment_norm_pos.transform(X_train['sentiment_pos_essay_Train'].values.r
eshape(1,-1))
senti_pos_ess_Tr_norm = (X_train['sentiment_pos_essay_Train'].values.reshape(-1,1))

senti_neg_ess_Tr_norm = sentiment_norm_neg.transform(X_train['sentiment_neg_essay_Train'].values.r
eshape(1,-1))
senti_neg_ess_Tr_norm = (X_train['sentiment_neg_essay_Train'].values.reshape(-1,1))

senti_neut_ess_Tr_norm =
sentiment_norm_neut.transform(X_train['sentiment_neut_essay_Train'].values.reshape(1,-1))
senti_neut_ess_Tr_norm = (X_train['sentiment_neut_essay_Train'].values.reshape(-1,1))

senti_com_ess_Tr_norm = sentiment_norm_com.transform(X_train['sentiment_com_essay_Train'].values.r
eshape(1,-1))
senti_com_ess_Tr_norm = (X_train['sentiment_com_essay_Train'].values.reshape(-1,1))


senti_pos_ess_Ts_norm =
sentiment_norm_pos.transform(X_Test['sentiment_pos_essay_Test'].values.reshape(1,-1))
senti_pos_ess_Ts_norm = (X_Test['sentiment_pos_essay_Test'].values.reshape(-1,1))
```

```
senti_neg_ess_Ts_norm =
sentiment_norm_neg.transform(X_Test['sentiment_neg_essay_Test'].values.reshape(1,-1))
senti_neg_ess_Ts_norm = (X_Test['sentiment_neg_essay_Test'].values.reshape(-1,1))

senti_neut_ess_Ts_norm = sentiment_norm_neut.transform(X_Test['sentiment_neut_essay_Test'].values.
reshape(1,-1))
senti_neut_ess_Ts_norm = (X_Test['sentiment_neut_essay_Test'].values.reshape(-1,1))

senti_com_ess_Ts_norm =
sentiment_norm_com.transform(X_Test['sentiment_com_essay_Test'].values.reshape(1,-1))
senti_com_ess_Ts_norm = (X_Test['sentiment_com_essay_Test'].values.reshape(-1,1))


print("Shape of senti_pos_ess_Tr_norm matrix after one hot encodig ",senti_pos_ess_Tr_norm.shape)
print("Shape of senti_neg_ess_Tr_norm matrix after one hot encodig ",senti_neg_ess_Tr_norm.shape)
print("Shape of senti_neut_ess_Tr_norm matrix after one hot encodig ",senti_neut_ess_Tr_norm.shape
)
print("Shape of senti_com_ess_Tr_norm matrix after one hot encodig ",senti_com_ess_Tr_norm.shape)
print("Shape of senti_pos_ess_Ts_norm matrix after one hot encodig ",senti_pos_ess_Ts_norm.shape)
print("Shape of senti_neg_ess_Ts_norm matrix after one hot encodig ",senti_neg_ess_Ts_norm.shape)
print("Shape of senti_neut_ess_Ts_norm matrix after one hot encodig ",senti_neut_ess_Ts_norm.shape
)
print("Shape of senti_com_ess_Ts_norm matrix after one hot encodig ",senti_com_ess_Ts_norm.shape)
```

```
100%|██████████| 73196/73196 [03:11<00:00, 382.47it/s]
100%|██████████| 36052/36052 [01:34<00:00, 379.76it/s]
```

```
Shape of senti_pos_ess_Tr_norm matrix after one hot encodig  (73196, 1)
Shape of senti_neg_ess_Tr_norm matrix after one hot encodig  (73196, 1)
Shape of senti_neut_ess_Tr_norm matrix after one hot encodig  (73196, 1)
Shape of senti_com_ess_Tr_norm matrix after one hot encodig  (73196, 1)
Shape of senti_pos_ess_Ts_norm matrix after one hot encodig  (36052, 1)
Shape of senti_neg_ess_Ts_norm matrix after one hot encodig  (36052, 1)
Shape of senti_neut_ess_Ts_norm matrix after one hot encodig  (36052, 1)
Shape of senti_com_ess_Ts_norm matrix after one hot encodig  (36052, 1)
```

2.3 Make Data Model Ready: encoding essay, and project_title

1.5.3 Vectorizing Text data

In [27]:

```
'''
# 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[27]:

```
'\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 [0]:

```
# 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('/content/gdrive/My Drive/Colab Notebooks/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

1.5.2.2 TFIDF vectorizer

In [29]:

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

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

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

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

TFIDF vectorizer for Project Title

In [30]:

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

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

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

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

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [0]:

```
# 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 [32]:

```
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
    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 [02:05<00:00, 584.08it/s]
```

```
100%|██████████| 36052/36052 [01:01<00:00, 586.90it/s]
```

```
36052
300
```

TFIDF weighted W2V on project_title

In [33]:

```
# 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 [00:02<00:00, 31525.69it/s]
100%|██████████| 36052/36052 [00:01<00:00, 32127.16it/s]
```

```
36052
300
```

1.5.4 Merging all the above features

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

Merging vectorised Train data

In [34]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
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, senti_pos_ess_Tr_norm
,senti_neg_ess_Tr_norm,senti_neut_ess_Tr_norm,senti_com_ess_Tr_norm))
print(X2_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,
senti_pos_ess_Tr_norm,senti_neg_ess_Tr_norm,senti_neut_ess_Tr_norm,senti_com_ess_Tr_norm))
print(X4_train.shape)

print(y_train.shape)
```

```
(73196, 16884)
(73196, 709)
(73196,)
```

Merging vectorised Test data

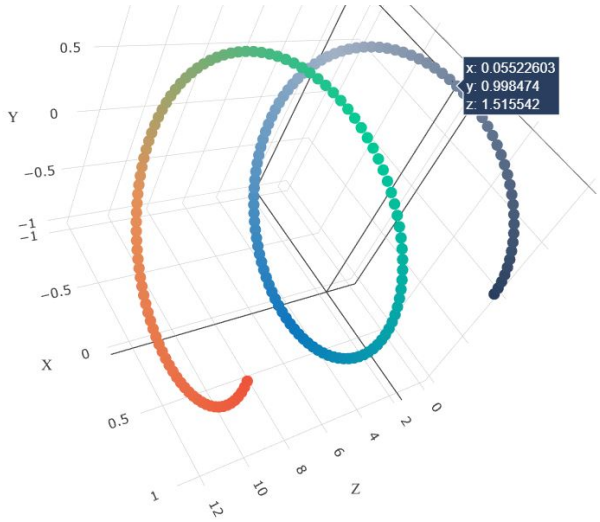
In [35]:

```
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,senti_pos_ess_Ts_norm,se
nti_neg_ess_Ts_norm,senti_neut_ess_Ts_norm,senti_com_ess_Ts_norm))
print(X2_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,
senti_pos_ess_Ts_norm,senti_neg_ess_Ts_norm,senti_neut_ess_Ts_norm,senti_com_ess_Ts_norm))
print(X4_Test.shape)


print(y_Test.shape)
```

```
(36052, 16884)
(36052, 709)
(36052,)
```

Assignment 8: DT

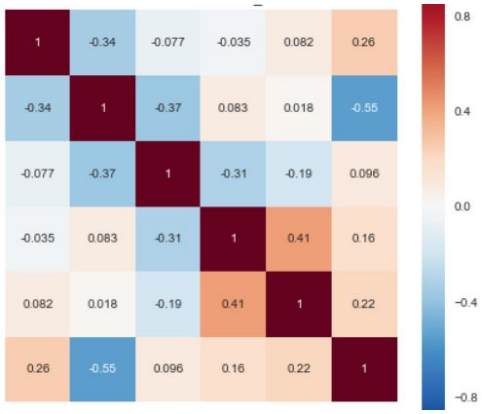
1. **Apply Decision Tree Classifier(DecisionTreeClassifier) on these feature sets**
    - Set 1: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
    - Set 2: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)
2. **The hyper paramter tuning (best `depth` in range [1, 5, 10, 50], and the best `min_samples_split` in range [5, 10, 100, 500])**
    - Find the best hyper parameter which will give the maximum AUC value
    - find the best hyper paramter using k-fold cross validation(use gridsearch cv or randomsearch cv)/simple cross validation data(you can write your own for loops refer sample solution)
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



with X-axis as **min_sample_split**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

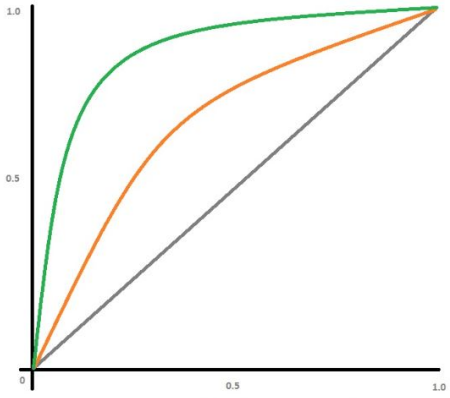
or

- 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



seaborn heat maps with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**

- You choose either of the plotting techniques out of 3d plot or heat map
- 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

	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

- Once after you plot the confusion matrix with the test data, get all the `false positive data points`

- Plot the WordCloud(https://www.geeksforgeeks.org/generating-word-cloud-python/) with the words of essay text of these `false positive data points`
- Plot the box plot with the `price` of these `false positive data points`
- Plot the pdf with the `teacher_number_of_previously_posted_projects` of these `false positive data points`

4. **Task 2:** For this task consider set-1 features. Select all the features which are having non-zero feature importance.You can get the feature importance using 'feature_importances_` (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html), discard the all other remaining features and then apply any of the model of you choice i.e. (Dession tree, Logistic Regression, Linear SVM), you need to do hyperparameter tuning corresponding to the model you selected and procedure in step 2 and step 3
   Note: when you want to find the feature importance make sure you don't use max_depth parameter keep it None.

5. You need to summarize the results at the end of the notebook, summarize it in the table format

Vectorizer	Model	Hyper parameter	AUC
BOW	Brute	7	0.78
TFIDF	Brute	12	0.79
W2V	Brute	10	0.78
TFIDFW2V	Brute	6	0.78

2.4 Appling Decision Tree on different kind of featurization as mentioned in the instructions

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

2.4.1 Applying Decision Trees on SET 1 - TFIDF

In [36]:

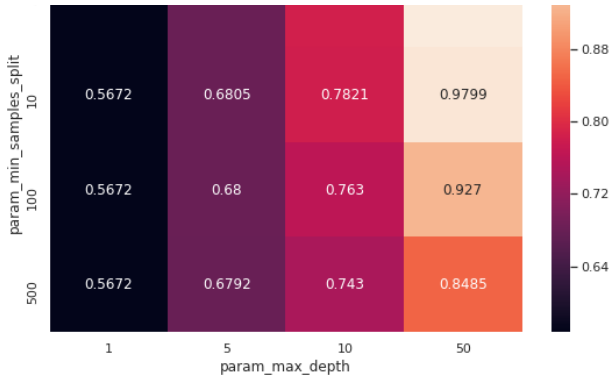
```
%%time
DTC = DecisionTreeClassifier(class_weight = 'balanced')
parameters = {'max_depth': [1, 5, 10, 50], 'min_samples_split': [5, 10, 100, 500]}
DTC_clf = GridSearchCV(DTC, parameters, cv=3, scoring='roc_auc',return_train_score=True)
DTC_clf.fit(X2_train, y_train)
print(DTC_clf.best_estimator_)
```

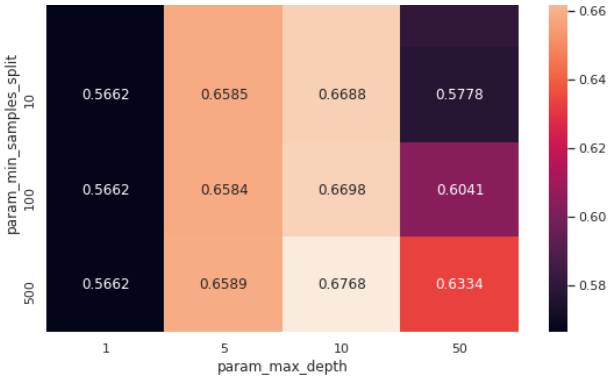
```
DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=10,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=500,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
CPU times: user 15min 1s, sys: 48.4 ms, total: 15min 1s
Wall time: 15min 1s
```

Hyper Parameter Tuning using Sns Heatmap

In [37]:

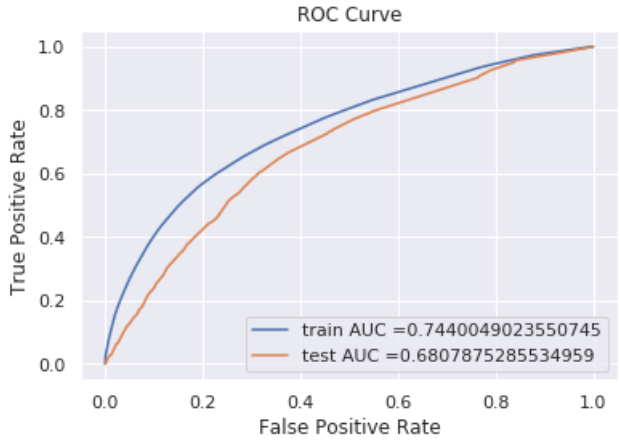
```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
sns.set()
DTC_cvresult = pd.DataFrame(DTC_clf.cv_results_).groupby(['param_min_samples_split',
'param_max_depth']).max().unstack()[['mean_test_score','mean_train_score']]
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(DTC_cvresult.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(DTC_cvresult.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Data')
ax[1].set_title('CV Data')
plt.show()
```





In [38]:

```
#Fitting Model to Hyper-Parameter Curve
#
https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc
ve

a1=(DTC_clf.best_params_['max_depth'])
a2=(DTC_clf.best_params_['min_samples_split'])
DTC_clf_opt=DecisionTreeClassifier(class_weight = 'balanced',max_depth=a1,min_samples_split=a2)
# for visulation
DTC_clf_opt.fit(X2_train, y_train)
#https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.
r_model.SGDClassifier.decision_function
y_train_pred = DTC_clf_opt.predict_proba(X2_train) [:,1]
y_test_pred = DTC_clf_opt.predict_proba(X2_Test) [:,1]

fpr_train, tpr_train, thresholds_train = roc_curve(y_train, y_train_pred)
fpr_test, tpr_test, thresholds_Test = roc_curve(y_Test, y_test_pred)

AUC1 = auc(fpr_test, tpr_test)

plt.plot(fpr_train, tpr_train, label="train AUC ="+str(auc(fpr_train, tpr_train)))
plt.plot(fpr_test, tpr_test, label="test AUC ="+str(auc(fpr_test, tpr_test)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.grid(True)
plt.show()
```



In [0]:

```
pred1 = DTC_clf_opt.predict(X2_train)
pred2 = DTC_clf_opt.predict(X2_Test)
```

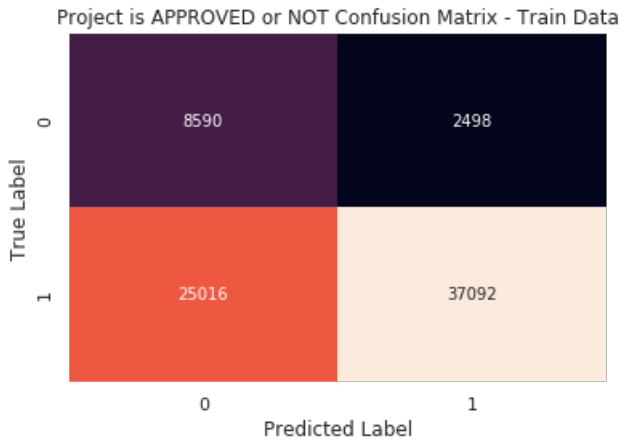
In [40]:

```
#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, pred1)
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[40]:

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



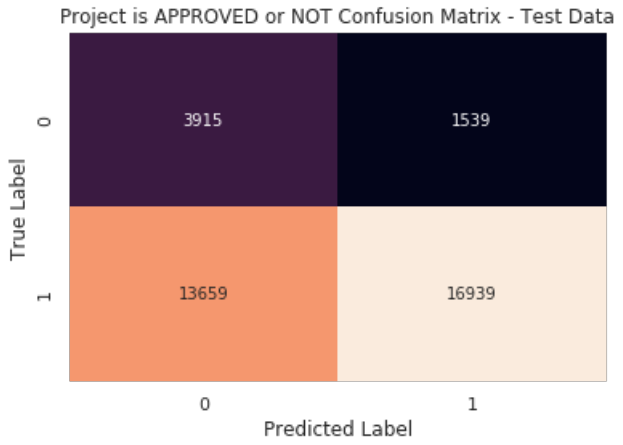
Observations for train data: Here we got 37092 - true positives, 8590 - true negatives, 25016 - false negatives, 2498 - false positives.

In [41]:

```
#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[41]:

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

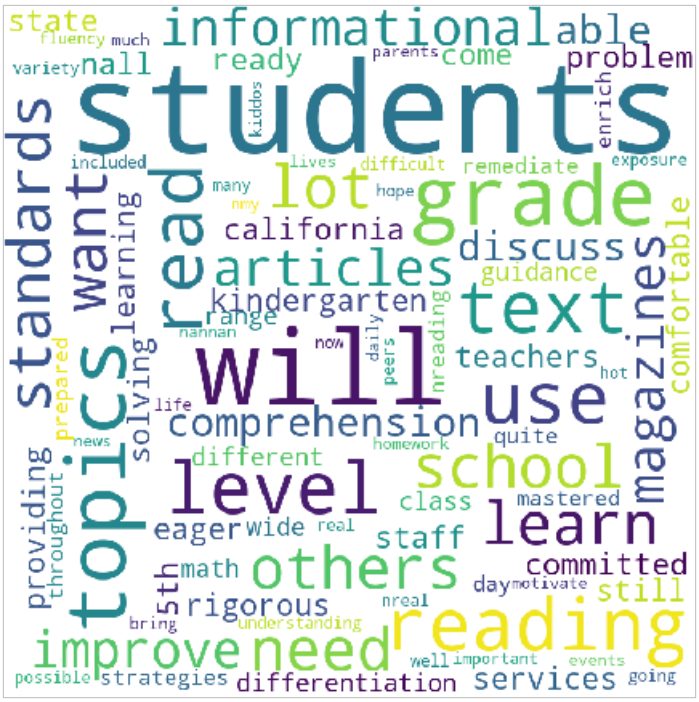


Observations for Test data: Here we got 16939 - true positives, 3915 - true negatives, 13659 - false negatives, 1539 - false positives.

Word Cloud

In [42]:

```
#https://github.com/pskadasi/DecisionTrees_DonorsChoose/blob/master/Copy_of_8_DonorsChoose_DT_(1)..

fpi = []
for i in range(len(y_Test)) :
  if (y_Test[i] == 0) & (pred2[i] == 1) :
    fpi.append(i)

X_T_Es_DS = pd.DataFrame(X_Test['essay'])

fp_essay1 = []
for i in fpi :
  #pdb.set_trace()
  fp_essay1.append(X_T_Es_DS['essay'][i])
# Word cloud of essay
from wordcloud import WordCloud, STOPWORDS
comment_words = ' '
stopwords = set(STOPWORDS)
for val in fp_essay1 :
  val = str(val)
  tokens = val.split()
for i in range(len(tokens)):
  tokens[i] = tokens[i].lower()
for words in tokens :
  comment_words = comment_words + words + ' '

wordcloud = WordCloud(width = 800, height = 800, background_color ='white', stopwords = stopwords,
min_font_size = 10).generate(comment_words)

plt.figure(figsize = (6, 6), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```



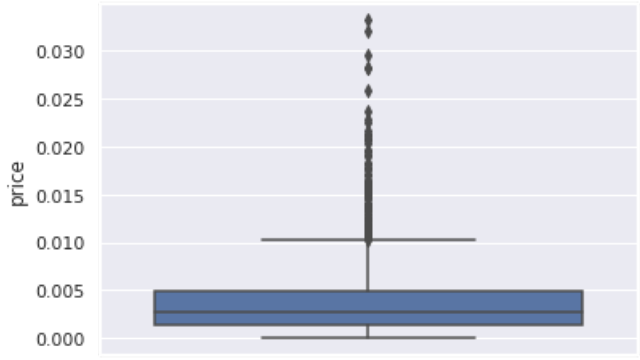
Box Plot of False Positives

In [43]:

```
col = X_Test.columns
X_Test_fp = pd.DataFrame(columns=col)
for i in fpi :
  X_Test_fp = X_Test_fp.append(X_Test.filter(items=[i], axis=0))
sns.boxplot(y='price', data=X_Test_fp)
```

Out[43]:

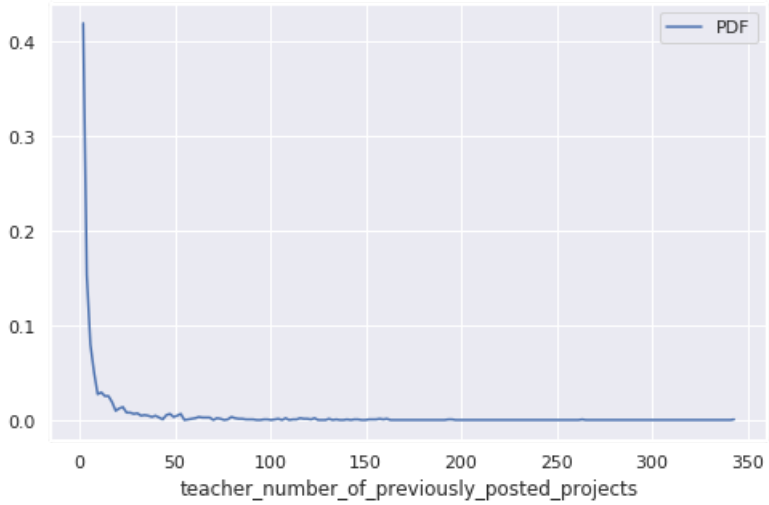
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd3231338d0>
```



PDF

In [44]:

```
plt.figure(figsize=(8,5))
counts, bin_edges = np.histogram(X_Test_fp['teacher_number_of_previously_posted_projects'],bins='a
uto', density=True)
pdf = counts/sum(counts)
pdfP, = plt.plot(bin_edges[1:], pdf)
plt.legend([pdfP], ["PDF"])
plt.xlabel('teacher_number_of_previously_posted_projects')
plt.show()
```



In [45]:

```
#Feature aggregation
f1=vectorizer_cat.get_feature_names()
f2=vectorizer_sub_cat.get_feature_names()
f3=vectorizer_sch.get_feature_names()
f4=vectorizer_grade.get_feature_names()
f5=vectorizer_prefix.get_feature_names()
f_es=vectorizer_essays_tfidf.get_feature_names()
f_ti=vectorizer_titles_tfidf.get_feature_names()

feature_agg_tfidf = f1 + f2 + f3 + f4 + f5 + f_es + f_ti
# p is price, q is quantity, t is teacher previous year projects
feature_agg_tfidf.append('price')
feature_agg_tfidf.append('quantity')
feature_agg_tfidf.append('prev_proposed_projects')
feature_agg_tfidf.append("title_word_count")
feature_agg_tfidf.append("essay_word_count")
feature_agg_tfidf.append('senti_pos_ess_Ts_norm')
feature_agg_tfidf.append('senti_neg_ess_Ts_norm')
feature_agg_tfidf.append('senti_neut_ess_Ts_norm')
feature_agg_tfidf.append('senti_com_ess_Ts_norm')

print(len(feature_agg_tfidf))
```

```
16884
```

2.4.1.1 Graphviz visualization of Decision Tree on TFIDF, SET 1

In [46]:

```
# https://medium.com/@rnbrown/creating-and-visualizing-decision-trees-with-python-f8e8fa394176
DTC_clf_opt=DecisionTreeClassifier(class_weight = 'balanced',max_depth=a1,min_samples_split=a2)
DTC_clf_opt.fit(X2_train, y_train)
dot_data = tree.export_graphviz(DTC_clf_opt, out_file=None, feature_names=feature_agg_tfidf)
graph = graphviz.Source(dot_data)
graph.render("Tfidf decision tree",view = True)
dot_data = StringIO()
export_graphviz(DTC_clf_opt, out_file=dot_data, filled=True, rounded=True, special_characters=True,
feature_names=feature_agg_tfidf,rotate=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

```
Output hidden; open in https://colab.research.google.com to view.
```

2.4.2 Applying Decision Trees on `Set 2 - TFIDF weighted W2V`

In [47]:

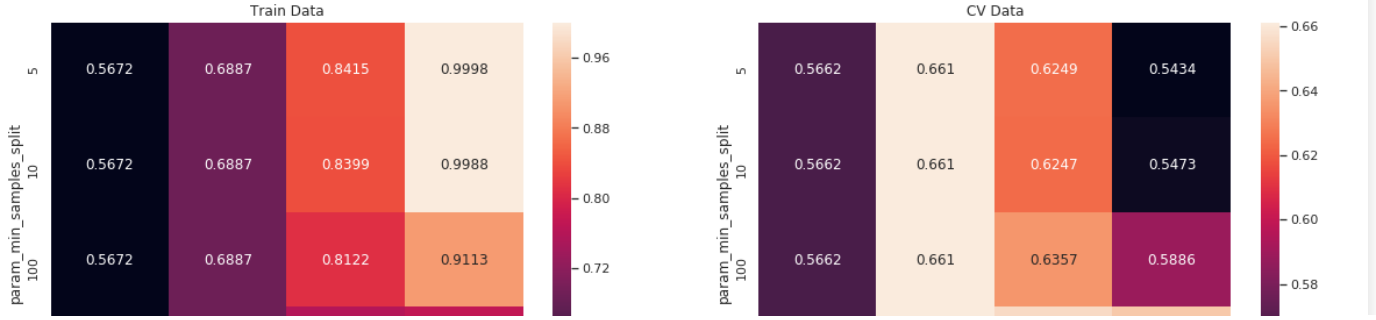
```
%%time
DTC = DecisionTreeClassifier(class_weight = 'balanced')
parameters = {'max_depth': [1, 5, 10, 50], 'min_samples_split': [5, 10, 100, 500]}
DTC_clf = GridSearchCV(DTC, parameters, cv=3, scoring='roc_auc',return_train_score=True)
DTC_clf.fit(X4_train, y_train)
print(DTC_clf.best_estimator_)
```

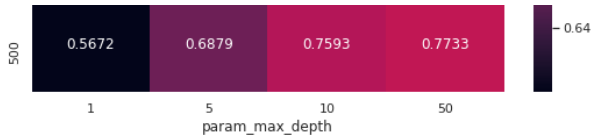
```
DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=5,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=500,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
CPU times: user 26min 4s, sys: 365 ms, total: 26min 5s
Wall time: 26min 5s
```

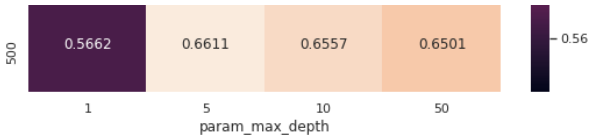
Hyper Parameter Tuning using Sns Heatmap

In [48]:

```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
sns.set()
DTC_cvresult = pd.DataFrame(DTC_clf.cv_results_).groupby(['param_min_samples_split',
'param_max_depth']).max().unstack()[['mean_test_score','mean_train_score']]

fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(DTC_cvresult.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(DTC_cvresult.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Data')
ax[1].set_title('CV Data')
plt.show()
```

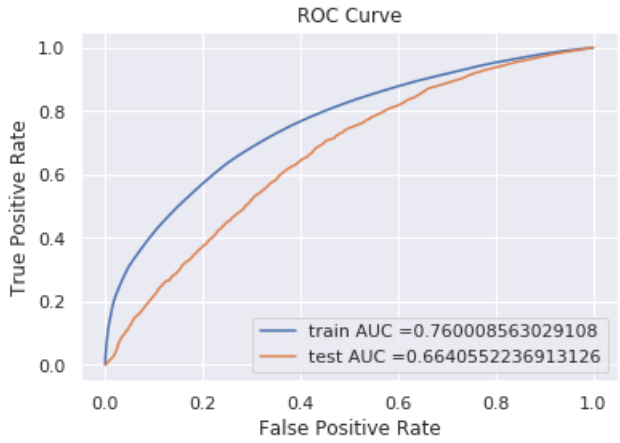






In [49]:

```
#Fitting Model to Hyper-Parameter Curve
#
https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.ro
ve

a3=(DTC_clf.best_params_['max_depth'])
a4=(DTC_clf.best_params_['min_samples_split'])
clfVDTC_clf_opt1=DecisionTreeClassifier(class_weight = 'balanced',max_depth=a3,min_samples_split=a
4)
# for visulation
DTC_clf_opt.fit(X4_train, y_train)
#https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.
r_model.SGDClassifier.decision_function
y_train_pred = DTC_clf_opt.predict_proba(X4_train) [:,1]
y_test_pred = DTC_clf_opt.predict_proba(X4_Test) [:,1]

fpr_train, tpr_train, tr_thresholds1 = roc_curve(y_train, y_train_pred)
fpr_test, tpr_test, te_thresholds1 = roc_curve(y_Test, y_test_pred)

AUC2 = auc(fpr_test, tpr_test)

plt.plot(fpr_train, tpr_train, label="train AUC ="+str(auc(fpr_train, tpr_train)))
plt.plot(fpr_test, tpr_test, label="test AUC ="+str(auc(fpr_test, tpr_test)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.grid(True)
plt.show()
```



In [0]:

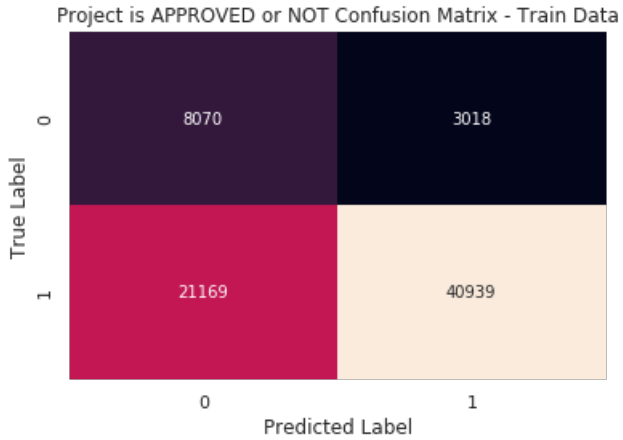
```
pred1 = DTC_clf_opt.predict(X4_train)
pred2 = DTC_clf_opt.predict(X4_Test)
```

In [51]:

```
#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, pred1)
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[51]:

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



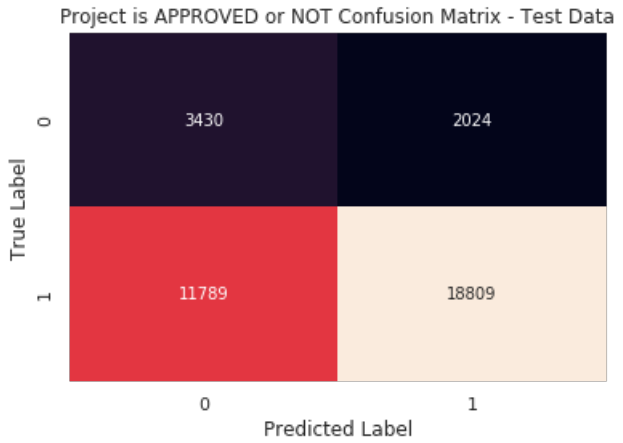
Observations for train data: Here we got 40939 - true positives, 8070 - true negatives, 21169 - false negatives, 3018 - false positives.

In [52]:

```
#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[52]:

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



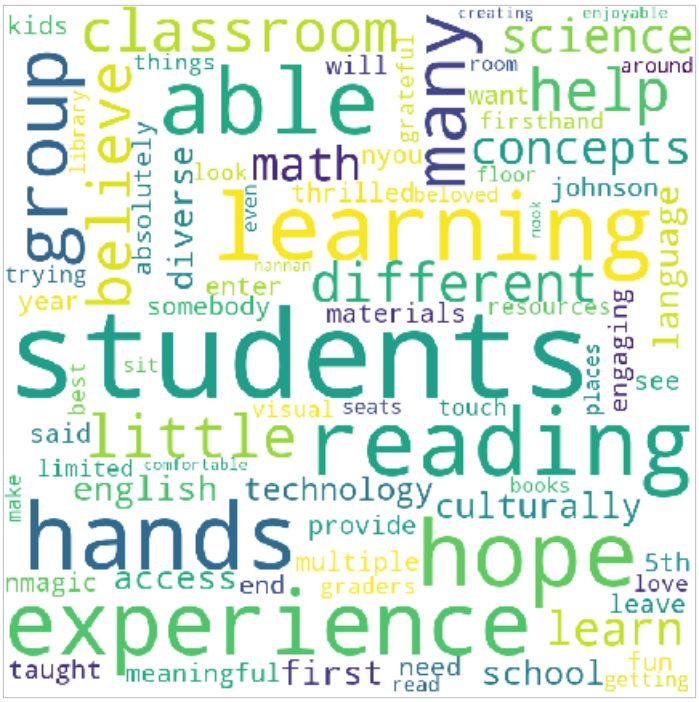
Observations for Test data: Here we got 18809 - true positives, 3430 - true negatives, 11789 - false negatives, 2024 - false positives.

Word Cloud

In [53]:

```
#https://github.com/pskadasi/DecisionTrees_DonorsChoose/blob/master/Copy_of_8_DonorsChoose_DT_(1).i

fpi = []
for i in range(len(y_Test)) :
  if (y_Test[i] == 0) & (pred2[i] == 1) :
    fpi.append(i)
```

```
X_T_Es_DS = pd.DataFrame(X_Test['essay'])
fp_essay1 = []
for i in fpi :
  #pdb.set_trace()
  fp_essay1.append(X_T_Es_DS['essay'][i])

# Word cloud of essay
from wordcloud import WordCloud, STOPWORDS
comment_words = ' '
stopwords = set(STOPWORDS)
for val in fp_essay1 :
  val = str(val)
  tokens = val.split()
for i in range(len(tokens)):
  tokens[i] = tokens[i].lower()
for words in tokens :
  comment_words = comment_words + words + ' '

wordcloud = WordCloud(width = 800, height = 800, background_color ='white', stopwords = stopwords,
min_font_size = 10).generate(comment_words)

plt.figure(figsize = (6, 6), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```



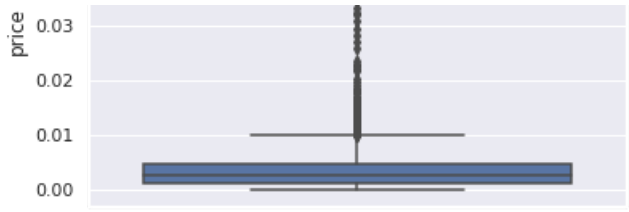
Box Plot

In [54]:

```
cols = X_Test.columns
X_Test_fp = pd.DataFrame(columns=cols)
for i in fpi :
  X_Test_fp = X_Test_fp.append(X_Test.filter(items=[i], axis=0))
sns.boxplot(y='price', data=X_Test_fp)
```

Out[54]:

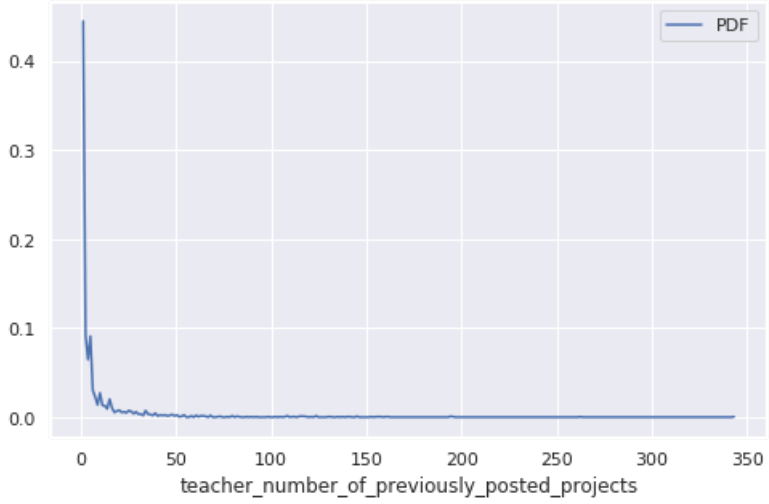
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd3e2473860>
```



PDF

In [55]:

```
plt.figure(figsize=(8,5))
counts, bin_edges = np.histogram(X_Test_fp['teacher_number_of_previously_posted_projects'],bins='a
uto', density=True)
pdf = counts/sum(counts)
pdfP, = plt.plot(bin_edges[1:], pdf)
plt.legend([pdfP], ["PDF"])
plt.xlabel('teacher_number_of_previously_posted_projects')
plt.show()
```



2.4.3 Task-2 -> Applying Decision Tree on Non-zero important features of Set-1

In [57]:

```
clfV3=DecisionTreeClassifier(class_weight = 'balanced',max_depth=None,min_samples_split=a2)
clfV3.fit(X2_train, y_train)
FI=clfV3.feature_importances_ # Getting the features based on feature importance
NFI = np.nonzero(FI) # Getting the non-zero features
df = pd.DataFrame(X2_train.toarray())
#https://www.geeksforgeeks.org/numpy-nonzero-in-python/

# Coverting the Non-zero important features into a new dataset
ls=[]
New_DS= pd.DataFrame()
NFI1=list(NFI)
for i in NFI1:
  ls.append(i)
for j in ls:
  for k in j:
      m=int(k)
      fn=feature_agg_tfidf[m]
      #print(df[m])
      t_ls=list(df[m])
      #pdb.set_trace()
      New_DS[fn] = t_ls
print(New_DS.head(10))

df_ts = pd.DataFrame(X2_Test.toarray())
ls_ts=[]
New_DS_ts= pd.DataFrame()
```

```
NFI1_ts=list(NFI1)
for i in NFI1_ts:
  ls_ts.append(i)
for j in ls_ts:
  for k in j:
      m=int(k)
      fn=feature_agg_tfidf[m]
      #print(df[m])
      t_ls_ts=list(df_ts[m])
      #pdb.set_trace()
      New_DS_ts[fn] = t_ls_ts
print(New_DS_ts.head(10))
print(New_DS.shape)
print(New_DS_ts.shape)
```

```
   Extracurricular  Literacy  ...  senti_neut_ess_Ts_norm  senti_com_ess_Ts_norm
0              0.0       0.0  ...                0.003725               0.003755
1              0.0       1.0  ...                0.003665               0.003725
2              0.0       1.0  ...                0.003997               0.003734
3              0.0       0.0  ...                0.004075               0.003445
4              0.0       1.0  ...                0.003743               0.003761
5              0.0       0.0  ...                0.003854               0.003778
6              0.0       1.0  ...                0.004047               0.003680
7              0.0       0.0  ...                0.003946               0.003728
8              0.0       0.0  ...                0.003937               0.003587
9              0.0       0.0  ...                0.003868               0.003785

[10 rows x 1187 columns]
   Extracurricular  Literacy  ...  senti_neut_ess_Ts_norm  senti_com_ess_Ts_norm
0              0.0       0.0  ...                0.005320               0.005375
1              0.0       0.0  ...                0.004690               0.005389
2              0.0       0.0  ...                0.004847               0.005401
3              0.0       0.0  ...                0.005071               0.005375
4              0.0       0.0  ...                0.005012               0.005406
5              0.0       0.0  ...                0.004979               0.005372
6              0.0       0.0  ...                0.005340               0.005389
7              0.0       0.0  ...                0.004860               0.005403
8              0.0       0.0  ...                0.005090               0.005370
9              0.0       0.0  ...                0.005025               0.005343

[10 rows x 1187 columns]
(73196, 1187)
(36052, 1187)
```

In [58]:

```
# Non-zero important features of Set-1
for j in ls:
  for k in j:
      m=int(k)
      fn=feature_agg_tfidf[m]
      print(fn)
```

```
Extracurricular
Literacy
wv
nc
fl
10
22
340
45
5th
600
6th
95
ability
abstract
academic
access
accessory
accommodations
accompanying
achieve
```

across
act
active
actively
activities
actual
actually
add
addition
additional
additionally
affluent
age
air
allow
along
aloud
already
also
although
always
amazing
amount
ample
analysis
analyze
anchor
another
answer
anything
app
applied
appreciated
approach
approaches
appropriate
approximately
apps
architecture
around
arrived
art
artists
arts
artwork
artworks
asia
ask
asked
asking
assess
assigning
assignments
assist
attack
attain
attends
attitude
authors
available
average
awareness
back
background
backgrounds
bags
baking
balance
ball
balls
band
basis
basketball
basketballs
bass
battle
bean

beanbag
beanbags
beat
became
become
becoming
begin
begins
behalf
behavioral
behaviors
believe
bell
belly
benches
biased
big
bilingual
bingo
bins
biographies
bit
blessed
board
boards
book
books
borrow
bottom
bouncing
bows
brightest
brings
budding
build
builders
buoyancy
busy
cafe
calculators
call
called
calories
cam
canvas
capable
card
cards
care
caretakers
carpet
carry
carrying
cart
carts
cash
catching
cause
causing
center
centers
central
century
cerebral
certain
chair
chairs
challenged
challenges
challenging
change
changes
channel
charged
chart
child
children

choice
choose
chromebook
chromebooks
circle
citizens
clarinet
classic
classroom
clean
clipboards
coaching
code
collaborate
collaboratively
collection
college
colleges
colored
colors
combining
come
comes
coming
commitment
common
communication
communities
complex
comprised
computers
concentrations
concepts
conduct
conducted
connect
consisting
contact
continue
conversation
conversations
costs
costumes
counselor
counting
craft
crayons
create
creates
creation
creations
critique
crucial
cry
curiosity
curriculum
cushions
cuts
cycle
daily
dance
dancing
dangle
date
days
deaf
decided
decreases
deepen
deeper
delayed
delve
department
depend
depending
deployed
desired

desk
desks
desktops
despise
develop
development
developmentally
devices
dice
diet
different
difficult
digital
direct
directions
disabilities
discover
disorders
disruptive
distinct
distract
distracting
district
document
dominant
donating
donation
donors
donorschoose
dot
dots
dove
dramatic
dress
driven
driving
drums
duct
duties
eagerness
earliest
earnings
ease
easier
easily
easy
eating
education
educators
effects
effort
eight
either
ela
electricity
eleven
email
emotions
employment
enables
enabling
end
endless
endure
energetic
energy
engage
engaging
engineer
engineering
english
enhance
enjoyable
enjoys
enough
enrich
enroll

enrolled
entered
entering
enthusiastic
enthusiastically
entire
enviornment
enviroment
environment
environments
erase
erasers
escape
esl
essentials
ethnic
europe
even
events
ever
every
everyday
everything
exam
exceptional
exceptionalities
excitement
execute
executive
exercise
exercises
exhibitions
expected
experiences
experiments
explore
explorers
exploring
exposing
express
eye
eyes
face
facets
facilitate
facilitating
factor
facts
failing
families
family
fan
farmer
farming
fast
faster
favorite
features
feel
feet
fiction
fields
fifth
figure
fill
finally
find
finding
fingertips
fire
fires
fit
fitbits
fitness
flexibility
floor
fluent

fluently
focus
folder
folders
following
food
foot
football
format
formats
former
forward
foster
founding
four
fourth
freedom
fresh
friendly
friends
fruits
fullest
fully
fun
function
funded
fundraising
future
game
games
gap
gaps
garden
gear
general
generated
germ
gets
giant
gifted
girls
give
global
globe
gloves
glue
go
goal
goals
goes
goggles
going
golf
good
google
grab
graduate
grandparents
graphic
great
greatly
greatness
greener
greetings
groups
growth
guides
guitar
hand
handle
hands
hang
happen
happiness
hardest
head
headphones

hear
heart
hearts
heating
help
hence
highlighting
history
hits
hold
home
homeless
homework
hone
hopefully
hopes
horizons
hot
hour
hours
however
hugging
hundreds
hungry
hurt
ideas
identification
identified
identify
identifying
image
imagination
imagine
implementation
implementing
important
importantly
impoverished
improvement
include
inclusive
income
incorporate
increased
increasingly
incredibly
indicator
individual
individualized
indoors
influenced
information
informational
inherited
injustice
ink
inquiry
inside
inspire
inspired
inspiring
instilling
instruction
integrated
intelligent
intensive
interested
interests
introduce
invite
involve
ipad
ipads
isolation
issue
issues
item

items
jazz
joined
judy
juvenile
keep
keeping
key
keyboards
kick
kiddos
kidney
kids
kindergarten
kindness
kits
knowing
knowledge
ks
labs
lack
lacrosse
laptops
large
larger
last
leader
learn
learners
learning
learns
leaves
lecture
left
legacy
legos
lenses
let
lets
letter
letters
leveled
librarian
library
life
lifelong
lights
like
lincoln
listening
literacy
literature
little
live
locate
located
logic
long
look
looks
lot
louisiana
love
loved
loving
low
lower
lowest
lunch
luxury
machine
made
magic
magical
magnetic
magnets
maintain

make
makes
manage
mandarin
manipulation
many
map
maps
marble
markers
market
mascot
master
mat
material
materials
math
mathematicians
mathematics
mats
may
meals
measuring
meet
meeting
meetings
meets
mexican
microorganisms
middle
might
miles
mind
minds
minimum
missing
mistakes
mobymax
modules
moms
money
monitor
mornings
motivate
motivated
movement
multiple
multiplication
music
musicians
must
names
nannan
native
near
necessary
need
needs
neighborhood
neighborhoods
new
newsletter
nex
nights
non
norm
normal
north
note
nothing
number
nutrition
observations
odd
offer
offering
often

oil
oklahoma
ones
online
open
opportunity
options
order
organized
organizers
ot
outdoor
outside
outstanding
overtime
ownership
packing
packs
paint
painted
paired
paper
parents
participation
parts
passion
passionate
past
path
pe
peer
pencil
pencils
per
percentage
percussion
perfect
performance
performing
perhaps
persevere
person
perspective
phonics
physical
physics
picked
pictures
pieces
pillows
place
planning
plants
playing
pocket
points
poorest
popular
portfolios
positive
positives
possess
possibilities
possible
post
posture
potential
poverty
powerful
practice
prefer
prekindergarten
prepared
preparing
preschool
presence
presentation

presentations
presenters
pressure
pressures
prevent
prezi
pride
printer
prison
privileged
process
processes
productivity
proficient
program
programming
project
projector
projects
promote
properties
protection
provided
provides
pump
puppets
purchased
putting
puzzle
puzzles
qualifies
quality
quickly
quite
race
range
rarely
read
readers
ready
real
realistic
realize
really
reason
recall
receive
received
receiving
recent
recess
recieve
reciting
recognize
recording
reduced
reflection
refugees
regardless
relates
relevant
religious
remind
replace
replaced
reply
reported
representing
represents
requesting
require
research
resident
resilient
resources
responsibility
results

review
revolve
rewards
rich
rise
risk
robot
robotics
rope
ropes
rouge
rug
ruler
run
safer
sanctuary
sand
school
schooler
schools
science
scientists
scissors
score
scores
season
seat
seats
sections
selections
self
send
sense
sensory
sent
sentence
sentences
separate
separated
sequence
series
sets
setting
settings
seuss
seven
several
sheets
shelters
show
side
significantly
simple
simply
sit
sixth
skill
skills
slides
smiling
snack
snacks
soccer
social
socio
socioeconomically
soft
solve
solving
something
sound
south
spanish
spans
speak
speaker
speaking

speaking
specially
spend
spent
spirit
sponges
sports
springs
stability
stamps
standard
standing
stands
starting
state
stationary
stay
stem
sticks
stimulated
stimulating
stool
stools
storage
store
stories
storyworks
strengthen
strengths
strong
stronger
strongly
structure
stuck
student
students
studies
styles
subject
succeed
successful
suffering
suits
supplemental
sure
symbols
symptoms
table
tables
tablet
tablets
taking
talented
tap
teach
teacher
teachers
teaches
teams
teamwork
tech
technology
temperature
testing
tests
texas
text
texts
thier
think
thought
thoughtful
thousands
three
thriving
thru
tiles
time

time
timely
today
together
tomorrow
tone
tool
topic
topics
toward
town
traditional
trait
trampoline
transform
transformation
transforming
transportation
trays
tremendously
truly
trying
tubes
turns
tutoring
tutors
twelve
twice
two
types
uniform
unique
unit
university
unlimited
unstable
upright
urban
us
use
used
using
utilize
valuable
values
variety
various
versus
via
visual
vocabulary
volleyball
wall
walls
want
wanted
wash
watch
watched
water
watercolor
ways
weakness
weave
weekly
weighted
well
went
west
wheels
whenever
wherever
whether
white
whiteboards
whose
wild
winters

writers
wipe
wipes
wire
wisdom
wishing
within
witness
wobble
women
word
words
working
works
world
worn
worries
worry
would
writer
writing
written
years
young
younger
youth
yummy
zone
abc
about
along
artists
arts
audio
autistic
be
bees
bonanza
books
brains
bringing
calm
centers
century
chromebook
class
classroom
communicate
composting
diversity
do
earth
easel
education
engaged
english
esl
events
eyes
flexible
full
fun
hands
hard
have
healthy
help
history
hot
increase
ipads
is
key
kindergarten
lead
learning
let
library

```
library
little
love
magic
makerspace
making
mentor
motivate
movers
multi
my
necessities
need
needs
novels
nutrition
on
our
perfect
physical
place
places
play
playing
positive
practice
program
reading
right
rugs
seating
sensory
snacks
solvers
solving
sticks
storage
students
success
successful
supplies
take
technology
text
through
time
tools
top
unique
use
way
welcome
wiggle
will
with
world
price
quantity
prev_proposed_projects
title_word_count
senti_pos_ess_Ts_norm
senti_neut_ess_Ts_norm
senti_com_ess_Ts_norm
```

Applying Gridsearchcv on the 'Non-zero important features - dataset'

In [59]:

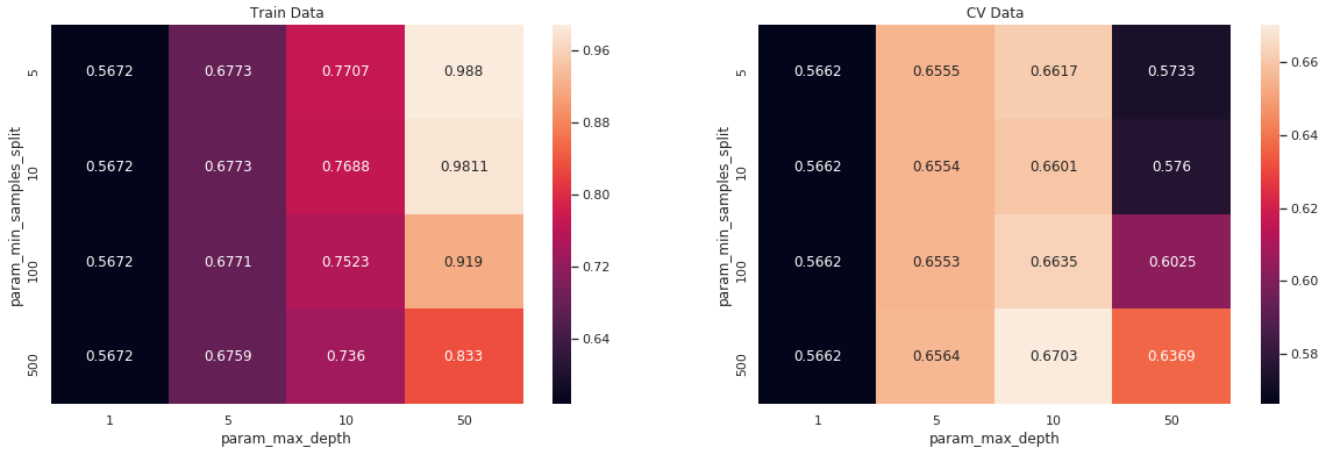
```
%%time
DTC = DecisionTreeClassifier(class_weight = 'balanced')
parameters = {'max_depth': [1, 5, 10, 50], 'min_samples_split': [5, 10, 100, 500]}
DTC_clf = GridSearchCV(DTC, parameters, cv=3, scoring='roc_auc',return_train_score=True)
DTC_clf.fit(New_DS, y_train)
print(DTC_clf.best_estimator_)
```

```
DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=10,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=500,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
CPU times: user 6min 31s, sys: 504 ms, total: 6min 31s
Wall time: 6min 31s
```

Hyper Parameter Tuning using Sns Heatmap

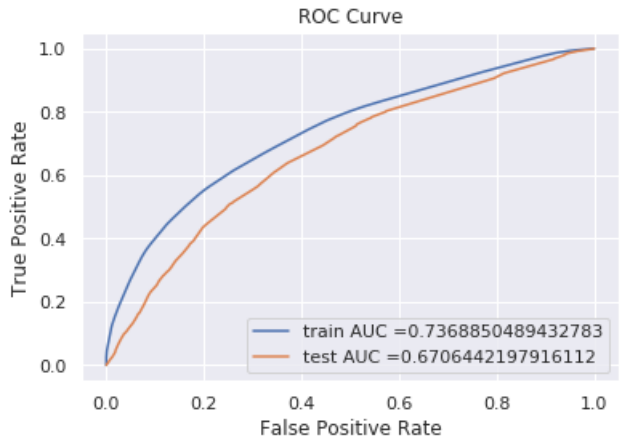
In [60]:

```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
sns.set()
DTC_cvresult = pd.DataFrame(DTC_clf.cv_results_).groupby(['param_min_samples_split',
'param_max_depth']).max().unstack()[['mean_test_score','mean_train_score']]

fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(DTC_cvresult.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(DTC_cvresult.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Data')
ax[1].set_title('CV Data')
plt.show()
```



In [61]:

```
#Fitting Model to Hyper-Parameter Curve
#
https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc
ve

a5=(DTC_clf.best_params_['max_depth'])
a6=(DTC_clf.best_params_['min_samples_split'])
DTC_clf_opt=DecisionTreeClassifier(class_weight = 'balanced',max_depth=a5,min_samples_split=a6)
# for visulation
DTC_clf_opt.fit(New_DS, y_train)
#https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.
r_model.SGDClassifier.decision_function
y_train_pred = DTC_clf_opt.predict_proba(New_DS) [:,1]
y_test_pred = DTC_clf_opt.predict_proba(New_DS_ts) [:,1]

fpr_train, tpr_train, tr_thresholds1 = roc_curve(y_train, y_train_pred)
fpr_test, tpr_test, te_thresholds1 = roc_curve(y_Test, y_test_pred)

AUC3 = auc(fpr_test, tpr_test)

plt.plot(fpr_train, tpr_train, label="train AUC ="+str(auc(fpr_train, tpr_train)))
plt.plot(fpr_test, tpr_test, label="test AUC ="+str(auc(fpr_test, tpr_test)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.grid(True)
plt.show()
```



In [0]:

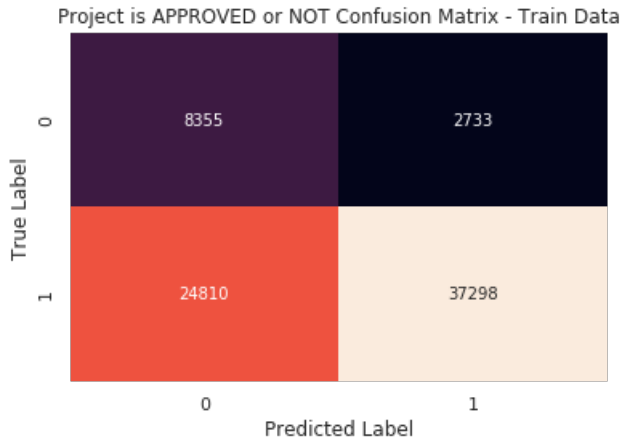
```
pred1 = DTC_clf_opt.predict(New_DS)
pred2 = DTC_clf_opt.predict(New_DS_ts)
```

In [63]:

```
#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, pred1)
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[63]:

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



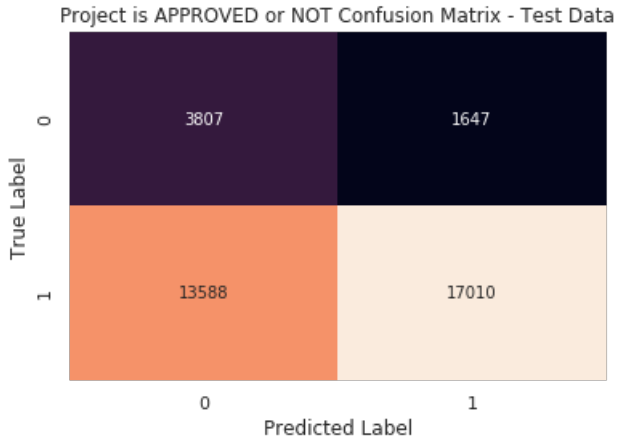
Observations for train data: Here we got 37298 - true positives, 8355 - true negatives, 24810 - false negatives, 2733 - false positives.

In [64]:

```
#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[64]:

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



Observations for Test data: Here we got 17010 - true positives, 3807 - true negatives, 13588 - false negatives, 1647 - false positives.

Graphviz visualization of Decision Tree on TFIDF Weighted W2V for Set-3 (Non-zero important features of set-1)

In [65]:

```
# https://medium.com/@rnbrown/creating-and-visualizing-decision-trees-with-python-f8e8fa394176
DTC_clf_opt=DecisionTreeClassifier(class_weight = 'balanced',max_depth=a5,min_samples_split=a6)
DTC_clf_opt.fit(New_DS, y_train)
dot_data = StringIO()
export_graphviz(DTC_clf_opt, out_file=dot_data, filled=True, rounded=True, special_characters=True,
feature_names=New_DS.columns,rotate=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

```
Output hidden; open in https://colab.research.google.com to view.
```

3. Conclusions

In [66]:

```
from prettytable import PrettyTable
pt = PrettyTable()
pt.field_names = ["Vectorizer", "Model", "max_depth", "min_samples_split","Test AUC"]
pt.add_row(["TFIDF", "Decision Tree", a1, a2, AUC1])
pt.add_row(["TFIDFW2V", "Decision Tree", a3, a4, AUC2])
pt.add_row(["Task-2 (Non-zero Important features)", "Decision Tree", a5, a6, AUC3])
print(pt)
```

```
+-------------------------------------+---------------+-----------+-------------------+-----------
------+
|              Vectorizer             |     Model     | max_depth | min_samples_split |      Test
UC      |
+-------------------------------------+---------------+-----------+-------------------+-----------
------+
|                TFIDF                | Decision Tree |     10    |        500        | 0.68078752
534959 |
|               TFIDFW2V              | Decision Tree |     5     |        500        |
0.6640552236913126 |
| Task-2 (Non-zero Important features) | Decision Tree |     10    |        500        |
0.6706442197916112 |
+-------------------------------------+---------------+-----------+-------------------+-----------
------+
```

SUMMARY:

1. 'Decision Tree' model's space and time consumption is lower compared to models like KNN, SVM

1. ~~Decision Tree models space and time consumption is lower compared to models like KNN, SVM.~~
2. The Test AUC score is moderately low.
3. The application of 'Word to Vector' concept on the TFIDF model didnot improve the Test AUC score.
4. Just the Non-zero important features gave almost the same Test AUC score as the TFIDF model.
5. Graphs - 'Box plot' and 'PDF' are not created for the Task-2's dataset as the features - 'price' and 'teacher_number_of_previously_posted_projects' are not selected as Non-zero important features.