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
import scipy
# 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.model_selection import cross_val_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')
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from mpl_toolkits.mplot3d import Axes3D
from xgboost import XGBClassifier
import pdb
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
```

1.1 Reading Data

In [5]:

```
Project_data = pd.read_csv('/content/gdrive/My Drive/Colab Notebooks/20k_train_data.csv')
Resource_data = pd.read_csv('/content/gdrive/My Drive/Colab Notebooks/resources.csv')
#Bk=Project_data
#print(Bk.shape)
print(Project_data.shape)
print(Resource_data.shape)
```

```
(20000, 17)
(1541272, 4)
```

In [6]:

```
# 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[6]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_
3287	159755	p147002	6ada7036aeb258d3653589d1f2a5b815	Mrs.	CA	2016-01-05 02:02:00	Grades 3-5	
19437	146532	p024903	55f60249d65840ee198285acdc455838	Mrs.	CA	2016-01-05 02:57:00	Grades 3-5	Math &

1.2 preprocessing of `project_subject_categories`

In [0]:

```
y = Project_data['project_is_approved'].values
# Project_data.drop(['project_is_approved'], axis=1, inplace=True)
lpd = len(Project_data)
ys = np.zeros(lpd, dtype=np.int32)
X = Project_data
```

In [8]:

```
#Spliting the Dataset into three Train and Test
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, y, test_size=0.33, random_state=0, stratify=
ys)

print('Shape of the X_Train data is {0} and Y_Train data is:
{1}'.format(X_Train.shape,Y_Train.shape[0]))

print('Shape of the X_Test data is  {0} and Y_Test data is : {1}'.format(X_Test.shape,Y_Test.shape[
0]))
```

```
Shape of the X_Train data is (13400, 17) and Y_Train data is: 13400
Shape of the X_Test data is  (6600, 17) and Y_Test data is : 6600
```

In [9]:

```
# 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

#************************************************************Train
Data*****************************************************************************************
catogories = list(X_Train['project_subject_categories'].values)
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
```

```
        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))


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

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 [10]:

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

```
    sub_cat_list.append(temp.strip())

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))


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

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 [14]:

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

#-----------------------------------PreProcessing of Essays in Train data set-----------------------
----------------------
preprocessed_essays_Train = []
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())
# pdb.set_trace()


#-----------------------------------PreProcessing of Essays in Test data set-----------------------
-----------------------
preprocessed_essays_Test = []
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())
# pdb.set_trace()
```

```
100%|██████████| 13400/13400 [00:07<00:00, 1856.70it/s]
100%|██████████| 6600/6600 [00:03<00:00, 1879.10it/s]
```

In [15]:

```
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%|██████████| 13400/13400 [00:00<00:00, 67576.45it/s]
100%|██████████| 6600/6600 [00:00<00:00, 68931.84it/s]
```

1.4 Preprocessing of `project_title`

In [16]:

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

#----------------------------------PreProcessing of Project Title in Train data set----------------
------------------------
preprocessed_titles_Train = []
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())
# pdb.set_trace()


#----------------------------------PreProcessing of Project Title in Test data set----------------
------------------------
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())
# pdb.set_trace()
```

```
100%|██████████| 13400/13400 [00:00<00:00, 42797.52it/s]
100%|██████████| 6600/6600 [00:00<00:00, 43281.55it/s]
```

In [17]:

```
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%|██████████| 13400/13400 [00:00<00:00, 965815.02it/s]
100%|██████████| 6600/6600 [00:00<00:00, 1004806.04it/s]
```

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 [0]:

```
def ResponseCoding(t,ts,c):
  U_cat = t[c].unique()
  acc_count=[]
  rej_count=[]
  P_acc_count=[]
  P_rej_count=[]
  tot_count=[]

  for i in U_cat:
    acc_count.append((len(t.loc[(t[c] == i) & (t['project_is_approved'] == 1)])))

  for i in U_cat:
    rej_count.append((len(t.loc[(t[c] == i) & (t['project_is_approved'] == 0)])))

  for i in range(len(acc_count)):
    try:
      P_acc_count.append(acc_count[i]/(acc_count[i]+rej_count[i]))
    except ZeroDivisionError:
      P_acc_count.append(0)

  for i in range(len(rej_count)):
    try:
      P_rej_count.append(rej_count[i]/(acc_count[i]+rej_count[i]))
    except ZeroDivisionError:
      P_rej_count.append(0)
  P_acc_count_dict = dict(zip(U_cat, P_acc_count))
  P_rej_count_dict = dict(zip(U_cat, P_rej_count))
  ac1=c+'_acc'
  ac2=c+'_rej'
  df = pd.DataFrame()
  df[ac1] = t[c].map(P_acc_count_dict)
  df[ac2] = t[c].map(P_rej_count_dict)
  acc = df[ac1].values.tolist()
  rej = df[ac2].values.tolist()
  f_tr = pd.DataFrame(list(zip(acc, rej)))

  #-----------------------------------------------------------
  df1 = pd.DataFrame()
  df1[ac1] = ts[c].map(P_acc_count_dict)
  df1[ac2] = ts[c].map(P_rej_count_dict)
  acc_ts = df1[ac1].values.tolist()
  rej_ts = df1[ac2].values.tolist()
  f_ts = pd.DataFrame(list(zip(acc_ts, rej_ts)))

  #pdb.set_trace()
  f_ts.fillna(0.5, inplace = True)
  return f_tr,f_ts
```

project_subject_categories,project_subject_subcategories, School State, Prefix, project_grade_category

In [0]:

```
X_train_clean_cat, X_Test_clean_cat=ResponseCoding(X_Train,X_Test,'clean_categories')
X_train_clean_subcat, X_Test_clean_subcat=ResponseCoding(X_Train,X_Test,'clean_subcategories')
X_train_state, X_Test_state=ResponseCoding(X_Train,X_Test,'school_state')
X_train_teacher, X_Test_teacher=ResponseCoding(X_Train,X_Test,'teacher_prefix')
```

```
X_train_grade, X_Test_grade=ResponseCoding(X_Train,X_Test,'project_grade_category')

X_train_clean_cat_csr = scipy.sparse.csr_matrix(X_train_clean_cat.values)
X_train_clean_subcat_csr = scipy.sparse.csr_matrix(X_train_clean_subcat.values)
X_train_grade_csr = scipy.sparse.csr_matrix(X_train_grade.values)
X_train_state_csr = scipy.sparse.csr_matrix(X_train_state.values)
X_train_teacher_csr = scipy.sparse.csr_matrix(X_train_teacher.values)

X_Test_clean_cat_csr = scipy.sparse.csr_matrix(X_Test_clean_cat.values)
X_Test_clean_subcat_csr = scipy.sparse.csr_matrix(X_Test_clean_subcat.values)
X_Test_grade_csr = scipy.sparse.csr_matrix(X_Test_grade.values)
X_Test_state_csr = scipy.sparse.csr_matrix(X_Test_state.values)
X_Test_teacher_csr = scipy.sparse.csr_matrix(X_Test_teacher.values)
```

1.5.2 Vectorizing Numerical features

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))

p=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("-"*115)
print('Shape of Train normalized price dataset matrix after one hot encoding is:
{0}'.format(price_norm_Train.shape))

print('Shape of Test normalized price dataset matrix after one hot encoding is:
{0}'.format(price_norm_Test.shape))
```

```
-----------------------------------------------------------------------------------------------------------
------------
Shape of Train normalized price dataset matrix after one hot encoding is: (13400, 1)
Shape of Test normalized price dataset matrix after one hot encoding is: (6600, 1)
```

In [22]:

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

quantity_norm.transform(X_Train['quantity'].values.reshape(1,-1))

quantity_norm.transform(X_Test['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))
print("-"*115)
print('Shape of Train normalized quantity dataset matrix after one hot encoding is: {0}'.format(qu
antity_norm_Train.shape))

print('Shape of Test normalized quantity dataset matrix after one hot encoding is: {0}'.format(qua
ntity_norm_Test.shape))
```

```
-----------------------------------------------------------------------------------------------------------
------------
Shape of Train normalized quantity dataset matrix after one hot encoding is: (13400, 1)
Shape of Test normalized quantity dataset matrix after one hot encoding is: (6600, 1)
```

In [23]:

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.transform(X_Train['teacher_number_of_previously_posted_projects'].values.re
shape(1,-1))

teacher_prev_post_norm.transform(X_Test['teacher_number_of_previously_posted_projects'].values.res
hape(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))
print("-"*115)
print('Shape of Train normalized previously posted project dataset matrix after one hot encoding i
s: {0}'.format(teacher_prev_post_norm_Train.shape))

print('Shape of Test normalized previously posted project dataset matrix after one hot encoding is
: {0}'.format(teacher_prev_post_norm_Test.shape))
```

```
-----------------------------------------------------------------------------------------------------
------------
Shape of Train normalized previously posted project dataset matrix after one hot encoding is: (134
00, 1)
Shape of Test normalized previously posted project dataset matrix after one hot encoding is: (6600
, 1)
```

In [24]:

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

title_norm.transform(X_Test['word_count_title_Test'].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))
print("-"*115)
print('Shape of Train normalized title dataset matrix after one hot encoding is:
{0}'.format(word_count_title_Train.shape))

print('Shape of Test normalized title dataset matrix after one hot encoding is:
{0}'.format(word_count_title_Test.shape))
```

```
-----------------------------------------------------------------------------------------------------
------------
Shape of Train normalized title dataset matrix after one hot encoding is: (13400, 1)
Shape of Test normalized title dataset matrix after one hot encoding is: (6600, 1)
```

In [25]:

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

essay_norm.transform(X_Test['word_count_essay_Test'].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))
print("-"*115)
print('Shape of Train normalized title dataset matrix after one hot encoding is:
{0}'.format(word_count_essay_Train.shape))

print('Shape of Test normalized title dataset matrix after one hot encoding is:
{0}'.format(word_count_essay_Test.shape))
```

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

```
------------
Shape of Train normalized title dataset matrix after one hot encoding is: (13400, 1)
Shape of Test normalized title dataset matrix after one hot encoding is: (6600, 1)
```

1.5.3 Vectorizing Text data

1.5.3.1 Bag of words

In [26]:

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

text_bow_Test = vectorizer_essays_bow.transform(preprocessed_essays_Test)
print("-"*115)
print("Applying Bag Of Words for Text Data")
print("-"*115)
print('Shape of Train dataset matrix after one hot encoding is: {0}'.format(text_bow_Train.shape))

print('Shape of Test dataset matrix after one hot encoding is: {0}'.format(text_bow_Test.shape))
```

```
---------------------------------------------------------------------------------------------------
------------
Applying Bag Of Words for Text Data
---------------------------------------------------------------------------------------------------
------------
Shape of Train dataset matrix after one hot encoding is: (13400, 6972)
Shape of Test dataset matrix after one hot encoding is: (6600, 6972)
```

Bag of Words for Project Title

In [27]:

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

title_bow_Test = vectorizer_titles_bow.transform(preprocessed_titles_Test)
print("-"*115)
print("Applying Bag Of Words for Project Title Data")
print("-"*115)
print('Shape of Train dataset matrix after one hot encoding is: {0}'.format(title_bow_Train.shape)
)

print('Shape of Test dataset matrix after one hot encoding is: {0}'.format(title_bow_Test.shape))
```

```
---------------------------------------------------------------------------------------------------
------------
Applying Bag Of Words for Project Title Data
---------------------------------------------------------------------------------------------------
------------
Shape of Train dataset matrix after one hot encoding is: (13400, 840)
Shape of Test dataset matrix after one hot encoding is: (6600, 840)
```

1.5.2.2 TFIDF vectorizer

In [28]:

```
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("-"*115)
print("Applying TFIDF for Text Data")
print("-"*115)
```

```
print('Shape of Train dataset matrix after one hot encoding is: {0}'.format(text_tfidf_Train.shape
))

print('Shape of Test dataset matrix after one hot encoding is: {0}'.format(text_tfidf_Test.shape))
```

```
------------------------------------------------------------------------------------------------------
------------
Applying TFIDF for Text Data
------------------------------------------------------------------------------------------------------
------------
Shape of Train dataset matrix after one hot encoding is: (13400, 6972)
Shape of Test dataset matrix after one hot encoding is: (6600, 6972)
```

TFIDF vectorizer for Project Title

In [29]:

```
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("-"*115)
print("Applying TFIDF for Project Title")
print("-"*115)
print('Shape of Train dataset matrix after one hot encoding is: {0}'.format(title_tfidf_Train.shap
e))

print('Shape of Test dataset matrix after one hot encoding is: {0}'.format(title_tfidf_Test.shape)
)
```

```
------------------------------------------------------------------------------------------------------
------------
Applying TFIDF for Project Title
------------------------------------------------------------------------------------------------------
------------
Shape of Train dataset matrix after one hot encoding is: (13400, 840)
Shape of Test dataset matrix after one hot encoding is: (6600, 840)
```

1.5.2.3 Using Pretrained Models: Avg W2V

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

In [31]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_Train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_Train): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_Train.append(vector)

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

avg_w2v_vectors_Test = []; # the avg-w2v for each sentence/review is stored in this list
```

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

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

```
100%|██████████| 13400/13400 [00:03<00:00, 3923.27it/s]
100%|██████████| 6600/6600 [00:01<00:00, 3949.75it/s]
```

```
6600
300
```

AVG W2V on project_title

In [32]:

```
# Similarly you can vectorize for title also
# compute average word2vec for each title.
avg_w2v_vectors_title_Train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles_Train): # for each review/sentence
    vector_title = np.zeros(300) # as word vectors are of zero length
    cnt_title_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector_title += model[word]
            cnt_title_words += 1
    if cnt_title_words != 0:
        vector_title /= cnt_title_words
    avg_w2v_vectors_title_Train.append(vector_title)



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

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

```
100%|██████████| 13400/13400 [00:00<00:00, 56539.63it/s]
100%|██████████| 6600/6600 [00:00<00:00, 69045.31it/s]
```

```
6600
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [0]:

```
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())
```

In [34]:

```
# average Word2Vec
# compute average word2vec for each review.
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%|██████████| 13400/13400 [00:22<00:00, 604.13it/s]
100%|██████████| 6600/6600 [00:10<00:00, 614.37it/s]
```

```
6600
300
```

Using Pretrained Models: TFIDF weighted W2V on project_title

In [35]:

```
# 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%|██████████| 13400/13400 [00:00<00:00, 33498.65it/s]
100%|██████████| 6600/6600 [00:00<00:00, 35879.37it/s]
```

```
6600
300
```

Calculating the sentiment score's of each of the essay

In [36]:

```
sid = SentimentIntensityAnalyzer()

essays = X_Train['essay']
essays_sentiment_TR_P = []
essays_sentiment_TR_N = []
essays_sentiment_TR_NE = []
essays_sentiment_TR_C = []
for essay in tqdm(essays):
    res = sid.polarity_scores(essay)
    essays_sentiment_TR_P.append(res['pos'])
    essays_sentiment_TR_N.append(res['neg'])
    essays_sentiment_TR_NE.append(res['neu'])
    essays_sentiment_TR_C.append(res['compound'])
X_Train['sentiment_essay_TR_P'] = essays_sentiment_TR_P
X_Train['sentiment_essay_TR_N'] = essays_sentiment_TR_N
X_Train['sentiment_essay_TR_NE'] = essays_sentiment_TR_NE
X_Train['sentiment_essay_TR_C'] = essays_sentiment_TR_C


essays = X_Test['essay']
essays_sentiment_TS_P = []
essays_sentiment_TS_N = []
essays_sentiment_TS_NE = []
essays_sentiment_TS_C = []
for essay in tqdm(essays):
    res = sid.polarity_scores(essay)
    essays_sentiment_TS_P.append(res['pos'])
    essays_sentiment_TS_N.append(res['neg'])
    essays_sentiment_TS_NE.append(res['neu'])
```

```
    essays_sentiment_TS_C.append(res['compound'])
X_Test['sentiment_essay_TS_P'] = essays_sentiment_TS_P
X_Test['sentiment_essay_TS_N'] = essays_sentiment_TS_N
X_Test['sentiment_essay_TS_NE'] = essays_sentiment_TS_NE
X_Test['sentiment_essay_TS_C'] = essays_sentiment_TS_C

sentiment_norm_P = Normalizer(norm='l2', copy=False)
sentiment_norm_N = Normalizer(norm='l2', copy=False)
sentiment_norm_NE = Normalizer(norm='l2', copy=False)
sentiment_norm_C = Normalizer(norm='l2', copy=False)


sentiment_norm_P.fit(X_Train['sentiment_essay_TR_P'].values.reshape(1,-1))
sentiment_norm_N.fit(X_Train['sentiment_essay_TR_N'].values.reshape(1,-1))
sentiment_norm_NE.fit(X_Train['sentiment_essay_TR_NE'].values.reshape(1,-1))
sentiment_norm_C.fit(X_Train['sentiment_essay_TR_C'].values.reshape(1,-1))

sentiment_Train_P = sentiment_norm_P.transform(X_Train['sentiment_essay_TR_P'].values.reshape(1,-1)
)

sentiment_Test_P = sentiment_norm_P.transform(X_Test['sentiment_essay_TS_P'].values.reshape(1,-1))
sentiment_Train_N = sentiment_norm_N.transform(X_Train['sentiment_essay_TR_N'].values.reshape(1,-1)
)

sentiment_Test_N = sentiment_norm_N.transform(X_Test['sentiment_essay_TS_N'].values.reshape(1,-1))
sentiment_Train_NE = sentiment_norm_NE.transform(X_Train['sentiment_essay_TR_NE'].values.reshape(1
,-1))

sentiment_Test_NE = sentiment_norm_NE.transform(X_Test['sentiment_essay_TS_NE'].values.reshape(1,-1
))
sentiment_Train_C = sentiment_norm_C.transform(X_Train['sentiment_essay_TR_C'].values.reshape(1,-1)
)

sentiment_Test_C = sentiment_norm_C.transform(X_Test['sentiment_essay_TS_C'].values.reshape(1,-1))

sentiment_Train_P = (X_Train['sentiment_essay_TR_P'].values.reshape(-1,1))

sentiment_Test_P = (X_Test['sentiment_essay_TS_P'].values.reshape(-1,1))
sentiment_Train_N = (X_Train['sentiment_essay_TR_N'].values.reshape(-1,1))

sentiment_Test_N = (X_Test['sentiment_essay_TS_N'].values.reshape(-1,1))
sentiment_Train_NE = (X_Train['sentiment_essay_TR_NE'].values.reshape(-1,1))

sentiment_Test_NE = (X_Test['sentiment_essay_TS_NE'].values.reshape(-1,1))
sentiment_Train_C = (X_Train['sentiment_essay_TR_C'].values.reshape(-1,1))

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


print("Shape of sentiment Train matrix after one hot encodig ",sentiment_Train_P.shape)

print("Shape of sentiment Test matrix after one hot encodig ",sentiment_Test_P.shape)
print("Shape of sentiment Train matrix after one hot encodig ",sentiment_Train_N.shape)

print("Shape of sentiment Test matrix after one hot encodig ",sentiment_Test_N.shape)
print("Shape of sentiment Train matrix after one hot encodig ",sentiment_Train_NE.shape)

print("Shape of sentiment Test matrix after one hot encodig ",sentiment_Test_NE.shape)
print("Shape of sentiment Train matrix after one hot encodig ",sentiment_Train_C.shape)

print("Shape of sentiment Test matrix after one hot encodig ",sentiment_Test_C.shape)
```

```
100%|██████████| 13400/13400 [00:33<00:00, 396.65it/s]
100%|██████████| 6600/6600 [00:16<00:00, 399.40it/s]
```

```
Shape of sentiment Train matrix after one hot encodig  (13400, 1)
Shape of sentiment Test matrix after one hot encodig  (6600, 1)
Shape of sentiment Train matrix after one hot encodig  (13400, 1)
Shape of sentiment Test matrix after one hot encodig  (6600, 1)
Shape of sentiment Train matrix after one hot encodig  (13400, 1)
Shape of sentiment Test matrix after one hot encodig  (6600, 1)
Shape of sentiment Train matrix after one hot encodig  (13400, 1)
Shape of sentiment Test matrix after one hot encodig  (6600, 1)
```

1.5.4 Merging all the above features

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

In [0]:

```
BOW_Train = hstack((X_train_clean_cat_csr, X_train_clean_subcat_csr, X_train_grade_csr,
X_train_state_csr,
X_train_teacher_csr,text_bow_Train,title_bow_Train,price_norm_Train,quantity_norm_Train,teacher_pre
v_post_norm_Train,word_count_title_Train,word_count_essay_Train,sentiment_Train_P,sentiment_Train_N
,sentiment_Train_NE,sentiment_Train_C))
BOW_Test = hstack((X_Test_clean_cat_csr,X_Test_clean_subcat_csr,X_Test_grade_csr,X_Test_state_csr,
X_Test_teacher_csr,text_bow_Test,title_bow_Test,price_norm_Test,quantity_norm_Test,teacher_prev_pos
t_norm_Test,word_count_title_Test,word_count_essay_Test,sentiment_Test_P,sentiment_Test_N,sentiment
_Test_NE,sentiment_Test_C))
print(BOW_Train.shape)
print(BOW_Test.shape)
```

```
(33500, 11968)
(16500, 11968)
```

In [0]:

```
TFIDF_Train = hstack((X_train_clean_cat_csr, X_train_clean_subcat_csr, X_train_grade_csr,
X_train_state_csr, X_train_teacher_csr,text_tfidf_Train,title_tfidf_Train, price_norm_Train,
quantity_norm_Train, teacher_prev_post_norm_Train, word_count_title_Train, word_count_essay_Train,
sentiment_Train_P,sentiment_Train_N,sentiment_Train_NE,sentiment_Train_C))
TFIDF_Test =
hstack((X_Test_clean_cat_csr,X_Test_clean_subcat_csr,X_Test_grade_csr,X_Test_state_csr,X_Test_teach
er_csr,text_tfidf_Test,title_tfidf_Test,
price_norm_Test,quantity_norm_Test,teacher_prev_post_norm_Test,word_count_title_Test,word_count_ess
ay_Test,sentiment_Test_P,sentiment_Test_N,sentiment_Test_NE,sentiment_Test_C))
print(TFIDF_Train.shape)
print(TFIDF_Test.shape)
```

```
(33500, 11968)
(16500, 11968)
```

In [37]:

```
AVG_W2V_Train = hstack((X_train_clean_cat_csr, X_train_clean_subcat_csr, X_train_grade_csr,
X_train_state_csr, X_train_teacher_csr,avg_w2v_vectors_Train,avg_w2v_vectors_title_Train,
price_norm_Train, quantity_norm_Train, teacher_prev_post_norm_Train, word_count_title_Train,
word_count_essay_Train,sentiment_Train_P,sentiment_Train_N,sentiment_Train_NE,sentiment_Train_C))
AVG_W2V_Test =
hstack((X_Test_clean_cat_csr,X_Test_clean_subcat_csr,X_Test_grade_csr,X_Test_state_csr,X_Test_teach
er_csr,avg_w2v_vectors_Test,avg_w2v_vectors_title_Test,
price_norm_Test,quantity_norm_Test,teacher_prev_post_norm_Test,word_count_title_Test,word_count_ess
ay_Test,sentiment_Test_P,sentiment_Test_N,sentiment_Test_NE,sentiment_Test_C))
print(AVG_W2V_Train.shape)
print(AVG_W2V_Test.shape)
```

```
(13400, 619)
(6600, 619)
```

In [38]:

```
TFIDF_W2V_Train = hstack((X_train_clean_cat_csr, X_train_clean_subcat_csr, X_train_grade_csr,
X_train_state_csr, X_train_teacher_csr,tfidf_w2v_vectors_Train,tfidf_w2v_vectors_title_Train,
price_norm_Train, quantity_norm_Train, teacher_prev_post_norm_Train, word_count_title_Train,
word_count_essay_Train,sentiment_Train_P,sentiment_Train_N,sentiment_Train_NE,sentiment_Train_C))
TFIDF_W2V_Test =
hstack((X_Test_clean_cat_csr,X_Test_clean_subcat_csr,X_Test_grade_csr,X_Test_state_csr,X_Test_teach
er_csr,tfidf_w2v_vectors_Test,tfidf_w2v_vectors_title_Test, price_norm_Test,quantity_norm_Test,tea
cher_prev_post_norm_Test,word_count_title_Test,word_count_essay_Test,sentiment_Test_P,sentiment_Tes
t_N,sentiment_Test_NE,sentiment_Test_C))
print(TFIDF_W2V_Train.shape)
print(TFIDF_W2V_Test.shape)
```

```
(13400, 619)
(6600, 619)
```

50k

In [0]:

```
ptt = open('/content/gdrive/My Drive/Colab Notebooks/BOW_Train', 'wb')
pickle.dump(BOW_Train, ptt)

ptt = open('/content/gdrive/My Drive/Colab Notebooks/BOW_Test', 'wb')
pickle.dump(BOW_Test, ptt)

ptt = open('/content/gdrive/My Drive/Colab Notebooks/TFIDF_Train', 'wb')
pickle.dump(TFIDF_Train, ptt)

ptt = open('/content/gdrive/My Drive/Colab Notebooks/TFIDF_Test', 'wb')
pickle.dump(TFIDF_Test, ptt)

ptt = open('/content/gdrive/My Drive/Colab Notebooks/Y_Train_50k', 'wb')
pickle.dump(Y_Train, ptt)

ptt = open('/content/gdrive/My Drive/Colab Notebooks/Y_Test_50k', 'wb')
pickle.dump(Y_Test, ptt)
```

In [0]:

```
ptt = open('/content/gdrive/My Drive/Colab Notebooks/BOW_Train', 'rb')
BOW_Train = pickle.load(ptt)
ptt.close()

ptt = open('/content/gdrive/My Drive/Colab Notebooks/BOW_Test', 'rb')
BOW_Test = pickle.load(ptt)
ptt.close()

ptt = open('/content/gdrive/My Drive/Colab Notebooks/TFIDF_Train', 'rb')
TFIDF_Train = pickle.load(ptt)
ptt.close()

ptt = open('/content/gdrive/My Drive/Colab Notebooks/TFIDF_Test', 'rb')
TFIDF_Test = pickle.load(ptt)
ptt.close()

ptt = open('/content/gdrive/My Drive/Colab Notebooks/Y_Train_50k', 'rb')
Y_Train = pickle.load(ptt)
ptt.close()

ptt = open('/content/gdrive/My Drive/Colab Notebooks/Y_Test_50k', 'rb')
Y_Test = pickle.load(ptt)
ptt.close()
```

20k

In [0]:

```
ptt = open('/content/gdrive/My Drive/Colab Notebooks/AVG_W2V_Train', 'wb')
pickle.dump(AVG_W2V_Train, ptt)

ptt = open('/content/gdrive/My Drive/Colab Notebooks/AVG_W2V_Test', 'wb')
pickle.dump(AVG_W2V_Test, ptt)

ptt = open('/content/gdrive/My Drive/Colab Notebooks/TFIDF_W2V_Train', 'wb')
pickle.dump(TFIDF_W2V_Train, ptt)

ptt = open('/content/gdrive/My Drive/Colab Notebooks/TFIDF_W2V_Test', 'wb')
pickle.dump(TFIDF_W2V_Test, ptt)

ptt = open('/content/gdrive/My Drive/Colab Notebooks/Y_Train_20k', 'wb')
pickle.dump(Y_Train, ptt)

ptt = open('/content/gdrive/My Drive/Colab Notebooks/Y_Test_20k', 'wb')
```

```
pickle.dump(Y_Test, ptt)
```

In [0]:

```
ptt = open('/content/gdrive/My Drive/Colab Notebooks/AVG_W2V_Train', 'rb')
AVG_W2V_Train = pickle.load(ptt)
ptt.close()

ptt = open('/content/gdrive/My Drive/Colab Notebooks/AVG_W2V_Test', 'rb')
AVG_W2V_Test = pickle.load(ptt)
ptt.close()

ptt = open('/content/gdrive/My Drive/Colab Notebooks/TFIDF_W2V_Train', 'rb')
TFIDF_W2V_Train = pickle.load(ptt)
ptt.close()

ptt = open('/content/gdrive/My Drive/Colab Notebooks/TFIDF_W2V_Test', 'rb')
TFIDF_W2V_Test = pickle.load(ptt)
ptt.close()

ptt = open('/content/gdrive/My Drive/Colab Notebooks/Y_Train_20k', 'rb')
Y_Train = pickle.load(ptt)
ptt.close()

ptt = open('/content/gdrive/My Drive/Colab Notebooks/Y_Test_20k', 'rb')
Y_Test = pickle.load(ptt)
ptt.close()
```

Assignment 9: RF and GBDT

Response Coding: Example

> The response tabel is built only on train dataset. For a category which is not there in train data and present in test data, we will encode them with default values Ex: in our test data if have State: D then we encode it as [0.5, 0.05]

1. **Apply both Random Forrest and GBDT on these feature sets**
   - Set 1: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(BOW) + preprocessed_eassay (BOW)
   - Set 2: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
   - Set 3: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
   - Set 4: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)
2. **The hyper paramter tuning (Consider any two hyper parameters preferably n_estimators, max_depth)**
   - Find the best hyper parameter which will give the maximum AUC value
   - find the best hyper paramter using k-fold cross validation/simple cross validation data
   - use gridsearch cv or randomsearch cv or you can write your own for loops to do this task
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 **n_estimators**, 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 can choose either of the plotting techniques: 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

4. **Conclusion**
   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

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

2. Random Forest and GBDT

Applying Random Forest

Apply Random Forest 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

In [41]:

```
n_estimators=[10, 50, 100, 150, 200, 300, 500, 1000]
max_depth=[2, 3, 4, 5, 6, 7, 8, 9, 10]
ES=[]
MD=[]
for i in max_depth:
  for j in n_estimators:
    ES.append(i)
    MD.append(j)
print(ES)
print(MD)
```

```
[2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 6,
6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7, 7, 8, 8, 8, 8, 8, 8, 8, 8, 9, 9, 9, 9, 9, 9, 9, 9, 10, 10
, 10, 10, 10, 10, 10, 10]
[10, 50, 100, 150, 200, 300, 500, 1000, 10, 50, 100, 150, 200, 300, 500, 1000, 10, 50, 100, 150, 2
00, 300, 500, 1000, 10, 50, 100, 150, 200, 300, 500, 1000, 10, 50, 100, 150, 200, 300, 500, 1000,
10, 50, 100, 150, 200, 300, 500, 1000, 10, 50, 100, 150, 200, 300, 500, 1000, 10, 50, 100, 150, 20
0, 300, 500, 1000, 10, 50, 100, 150, 200, 300, 500, 1000]
```

2.4.1 Applying Random Forests on BOW, SET 1

In [0]:

```
%%time
RF_clf = RandomForestClassifier(class_weight='balanced')
parameters = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth':[2, 3, 4, 5, 6, 7
, 8, 9, 10]}
RFGrid_clf = GridSearchCV(RF_clf, parameters, cv=3, scoring='roc_auc',n_jobs=-1,return_train_score=
True,verbose=1)
RFGrid_clf.fit(BOW_Train, Y_Train)
print(RFGrid_clf.best_estimator_)
a1=RFGrid_clf.best_params_['n_estimators']
a2=RFGrid_clf.best_params_['max_depth']
AUC_Train= RFGrid_clf.cv_results_['mean_train_score']
```

```
AUC_Cv = RFGrid_clf.cv_results_['mean_test_score']
```

```
Fitting 3 folds for each of 72 candidates, totalling 216 fits
```

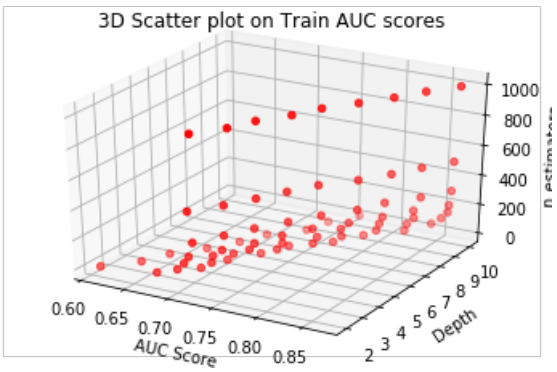
```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 40 concurrent workers.
[Parallel(n_jobs=-1)]: Done 121 tasks      | elapsed:   33.7s
[Parallel(n_jobs=-1)]: Done 216 out of 216 | elapsed:  1.5min finished
```

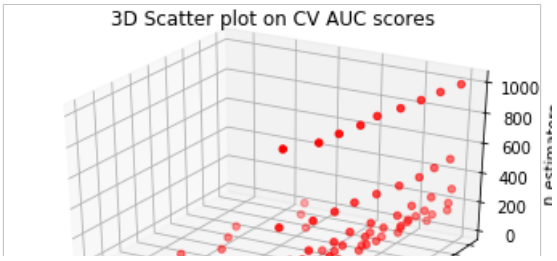
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight='balanced',
                       criterion='gini', max_depth=10, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=1000,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
CPU times: user 33.6 s, sys: 1.49 s, total: 35.1 s
Wall time: 2min 1s
```

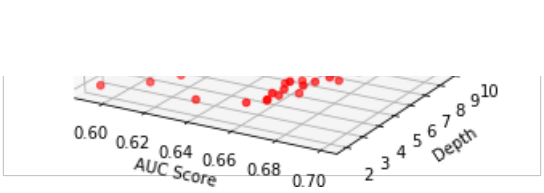
3D Plot

In [0]:

```
#-----------------------------------------------------------3D-Plot for Train Dataset-----------------
-------------------------------------------------------
figure = plt.figure()
ax = figure.add_subplot(111, projection='3d')
ax.scatter(AUC_Train, ES, MD, c='r', marker='o')
ax.set_xlabel('AUC Score')
ax.yaxis.set_label_text('Depth')
ax.zaxis.set_label_text('n_estimators')
plt.title('3D Scatter plot on Train AUC scores')
plt.show()
plt.close()

#-----------------------------------------------------------3D-Plot for CV Dataset--------------------
-----------------------------------------------------
figure = plt.figure()
ax = figure.add_subplot(111, projection='3d')
ax.scatter(AUC_Cv, ES, MD, c='r', marker='o')
ax.set_xlabel('AUC Score')
ax.yaxis.set_label_text('Depth')
ax.zaxis.set_label_text('n_estimators')
plt.title('3D Scatter plot on CV AUC scores')
plt.show()
plt.close()
```





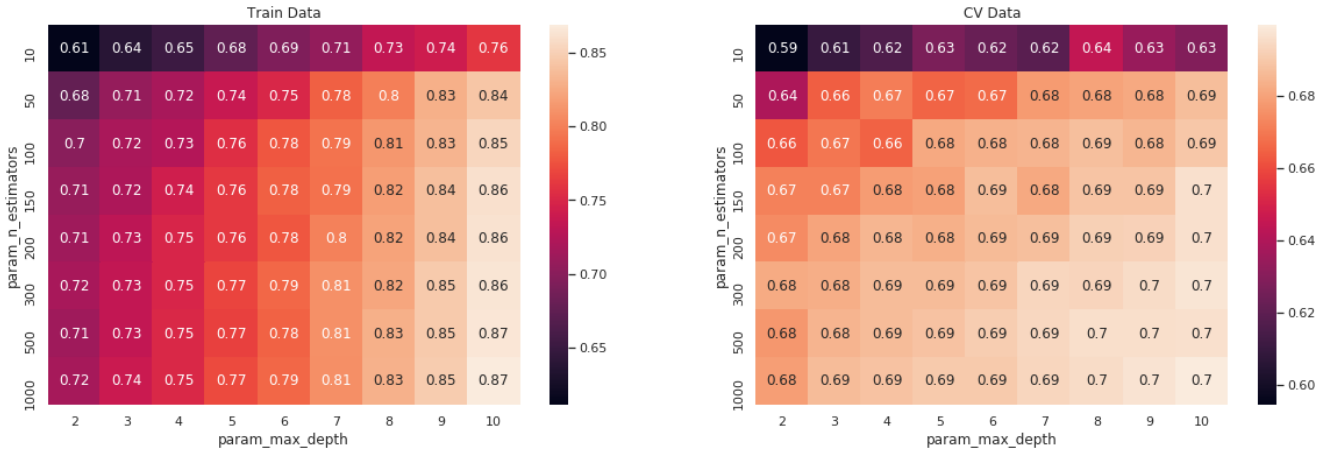


OBSERVATION:

Heat Map

In [0]:

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



In [0]:

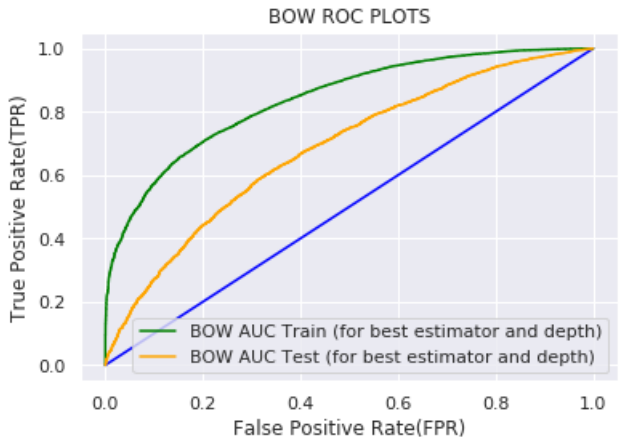
```
BOW_O_CLF = RandomForestClassifier(n_estimators=a1,max_depth=a2,n_jobs=-1,class_weight='balanced')
#pdb.set_trace()
BOW_O_CLF.fit(BOW_Train, Y_Train)
pred = BOW_O_CLF.predict(BOW_Test)

fpr_train, tpr_train, thresholds = roc_curve(Y_Train, BOW_O_CLF.predict_proba(BOW_Train)[:,1])
fpr_Test, tpr_Test, thresholds = roc_curve(Y_Test, BOW_O_CLF.predict_proba(BOW_Test)[:,1])
```

ROC PLOT

In [0]:

```
%%time
#https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-algorithm-using-python-and-sci
plt.plot([0,1],[0,1],'k-', color='blue')
plt.plot(fpr_train, tpr_train, label="BOW AUC Train (for best estimator and depth)", color='green'
)
plt.plot(fpr_Test, tpr_Test, label="BOW AUC Test (for best estimator and depth)", color='orange')
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("BOW ROC PLOTS")
plt.show()
print("-"*115)
print("AUC Train (for best estimator and depth) =", auc(fpr_train, tpr_train))
print("AUC Test (for best estimator and depth) =", auc(fpr_Test, tpr_Test))
```

```
BOW_AUC=round(auc(fpr_Test, tpr_Test)*100)
pred1 = BOW_O_CLF.predict(BOW_Train)
pred2 = BOW_O_CLF.predict(BOW_Test)
```




```
--------------------------------------------------------------------------------------------------------------
------------
AUC Train (for best estimator and depth) = 0.8391773228897468
AUC Test (for best estimator and depth) = 0.6836306443198281
CPU times: user 24.7 s, sys: 809 ms, total: 25.5 s
Wall time: 2.68 s
```

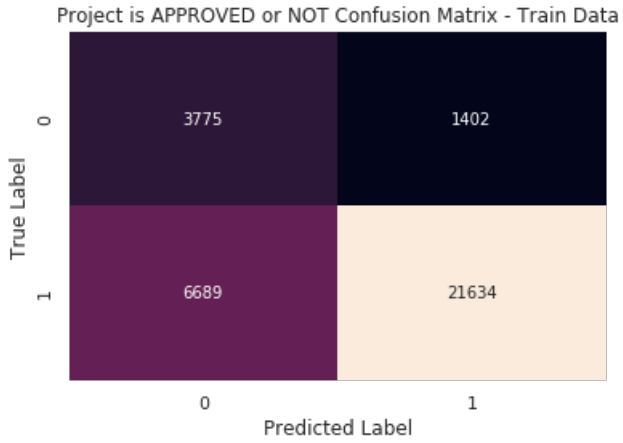
CONFUSION MATRIX

In [0]:

```
#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.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Train Data')
```

Out[0]:

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



Observations for Train data: True Positives - 21634, True Negatives - 3775, False Positives - 1402, False Negatives - 6689.

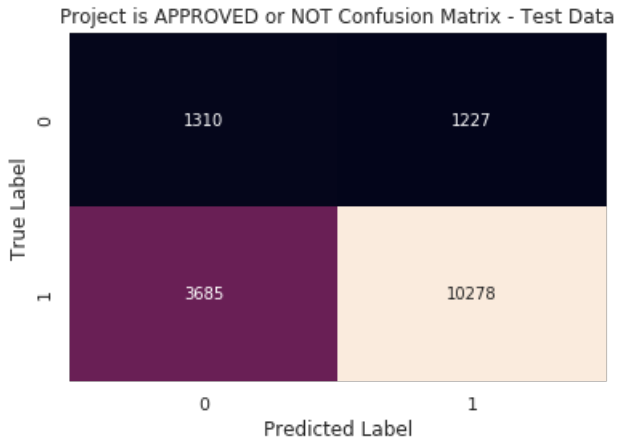
In [0]:

```
#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
Test = confusion_matrix(Y_Test, pred2)
sns.heatmap(Test,annot=True,cbar=False,fmt='g')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Test Data')
```

Out[0]:

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



Observations for Test data: True Positives - 10278, True Negatives - 1310, False Positives - 1227, False Negatives - 3685.

2.4.2 Applying Random Forests on TFIDF, SET 2

In [0]:

```
%%time
RF_clf = RandomForestClassifier(class_weight='balanced')
parameters = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth':[2, 3, 4, 5, 6, 7
, 8, 9, 10]}
RFGrid_clf = GridSearchCV(RF_clf, parameters, cv=3, scoring='roc_auc',n_jobs=-1,return_train_score=
True,verbose=1)
RFGrid_clf.fit(TFIDF_Train, Y_Train)
print(RFGrid_clf.best_estimator_)
a3=RFGrid_clf.best_params_['n_estimators']
a4=RFGrid_clf.best_params_['max_depth']
AUC_Train= RFGrid_clf.cv_results_['mean_train_score']
AUC_Cv = RFGrid_clf.cv_results_['mean_test_score']
```

```
Fitting 3 folds for each of 72 candidates, totalling 216 fits
```

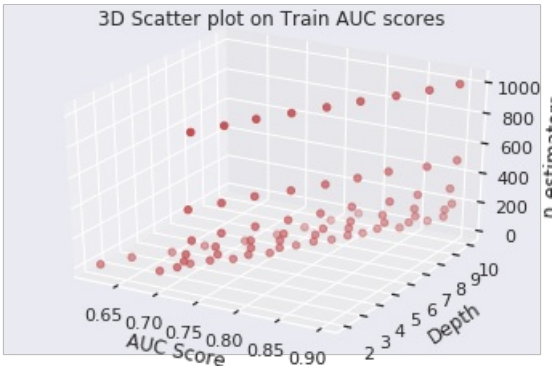
```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 40 concurrent workers.
[Parallel(n_jobs=-1)]: Done 120 tasks      | elapsed:   36.4s
[Parallel(n_jobs=-1)]: Done 216 out of 216 | elapsed:  1.8min finished
```

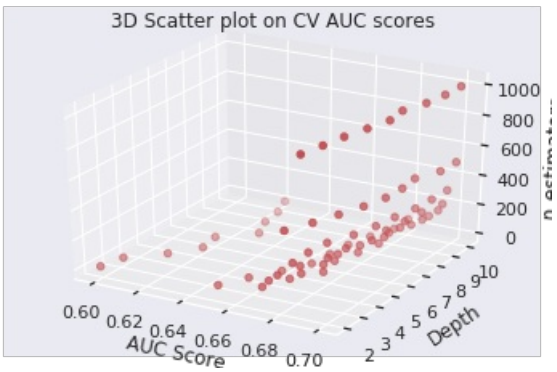
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight='balanced',
                       criterion='gini', max_depth=10, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=1000,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
CPU times: user 53.5 s, sys: 231 ms, total: 53.7 s
Wall time: 2min 41s
```

3D Plot

In [0]:

```
%%time
#https://stackoverflow.com/questions/53311685/difference-between-ax-set-xlabel-and-ax-xaxis-set-la
bel-in-matplotlib-3-0-1
#----------------------------------------------------------3D-Plot for Train Dataset-----------------
----------------------------------------------------------
figure = plt.figure()
ax = figure.add_subplot(111, projection='3d')
ax.scatter(AUC_Train, ES, MD, c='r', marker='o')
ax.set_xlabel('AUC Score')
ax.yaxis.set_label_text('Depth')
ax.zaxis.set_label_text('n_estimators')
plt.title('3D Scatter plot on Train AUC scores')
plt.show()
plt.close()

#----------------------------------------------------------3D-Plot for CV Dataset--------------------
----------------------------------------------------
figure = plt.figure()
ax = figure.add_subplot(111, projection='3d')

ax.scatter(AUC_Cv, ES, MD, c='r', marker='o')
ax.set_xlabel('AUC Score')
ax.yaxis.set_label_text('Depth')
ax.zaxis.set_label_text('n_estimators')
plt.title('3D Scatter plot on CV AUC scores')
plt.show()
plt.close()
```





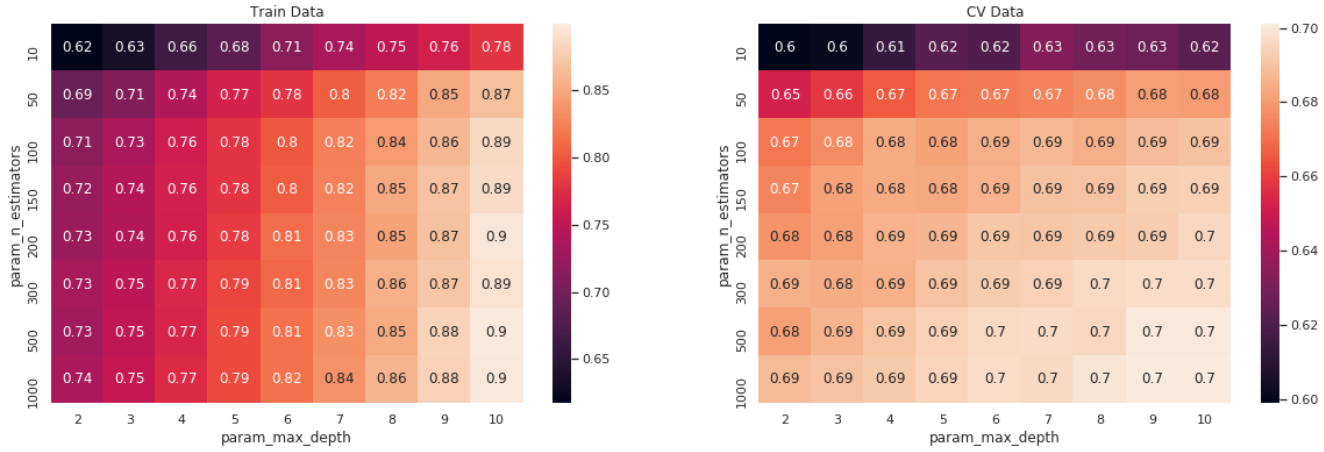

```
CPU times: user 422 ms, sys: 399 ms, total: 821 ms
Wall time: 344 ms
```

Heat Map

In [0]:

```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
sns.set()
RF_cvresult = pd.DataFrame(RFGrid_clf.cv_results_).groupby(['param_n_estimators',
'param_max_depth']).max().unstack()[['mean_test_score','mean_train_score']]
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(RF_cvresult.mean_train_score, annot = True, fmt='.2g', ax=ax[0])
```

```
sns.heatmap(RF_cvresult.mean_test_score, annot = True, fmt='.2g', ax=ax[1])
ax[0].set_title('Train Data')
ax[1].set_title('CV Data')
plt.show()
```



In [0]:

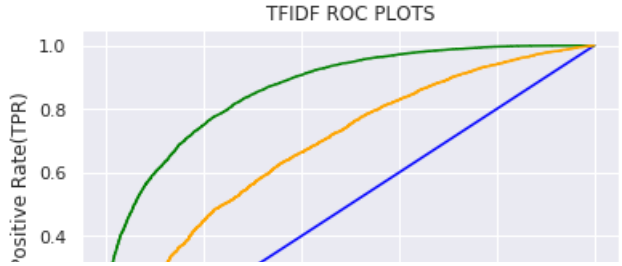
```
%%time
TFIDF_O_CLF =
RandomForestClassifier(n_estimators=a3,max_depth=a4,n_jobs=-1,class_weight='balanced')
TFIDF_O_CLF.fit(TFIDF_Train, Y_Train)
pred = TFIDF_O_CLF.predict(TFIDF_Test)

fpr_train, tpr_train, thresholds = roc_curve(Y_Train, TFIDF_O_CLF.predict_proba(TFIDF_Train)[:,1])
fpr_Test, tpr_Test, thresholds = roc_curve(Y_Test, TFIDF_O_CLF.predict_proba(TFIDF_Test)[:,1])
```

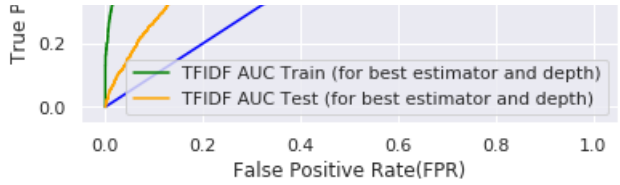
```
CPU times: user 1min 38s, sys: 3.94 s, total: 1min 42s
Wall time: 9.44 s
```

ROC PLOT

In [0]:

```
%%time
#https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-algorithm-using-python-and-sci
plt.plot([0,1],[0,1],'k-', color='blue')
plt.plot(fpr_train, tpr_train, label="TFIDF AUC Train (for best estimator and depth)",
color='green')
plt.plot(fpr_Test, tpr_Test, label="TFIDF AUC Test (for best estimator and depth)", color='orange'
)
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("TFIDF ROC PLOTS")
plt.show()
print("-"*115)
print("AUC Train (for best estimator and depth) =", auc(fpr_train, tpr_train))
print("AUC Test (for best estimator and depth) =", auc(fpr_Test, tpr_Test))

TFIDF_AUC=round(auc(fpr_Test, tpr_Test)*100)
pred1 = TFIDF_O_CLF.predict(TFIDF_Train)
pred2 = TFIDF_O_CLF.predict(TFIDF_Test)
```





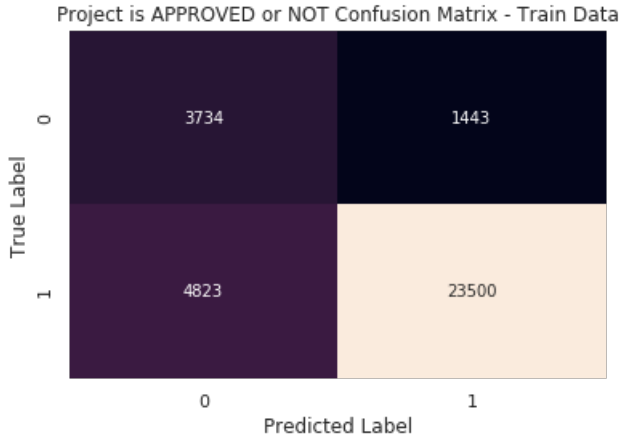

```
--------------------------------------------------------------------------------------------------------
------------
AUC Train (for best estimator and depth) = 0.8675966639452932
AUC Test (for best estimator and depth) = 0.6873725145155996
CPU times: user 25.5 s, sys: 753 ms, total: 26.2 s
Wall time: 2.71 s
```

In [0]:

```
#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.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Train Data')
```

Out[0]:

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



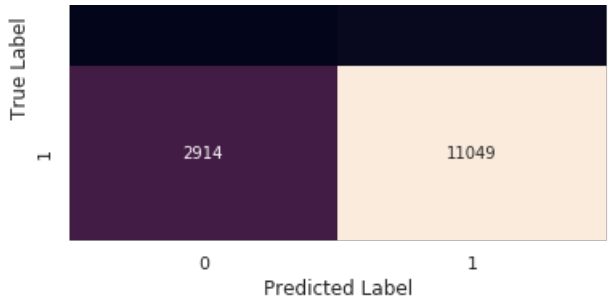
Observations for Train data: True Positives - 23500, True Negatives - 3734, False Positives - 1443, False Negatives - 4823.

In [0]:

```
#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
Test = confusion_matrix(Y_Test, pred2)
sns.heatmap(Test,annot=True,cbar=False,fmt='g')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Test Data')
```

Out[0]:

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



Observations for Test data: True Positives - 11049, True Negatives - 1158, False Positives - 1379, False Negatives - 2914.

2.4.3 Applying Random Forests on AVG_W2V, SET 3

In [0]:

```
%%time
RF_clf = RandomForestClassifier(class_weight='balanced')
parameters = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth':[2, 3, 4, 5, 6, 7
, 8, 9, 10]}
RFGrid_clf = GridSearchCV(RF_clf, parameters, cv=3, scoring='roc_auc',n_jobs=-1,return_train_score=
True,verbose=1)
RFGrid_clf.fit(AVG_W2V_Train, Y_Train)
print(RFGrid_clf.best_estimator_)
a5=RFGrid_clf.best_params_['n_estimators']
a6=RFGrid_clf.best_params_['max_depth']
AUC_Train= RFGrid_clf.cv_results_['mean_train_score']
AUC_Cv = RFGrid_clf.cv_results_['mean_test_score']
```

```
Fitting 3 folds for each of 72 candidates, totalling 216 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 40 concurrent workers.
[Parallel(n_jobs=-1)]: Done 121 tasks      | elapsed:  2.5min
[Parallel(n_jobs=-1)]: Done 216 out of 216 | elapsed: 10.4min finished
```

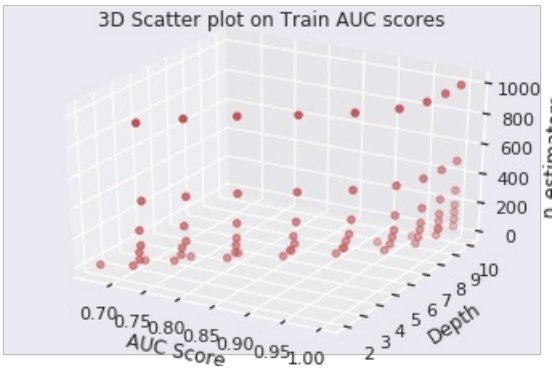
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight='balanced',
                       criterion='gini', max_depth=6, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=1000,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
CPU times: user 3min 54s, sys: 1.69 s, total: 3min 56s
Wall time: 14min 18s
```

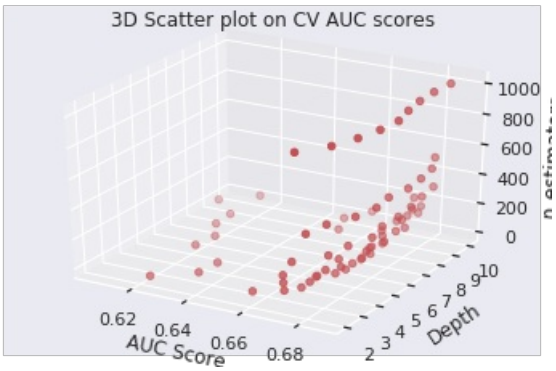
3D Plot

In [0]:

```
%%time
#https://stackoverflow.com/questions/53311685/difference-between-ax-set-xlabel-and-ax-xaxis-set-la
bel-in-matplotlib-3-0-1
#----------------------------------------------------------3D-Plot for Train Dataset-----------------
-----------------------------------------------------------
figure = plt.figure()
ax = figure.add_subplot(111, projection='3d')
ax.scatter(AUC_Train, ES, MD, c='r', marker='o')
ax.set_xlabel('AUC Score')
ax.yaxis.set_label_text('Depth')
ax.zaxis.set_label_text('n_estimators')
plt.title('3D Scatter plot on Train AUC scores')
plt.show()
plt.close()

#----------------------------------------------------------3D-Plot for CV Dataset--------------------
-----------------------------------------------------
```

```
figure = plt.figure()
ax = figure.add_subplot(111, projection='3d')

ax.scatter(AUC_Cv, ES, MD, c='r', marker='o')
ax.set_xlabel('AUC Score')
ax.yaxis.set_label_text('Depth')
ax.zaxis.set_label_text('n_estimators')
plt.title('3D Scatter plot on CV AUC scores')
plt.show()
plt.close()
```



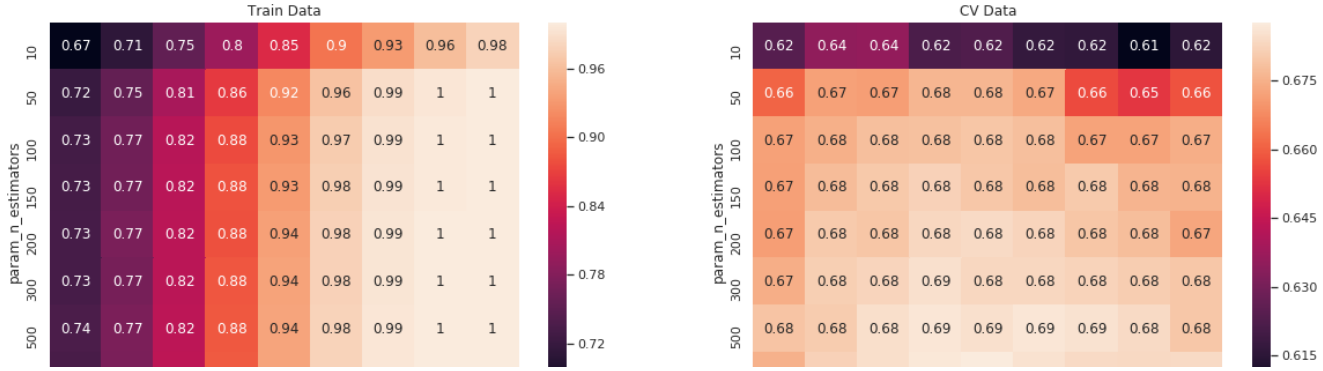


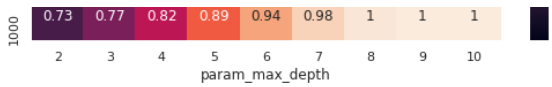

```
CPU times: user 977 ms, sys: 2.98 s, total: 3.96 s
Wall time: 398 ms
```

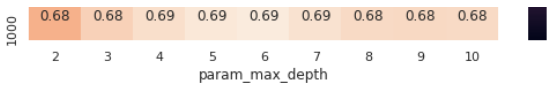
Heat Map

In [0]:

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







In [0]:

```
%%time
AVG_W2V_O_CLF =
RandomForestClassifier(n_estimators=a5,max_depth=a6,n_jobs=-1,class_weight='balanced')
AVG_W2V_O_CLF.fit(AVG_W2V_Train, Y_Train)
pred = AVG_W2V_O_CLF.predict(AVG_W2V_Test)

fpr_train, tpr_train, thresholds = roc_curve(Y_Train, AVG_W2V_O_CLF.predict_proba(AVG_W2V_Train)
[:,1])
fpr_Test, tpr_Test, thresholds = roc_curve(Y_Test, AVG_W2V_O_CLF.predict_proba(AVG_W2V_Test)[:,1])
```

```
CPU times: user 5min 57s, sys: 2.72 s, total: 5min 59s
Wall time: 12.6 s
```

ROC PLOT

In [0]:

```
%%time
#https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-al
gorithm-using-python-and-sci
plt.plot([0,1],[0,1],'k-', color='blue')
plt.plot(fpr_train, tpr_train, label="AVG_W2V AUC Train (for best estimator and depth)", color='gr
een')
plt.plot(fpr_Test, tpr_Test, label="AVG_W2V AUC Test (for best estimator and depth)",
color='orange')
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("AVG_W2V ROC PLOTS")
plt.show()
print("-"*115)
print("AUC Train (for best estimator and depth) =", auc(fpr_train, tpr_train))
print("AUC Test (for best estimator and depth) =", auc(fpr_Test, tpr_Test))

AVG_W2V_AUC=round(auc(fpr_Test, tpr_Test)*100)
pred1 = AVG_W2V_O_CLF.predict(AVG_W2V_Train)
pred2 = AVG_W2V_O_CLF.predict(AVG_W2V_Test)
```

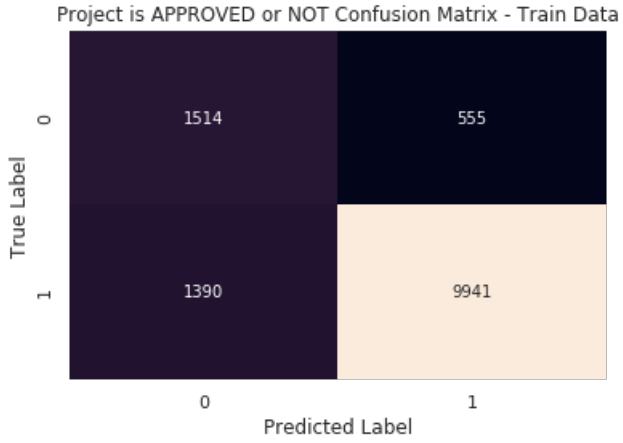

```
---------------------------------------------------------------------------------------------------
------------
AUC Train (for best estimator and depth) = 0.8994400618431135
AUC Test (for best estimator and depth) = 0.6659272641440918
CPU times: user 20.9 s, sys: 609 ms, total: 21.5 s
Wall time: 2.62 s
```

In [0]:

```
#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.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Train Data')
```

Out[0]:

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



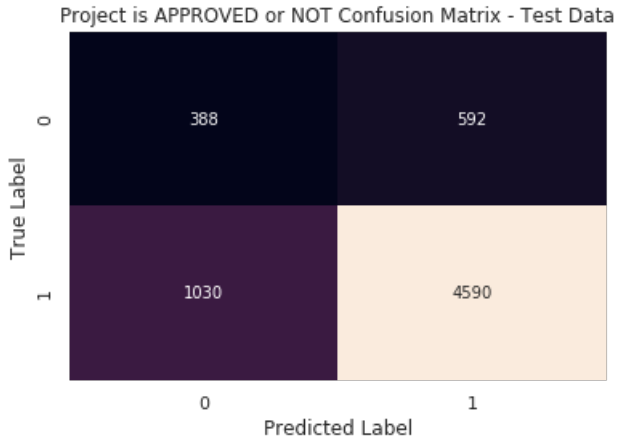
Observations for Train data: True Positives - 9941, True Negatives - 1514, False Positives - 555, False Negatives - 1390.

In [0]:

```
#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
Test = confusion_matrix(Y_Test, pred2)
sns.heatmap(Test,annot=True,cbar=False,fmt='g')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Test Data')
```

Out[0]:

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



Observations for Test data: True Positives - 4590, True Negatives - 388, False Positives - 592, False Negatives - 1030.

2.4.4 Applying Random Forests on TFIDF W2V, SET 4

In [0]:

```
%%time
RF_clf = RandomForestClassifier(class_weight='balanced')
parameters = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth':[2, 3, 4, 5, 6, 7
, 8, 9, 10]}
RFGrid_clf = GridSearchCV(RF_clf, parameters, cv=3, scoring='roc_auc',n_jobs=-1,return_train_score=
True,verbose=1)
RFGrid_clf.fit(TFIDF_W2V_Train, Y_Train)
print(RFGrid_clf.best_estimator_)
a7=RFGrid_clf.best_params_['n_estimators']
a8=RFGrid_clf.best_params_['max_depth']
AUC_Train= RFGrid_clf.cv_results_['mean_train_score']
AUC_Cv = RFGrid_clf.cv_results_['mean_test_score']
```

```
Fitting 3 folds for each of 72 candidates, totalling 216 fits
```

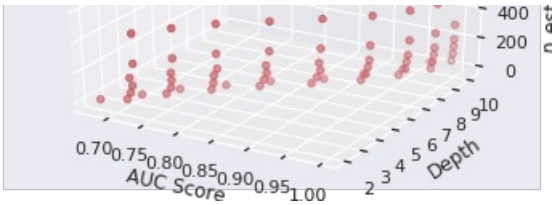
```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 40 concurrent workers.
[Parallel(n_jobs=-1)]: Done 121 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done 216 out of 216 | elapsed:  5.1min finished
```

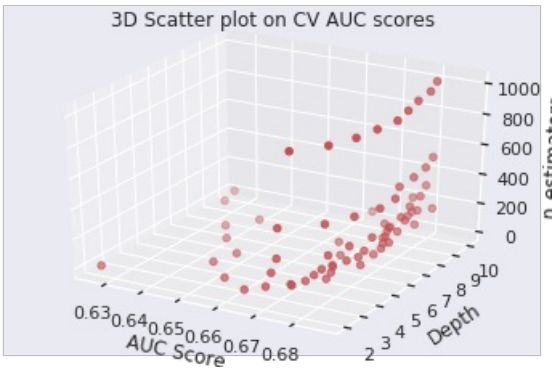
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight='balanced',
                       criterion='gini', max_depth=5, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=500,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
CPU times: user 49.9 s, sys: 1.35 s, total: 51.3 s
Wall time: 5min 56s
```

3D Plot

In [0]:

```
%%time
#https://stackoverflow.com/questions/53311685/difference-between-ax-set-xlabel-and-ax-xaxis-set-la
bel-in-matplotlib-3-0-1
#----------------------------------------------------------3D-Plot for Train Dataset-----------------
----------------------------------------------------------
figure = plt.figure()
ax = figure.add_subplot(111, projection='3d')
ax.scatter(AUC_Train, ES, MD, c='r', marker='o')
ax.set_xlabel('AUC Score')
ax.yaxis.set_label_text('Depth')
ax.zaxis.set_label_text('n_estimators')
plt.title('3D Scatter plot on Train AUC scores')
plt.show()
plt.close()

#----------------------------------------------------------3D-Plot for CV Dataset--------------------
-----------------------------------------------------
figure = plt.figure()
ax = figure.add_subplot(111, projection='3d')

ax.scatter(AUC_Cv, ES, MD, c='r', marker='o')
ax.set_xlabel('AUC Score')
ax.yaxis.set_label_text('Depth')
ax.zaxis.set_label_text('n_estimators')
plt.title('3D Scatter plot on CV AUC scores')
plt.show()
plt.close()
```



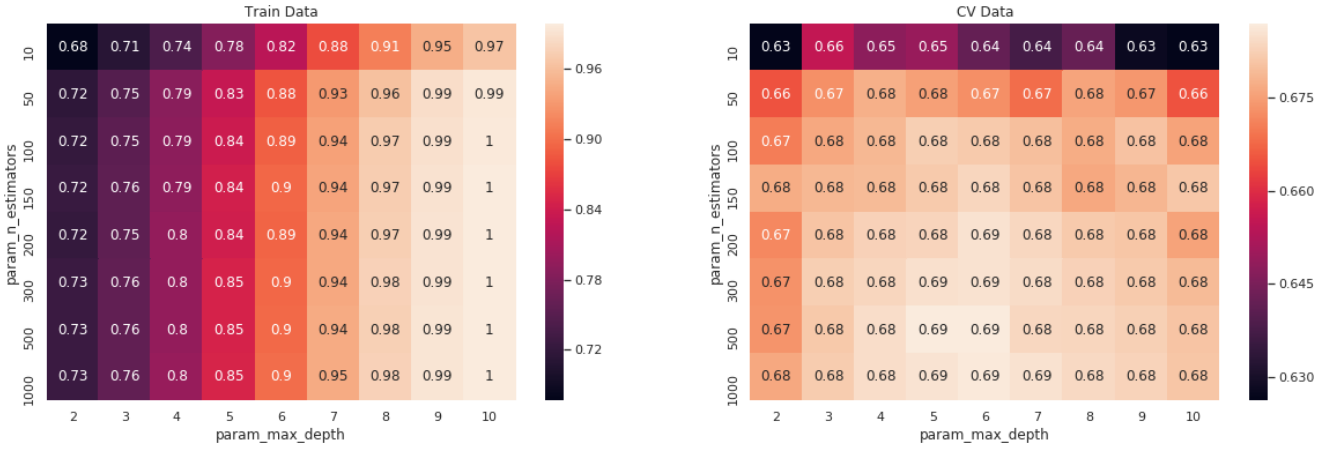



```
CPU times: user 897 ms, sys: 2.85 s, total: 3.75 s
Wall time: 370 ms
```

Heat Map

In [0]:

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



In [0]:

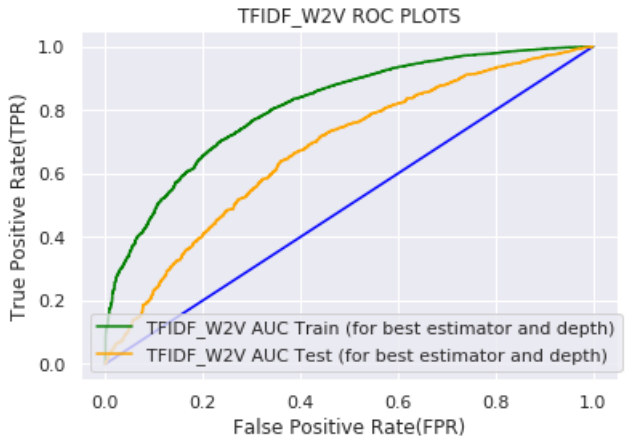
```
%%time
TFIDF_W2V_O_CLF =
RandomForestClassifier(n_estimators=a7,max_depth=a8,n_jobs=-1,class_weight='balanced')
TFIDF_W2V_O_CLF.fit(TFIDF_W2V_Train, Y_Train)
pred = TFIDF_W2V_O_CLF.predict(TFIDF_W2V_Test)

fpr_train, tpr_train, thresholds = roc_curve(Y_Train,
TFIDF_W2V_O_CLF.predict_proba(TFIDF_W2V_Train)[:,1])
fpr_Test, tpr_Test, thresholds = roc_curve(Y_Test, TFIDF_W2V_O_CLF.predict_proba(TFIDF_W2V_Test)[:,
1])
```

```
CPU times: user 1min 12s, sys: 1.42 s, total: 1min 14s
Wall time: 4.32 s
```

ROC PLOT

In [0]:

```
%%time
#https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-algorithm-using-python-and-sci
plt.plot([0,1],[0,1],'k-', color='blue')
plt.plot(fpr_train, tpr_train, label="TFIDF_W2V AUC Train (for best estimator and depth)", color='green')
plt.plot(fpr_Test, tpr_Test, label="TFIDF_W2V AUC Test (for best estimator and depth)", color='orange')
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("TFIDF_W2V ROC PLOTS")
plt.show()
print("-"*115)
print("AUC Train (for best estimator and depth) =", auc(fpr_train, tpr_train))
print("AUC Test (for best estimator and depth) =", auc(fpr_Test, tpr_Test))

TFIDF_W2V_AUC=round(auc(fpr_Test, tpr_Test)*100)
pred1 = TFIDF_W2V_O_CLF.predict(TFIDF_W2V_Train)
pred2 = TFIDF_W2V_O_CLF.predict(TFIDF_W2V_Test)
```



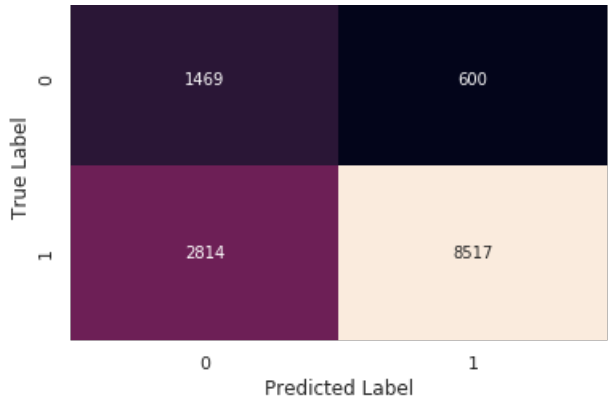

```
------------------------------------------------------------------------------------------------------------
AUC Train (for best estimator and depth) = 0.8109958441533404
AUC Test (for best estimator and depth) = 0.6737840438666569
CPU times: user 5.64 s, sys: 263 ms, total: 5.91 s
Wall time: 1.32 s
```

In [0]:

```
#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.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Train Data')
```

Out[0]:

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



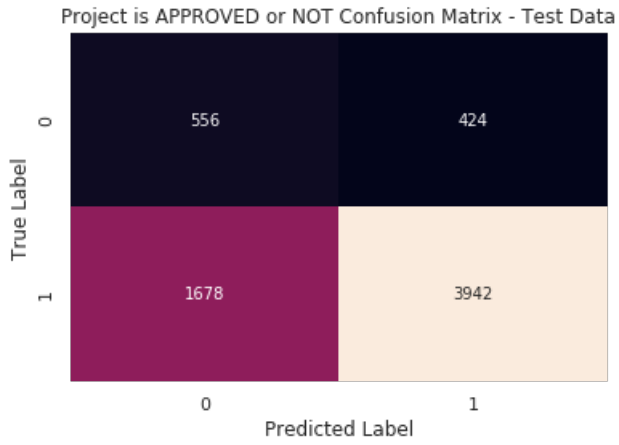
Observations for Train data: True Positives - 8517, True Negatives - 1469, False Positives - 600, False Negatives - 2814.

In [0]:

```
#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
Test = confusion_matrix(Y_Test, pred2)
sns.heatmap(Test,annot=True,cbar=False,fmt='g')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Test Data')
```

Out[0]:

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



Observations for Test data: True Positives - 3942, True Negatives - 556, False Positives - 424, False Negatives - 1678.

2.4 Applying GBDT

Apply GBDT 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

2.4.5 Applying GBDT on BOW, SET 1

In [0]:

```
%%time
GBDT_clf = GradientBoostingClassifier()
parameters = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth':[2, 3, 4, 5, 6, 7
, 8, 9, 10]}
GBDTGrid_clf = GridSearchCV(GBDT_clf, parameters, cv=3, scoring='roc_auc',n_jobs=-1,return_train_sc
ore=True,verbose=1)
```

```
GBDTGrid_clf.fit(BOW_Train, Y_Train)
print(GBDTGrid_clf.best_estimator_)
a9=GBDTGrid_clf.best_params_['n_estimators']
a10=GBDTGrid_clf.best_params_['max_depth']
AUC_Train= GBDTGrid_clf.cv_results_['mean_train_score']
AUC_Cv = GBDTGrid_clf.cv_results_['mean_test_score']
```

```
Fitting 3 folds for each of 72 candidates, totalling 216 fits
```

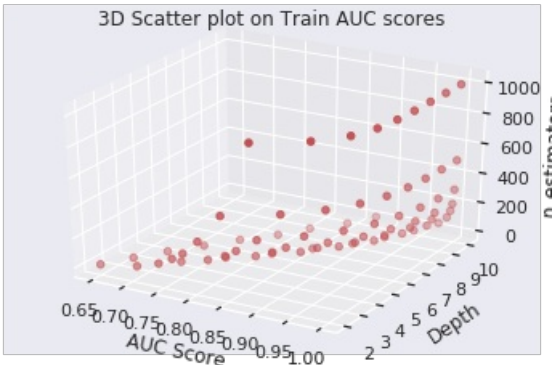
```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 40 concurrent workers.
[Parallel(n_jobs=-1)]: Done 120 tasks      | elapsed:  9.4min
[Parallel(n_jobs=-1)]: Done 216 out of 216 | elapsed: 34.9min finished
```

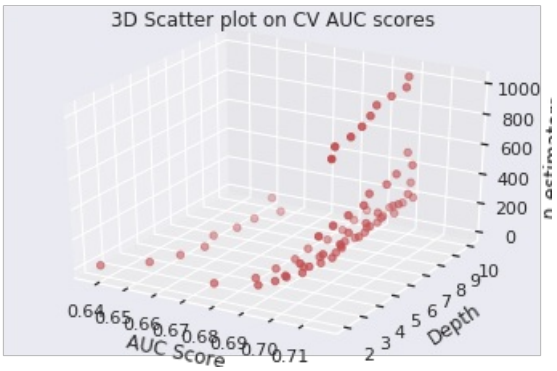
```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=2,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=1000,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
CPU times: user 4min 42s, sys: 397 ms, total: 4min 42s
Wall time: 39min 36s
```

3D Plot

In [0]:

```
%%time
#https://stackoverflow.com/questions/53311685/difference-between-ax-set-xlabel-and-ax-xaxis-set-label-in-matplotlib-3-0-1
#----------------------------------------------------------3D-Plot for Train Dataset-----------------
-----------------------------------------------------------
figure = plt.figure()
ax = figure.add_subplot(111, projection='3d')
ax.scatter(AUC_Train, ES, MD, c='r', marker='o')
ax.set_xlabel('AUC Score')
ax.yaxis.set_label_text('Depth')
ax.zaxis.set_label_text('n_estimators')
plt.title('3D Scatter plot on Train AUC scores')
plt.show()
plt.close()

#----------------------------------------------------------3D-Plot for CV Dataset--------------------
-------------------------------------------------
figure = plt.figure()
ax = figure.add_subplot(111, projection='3d')

ax.scatter(AUC_Cv, ES, MD, c='r', marker='o')
ax.set_xlabel('AUC Score')
ax.yaxis.set_label_text('Depth')
ax.zaxis.set_label_text('n_estimators')
plt.title('3D Scatter plot on CV AUC scores')
plt.show()
plt.close()
```



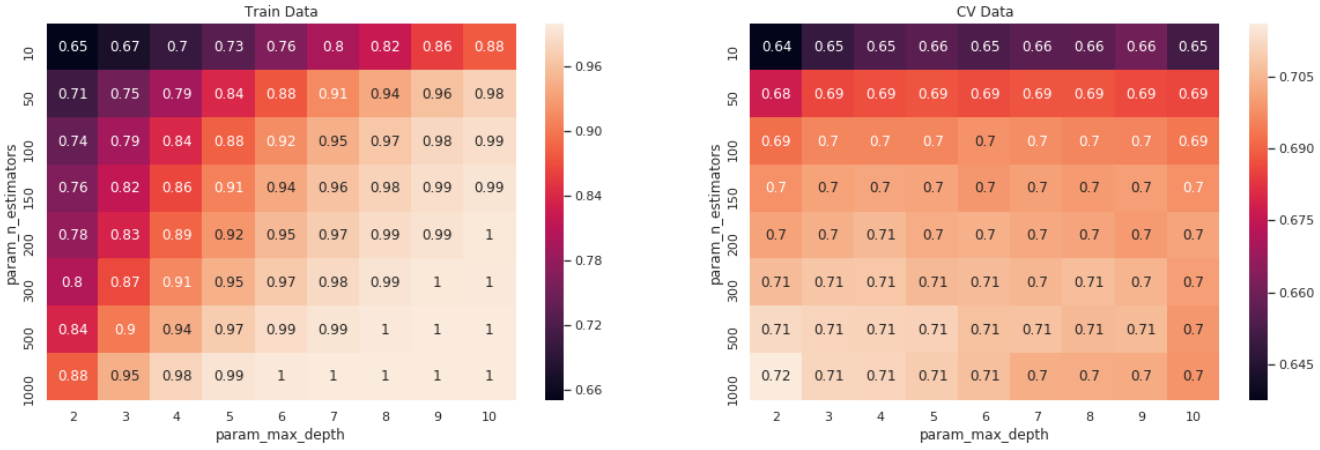



```
CPU times: user 492 ms, sys: 435 ms, total: 926 ms
Wall time: 444 ms
```

Heat Map

In [0]:

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



In [0]:

```
%%time
BOW_GBDT_O_CLF = GradientBoostingClassifier(n_estimators=a9,max_depth=a10)
BOW_GBDT_O_CLF.fit(BOW_Train, Y_Train)
pred = BOW_GBDT_O_CLF.predict(BOW_Test)

fpr_train, tpr_train, thresholds = roc_curve(Y_Train, BOW_GBDT_O_CLF.predict_proba(BOW_Train)
[:,1])
fpr_Test, tpr_Test, thresholds = roc_curve(Y_Test, BOW_GBDT_O_CLF.predict_proba(BOW_Test)[:,1])
```

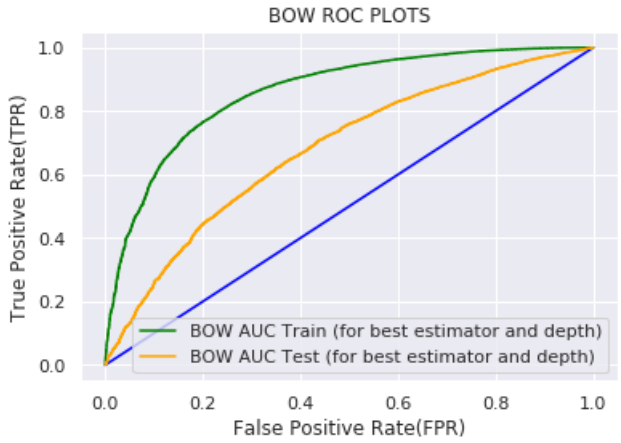
```
CPU times: user 4min 46s, sys: 444 ms, total: 4min 46s
Wall time: 4min 46s
```

ROC PLOT

In [0]:

In [0]:

```
%%time
#https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-algorithm-using-python-and-sci
plt.plot([0,1],[0,1],'k-', color='blue')
plt.plot(fpr_train, tpr_train, label="BOW AUC Train (for best estimator and depth)", color='green')
plt.plot(fpr_Test, tpr_Test, label="BOW AUC Test (for best estimator and depth)", color='orange')
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("BOW ROC PLOTS")
plt.show()
print("-"*115)
print("AUC Train (for best estimator and depth) =", auc(fpr_train, tpr_train))
print("AUC Test (for best estimator and depth) =", auc(fpr_Test, tpr_Test))

BOW_GBDT_AUC=round(auc(fpr_Test, tpr_Test)*100)
pred1 = BOW_GBDT_O_CLF.predict(BOW_Train)
pred2 = BOW_GBDT_O_CLF.predict(BOW_Test)
```



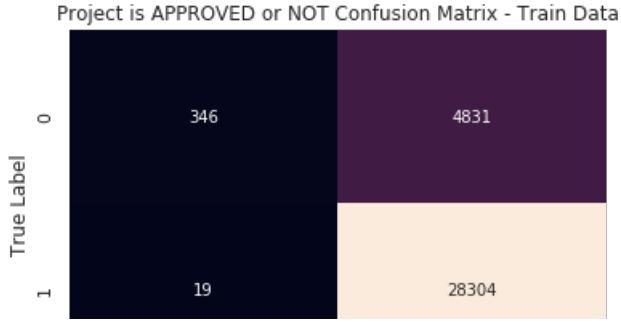

```
-------------------------------------------------------------------------------------------------------------
AUC Train (for best estimator and depth) = 0.8600601960724177
AUC Test (for best estimator and depth) = 0.6808347818045275
CPU times: user 1.17 s, sys: 7 ms, total: 1.18 s
Wall time: 1.17 s
```

In [0]:

```
#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.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Train Data')
```

Out[0]:

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



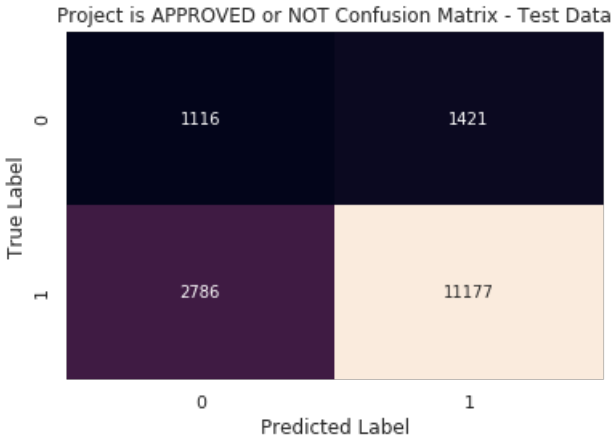
Observations for Train data: True Positives - 28304, True Negatives - 346, False Positives - 4831, False Negatives - 19.

In [0]:

```
#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
Test = confusion_matrix(Y_Test, pred2)
sns.heatmap(Test,annot=True,cbar=False,fmt='g')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Test Data')
```

Out[0]:

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



Observations for Test data: True Positives - 11177, True Negatives - 1116, False Positives - 1421, False Negatives - 2786.

2.4.6 Applying GBDT on TFIDF, SET 2

In [0]:

```
%%time
GBDT_clf = GradientBoostingClassifier()
parameters = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth':[2, 3, 4, 5, 6, 7
, 8, 9, 10]}
GBDTGrid_clf = GridSearchCV(GBDT_clf, parameters, cv=3, scoring='roc_auc',n_jobs=-1,return_train_sc
ore=True,verbose=1)
GBDTGrid_clf.fit(TFIDF_Train, Y_Train)
print(GBDTGrid_clf.best_estimator_)
a11=GBDTGrid_clf.best_params_['n_estimators']
a12=GBDTGrid_clf.best_params_['max_depth']
AUC_Train= GBDTGrid_clf.cv_results_['mean_train_score']
AUC_Cv = GBDTGrid_clf.cv_results_['mean_test_score']
```

```
Fitting 3 folds for each of 72 candidates, totalling 216 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 40 concurrent workers.
[Parallel(n_jobs=-1)]: Done 120 tasks      | elapsed: 26.0min
[Parallel(n_jobs=-1)]: Done 216 out of 216 | elapsed: 106.7min finished
```

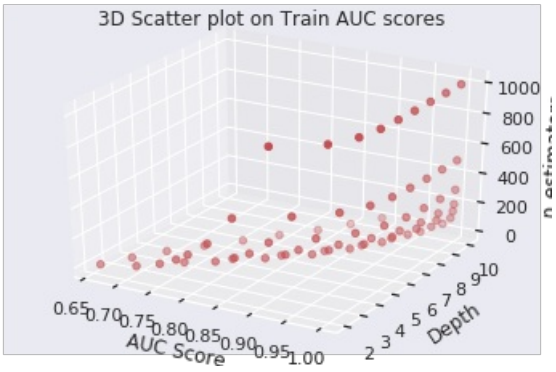
```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=2,
                           max_features=None, max_leaf_nodes=None,
```

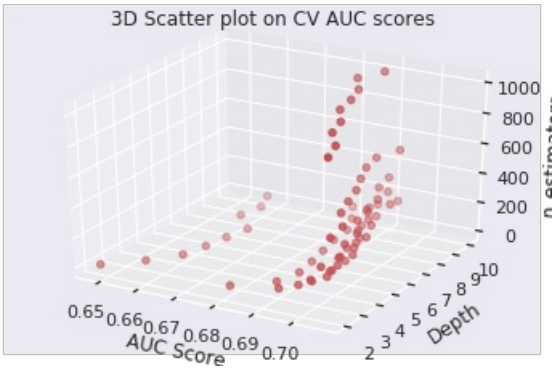
```
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=500,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
CPU times: user 8min 17s, sys: 1.43 s, total: 8min 18s
Wall time: 1h 54min 56s
```

3D Plot

In [0]:

```
%%time
#https://stackoverflow.com/questions/53311685/difference-between-ax-set-xlabel-and-ax-xaxis-set-label-in-matplotlib-3-0-1
#----------------------------------------------------------3D-Plot for Train Dataset-----------------------------------------------------------
figure = plt.figure()
ax = figure.add_subplot(111, projection='3d')
ax.scatter(AUC_Train, ES, MD, c='r', marker='o')
ax.set_xlabel('AUC Score')
ax.yaxis.set_label_text('Depth')
ax.zaxis.set_label_text('n_estimators')
plt.title('3D Scatter plot on Train AUC scores')
plt.show()
plt.close()

#----------------------------------------------------------3D-Plot for CV Dataset-----------------------------------------------------------
figure = plt.figure()
ax = figure.add_subplot(111, projection='3d')

ax.scatter(AUC_Cv, ES, MD, c='r', marker='o')
ax.set_xlabel('AUC Score')
ax.yaxis.set_label_text('Depth')
ax.zaxis.set_label_text('n_estimators')
plt.title('3D Scatter plot on CV AUC scores')
plt.show()
plt.close()
```





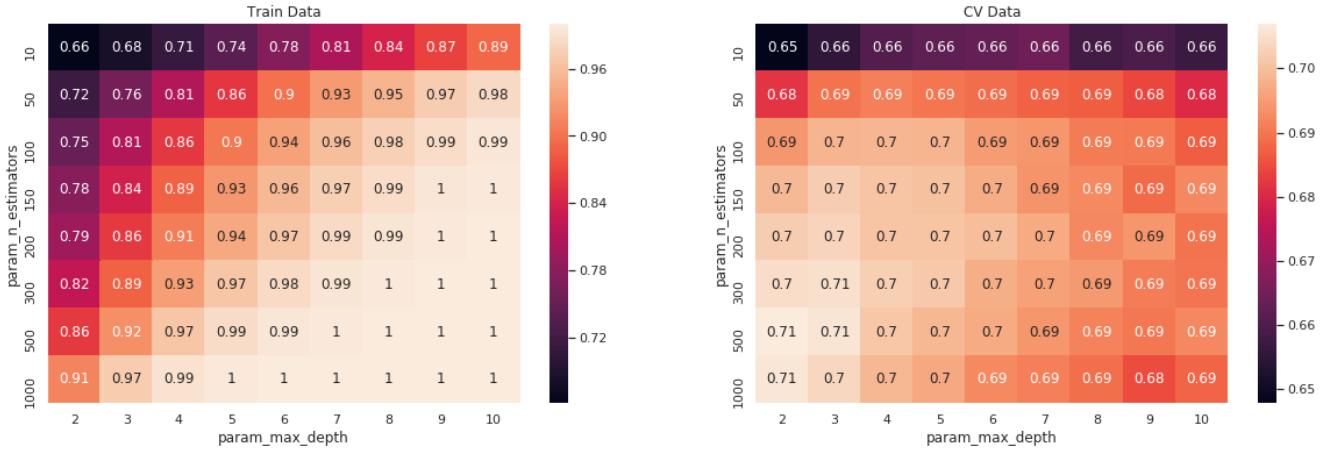

```
CPU times: user 1.56 s, sys: 2.8 s, total: 4.37 s
```

Wall time: 1 s

Heat Map

In [0]:

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



In [0]:

```
%%time
TFIDF_GBDT_O_CLF = GradientBoostingClassifier(n_estimators=a11,max_depth=a12)
TFIDF_GBDT_O_CLF.fit(TFIDF_Train, Y_Train)
pred = TFIDF_GBDT_O_CLF.predict(TFIDF_Test)

fpr_train, tpr_train, thresholds = roc_curve(Y_Train, TFIDF_GBDT_O_CLF.predict_proba(TFIDF_Train)
[:,1])
fpr_Test, tpr_Test, thresholds = roc_curve(Y_Test, TFIDF_GBDT_O_CLF.predict_proba(TFIDF_Test)[:,1])
```

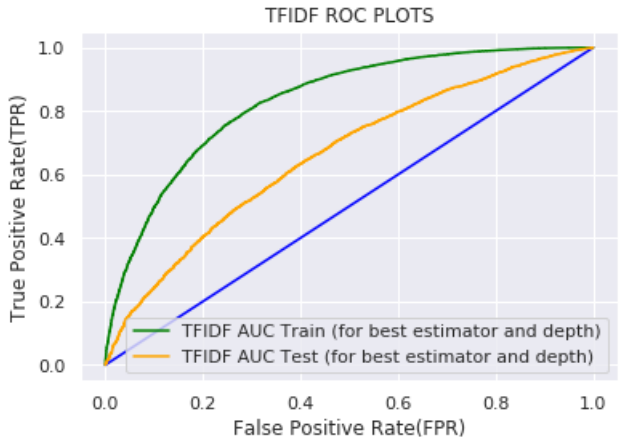
CPU times: user 8min 17s, sys: 254 ms, total: 8min 17s
Wall time: 8min 17s

ROC PLOT

In [0]:

```
%%time
#https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-al
gorithm-using-python-and-sci
plt.plot([0,1],[0,1],'k-', color='blue')
plt.plot(fpr_train, tpr_train, label="TFIDF AUC Train (for best estimator and depth)",
color='green')
plt.plot(fpr_Test, tpr_Test, label="TFIDF AUC Test (for best estimator and depth)", color='orange'
)
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("TFIDF ROC PLOTS")
plt.show()
print("-"*115)
print("AUC Train (for best estimator and depth) =", auc(fpr_train, tpr_train))
print("AUC Test (for best estimator and depth) =", auc(fpr_Test, tpr_Test))
```

```
TFIDF_GBDT_AUC=round(auc(fpr_Test, tpr_Test)*100)
pred1 = TFIDF_GBDT_O_CLF.predict(TFIDF_Train)
pred2 = TFIDF_GBDT_O_CLF.predict(TFIDF_Test)
```




```
------------------------------------------------------------------------------------------------------------
AUC Train (for best estimator and depth) = 0.83113411064781
AUC Test (for best estimator and depth) = 0.6605645597911773
CPU times: user 917 ms, sys: 6 ms, total: 923 ms
Wall time: 919 ms
```

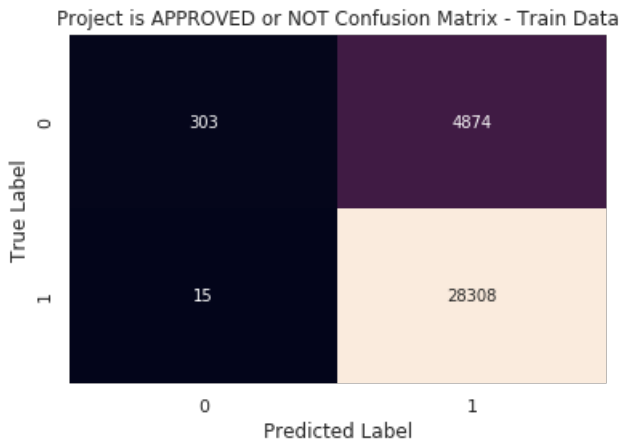
Confusion Matrix

In [0]:

```
#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.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Train Data')
```

Out[0]:

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



Observations for Train data: True Positives - 28308, True Negatives - 303, False Positives - 4874, False Negatives - 15.

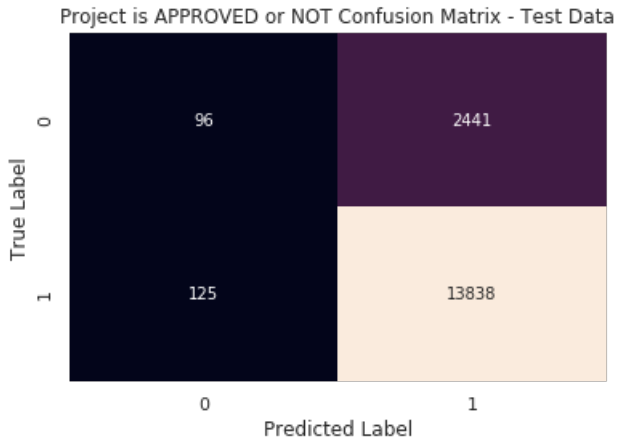
In [0]:

```
#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
Test = confusion_matrix(Y_Test, pred2)
sns.heatmap(Test,annot=True,cbar=False,fmt='g')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Test Data')
```

Out[0]:

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



Observations for Test data: True Positives - 13838, True Negatives - 96, False Positives - 2441, False Negatives - 125.

2.4.7 Applying GBDT on AVG_W2V, SET 3

In [0]:

```
%%time
GBDT_clf = GradientBoostingClassifier()
parameters = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth':[2, 3, 4, 5, 6, 7
, 8, 9, 10]}
GBDTGrid_clf = GridSearchCV(GBDT_clf, parameters, cv=3, scoring='roc_auc',n_jobs=-1,return_train_sc
ore=True,verbose=1)
GBDTGrid_clf.fit(AVG_W2V_Train, Y_Train)
print(GBDTGrid_clf.best_estimator_)
a13=GBDTGrid_clf.best_params_['n_estimators']
a14=GBDTGrid_clf.best_params_['max_depth']
AUC_Train= GBDTGrid_clf.cv_results_['mean_train_score']
AUC_Cv = GBDTGrid_clf.cv_results_['mean_test_score']
```

```
Fitting 3 folds for each of 72 candidates, totalling 216 fits
```

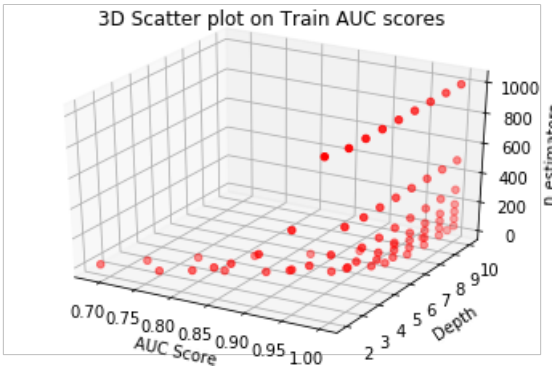
```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 40 concurrent workers.
[Parallel(n_jobs=-1)]: Done 120 tasks      | elapsed: 60.0min
[Parallel(n_jobs=-1)]: Done 216 out of 216 | elapsed: 199.1min finished
```

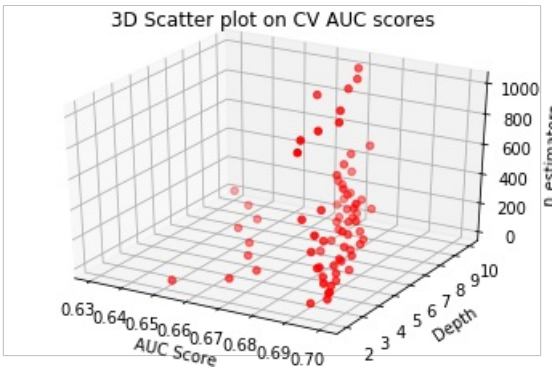
```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=2,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=150,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
CPU times: user 5min 43s, sys: 781 ms, total: 5min 44s
Wall time: 3h 24min 46s
```

3D Plot

In [0]:

```
%%time
#https://stackoverflow.com/questions/53311685/difference-between-ax-set-xlabel-and-ax-xaxis-set-la
bel-in-matplotlib-3-0-1
#----------------------------------------------------------3D-Plot for Train Dataset-----------------
----------------------------------------------------------
figure = plt.figure()
ax = figure.add_subplot(111, projection='3d')
ax.scatter(AUC_Train, ES, MD, c='r', marker='o')
ax.set_xlabel('AUC Score')
ax.yaxis.set_label_text('Depth')
ax.zaxis.set_label_text('n_estimators')
plt.title('3D Scatter plot on Train AUC scores')
plt.show()
plt.close()

#----------------------------------------------------------3D-Plot for CV Dataset--------------------
----------------------------------------------------
figure = plt.figure()
ax = figure.add_subplot(111, projection='3d')

ax.scatter(AUC_Cv, ES, MD, c='r', marker='o')
ax.set_xlabel('AUC Score')
ax.yaxis.set_label_text('Depth')
ax.zaxis.set_label_text('n_estimators')
plt.title('3D Scatter plot on CV AUC scores')
plt.show()
plt.close()
```






```
CPU times: user 1.42 s, sys: 2.88 s, total: 4.3 s
Wall time: 570 ms
```

In [0]:

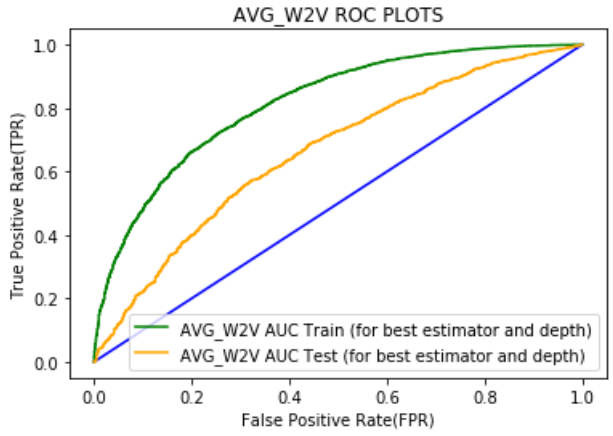
```
%%time
AVG_W2V_GBDT_O_CLF = GradientBoostingClassifier(n_estimators=a13,max_depth=a14)
AVG_W2V_GBDT_O_CLF.fit(AVG_W2V_Train, Y_Train)
pred = AVG_W2V_GBDT_O_CLF.predict(AVG_W2V_Test)

fpr_train, tpr_train, thresholds = roc_curve(Y_Train,
AVG_W2V_GBDT_O_CLF.predict_proba(AVG_W2V_Train)[:,1])
fpr_Test, tpr_Test, thresholds = roc_curve(Y_Test, AVG_W2V_GBDT_O_CLF.predict_proba(AVG_W2V_Test)[:
,1])
```

```
CPU times: user 5min 42s, sys: 108 ms, total: 5min 42s
Wall time: 5min 42s
```

ROC PLOT

In [0]:

```
%%time
#https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-algorithm-using-python-and-sci
plt.plot([0,1],[0,1],'k-', color='blue')
plt.plot(fpr_train, tpr_train, label="AVG_W2V AUC Train (for best estimator and depth)", color='green')
plt.plot(fpr_Test, tpr_Test, label="AVG_W2V AUC Test (for best estimator and depth)",
color='orange')
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("AVG_W2V ROC PLOTS")
plt.show()
print("-"*115)
print("AUC Train (for best estimator and depth) =", auc(fpr_train, tpr_train))
print("AUC Test (for best estimator and depth) =", auc(fpr_Test, tpr_Test))

AVG_W2V_GBDT_AUC=round(auc(fpr_Test, tpr_Test)*100)
pred1 = AVG_W2V_GBDT_O_CLF.predict(AVG_W2V_Train)
pred2 = AVG_W2V_GBDT_O_CLF.predict(AVG_W2V_Test)
```



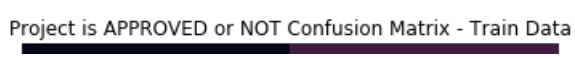

```
--------------------------------------------------------------------------------------------------------------
------------
AUC Train (for best estimator and depth) = 0.8159421756820631
AUC Test (for best estimator and depth) = 0.6621092671944222
CPU times: user 586 ms, sys: 9.03 ms, total: 595 ms
Wall time: 591 ms
```

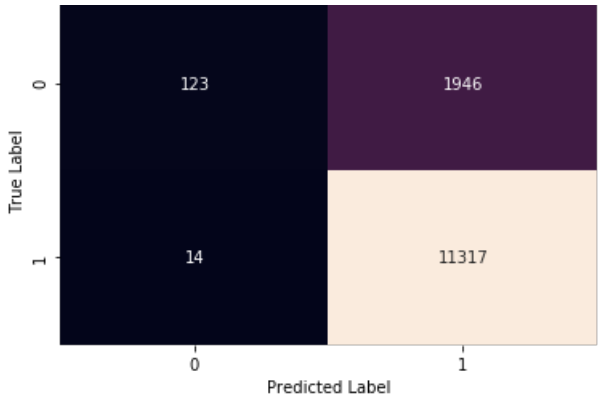
In [0]:

```
#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.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Train Data')
```

Out[0]:

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





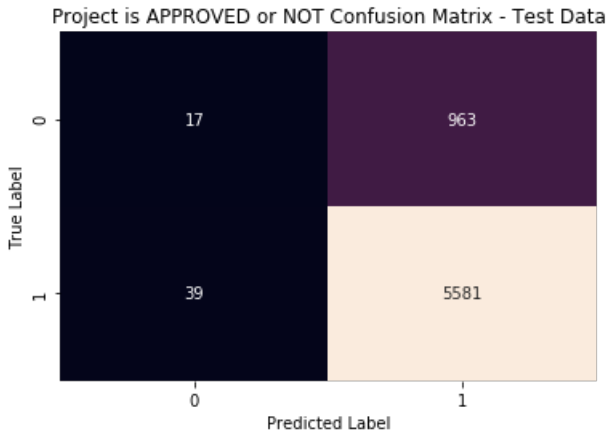
Observations for Train data: True Positives - 11317, True Negatives - 123, False Positives - 1946, False Negatives - 14.

In [0]:

```
#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
Test = confusion_matrix(Y_Test, pred2)
sns.heatmap(Test,annot=True,cbar=False,fmt='g')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Test Data')
```

Out[0]:

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



Observations for Test data: True Positives - 5581, True Negatives - 17, False Positives - 963, False Negatives - 39.

2.4.8 Applying GBDT on TFIDF_W2V, SET 4

In [42]:

```
%%time
GBDT_clf = GradientBoostingClassifier()
parameters = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth':[2, 3, 4, 5, 6, 7
, 8, 9, 10]}
GBDTGrid_clf = GridSearchCV(GBDT_clf, parameters, cv=3, scoring='roc_auc',n_jobs=-1,return_train_sc
ore=True,verbose=1)
GBDTGrid_clf.fit(TFIDF_W2V_Train, Y_Train)
print(GBDTGrid_clf.best_estimator_)
a15=GBDTGrid_clf.best_params_['n_estimators']
a16=GBDTGrid_clf.best_params_['max_depth']
AUC_Train= GBDTGrid_clf.cv_results_['mean_train_score']
AUC_Cv = GBDTGrid_clf.cv_results_['mean_test_score']
```

```
Fitting 3 folds for each of 72 candidates, totalling 216 fits
```

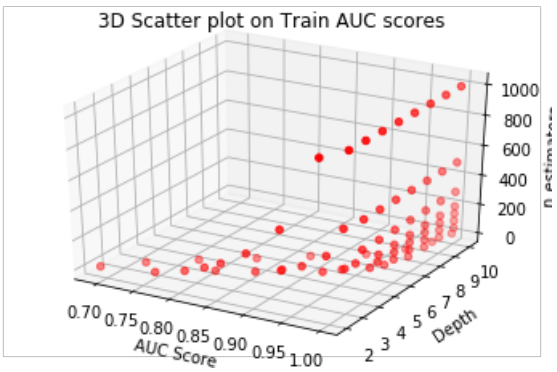
```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 40 concurrent workers.
[Parallel(n_jobs=-1)]: Done 120 tasks      | elapsed: 29.7min
[Parallel(n_jobs=-1)]: Done 216 out of 216 | elapsed: 82.3min finished
```

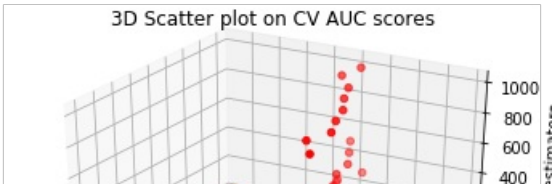
```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=2,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=300,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
CPU times: user 5min 45s, sys: 1.11 s, total: 5min 46s
Wall time: 1h 28min
```

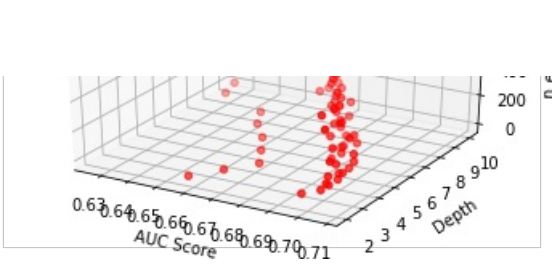
3D Plot

In [43]:

```
%%time
#https://stackoverflow.com/questions/53311685/difference-between-ax-set-xlabel-and-ax-xaxis-set-label-in-matplotlib-3-0-1
#-----------------------------------------------------------3D-Plot for Train Dataset-----------------------------------------------------------
figure = plt.figure()
ax = figure.add_subplot(111, projection='3d')
ax.scatter(AUC_Train, ES, MD, c='r', marker='o')
ax.set_xlabel('AUC Score')
ax.yaxis.set_label_text('Depth')
ax.zaxis.set_label_text('n_estimators')
plt.title('3D Scatter plot on Train AUC scores')
plt.show()
plt.close()

#-----------------------------------------------------------3D-Plot for CV Dataset-----------------------------------------------------------
figure = plt.figure()
ax = figure.add_subplot(111, projection='3d')

ax.scatter(AUC_Cv, ES, MD, c='r', marker='o')
ax.set_xlabel('AUC Score')
ax.yaxis.set_label_text('Depth')
ax.zaxis.set_label_text('n_estimators')
plt.title('3D Scatter plot on CV AUC scores')
plt.show()
plt.close()
```





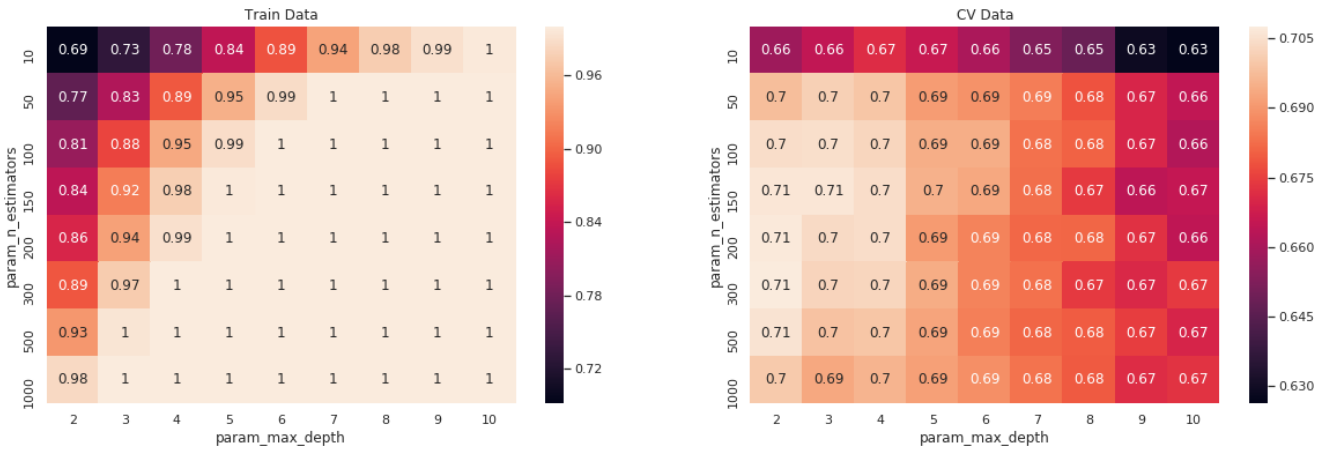



```
CPU times: user 1.01 s, sys: 3.18 s, total: 4.18 s
Wall time: 393 ms
```

Heat Map

In [44]:

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



In [45]:

```
%%time
TFIDF_W2V_GBDT_O_CLF = GradientBoostingClassifier(n_estimators=a15,max_depth=a16)
TFIDF_W2V_GBDT_O_CLF.fit(TFIDF_W2V_Train, Y_Train)
pred = TFIDF_W2V_GBDT_O_CLF.predict(TFIDF_W2V_Test)

fpr_train, tpr_train, thresholds = roc_curve(Y_Train,
TFIDF_W2V_GBDT_O_CLF.predict_proba(TFIDF_W2V_Train)[:,1])
fpr_Test, tpr_Test, thresholds = roc_curve(Y_Test, TFIDF_W2V_GBDT_O_CLF.predict_proba(TFIDF_W2V_Tes
t)[:,1])
```

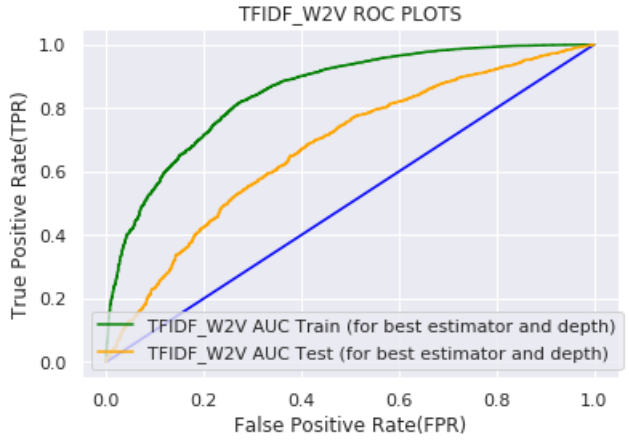
```
CPU times: user 5min 46s, sys: 135 ms, total: 5min 46s
Wall time: 5min 46s
```

ROC PLOT

In [46]:

```
%%time
#https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-al
gorithm-using-python-and-sci
plt.plot([0,1],[0,1],'k-', color='blue')
plt.plot(fpr_train, tpr_train, label="TFIDF_W2V AUC Train (for best estimator and depth)", color='
green')
```

```
plt.plot(fpr_Test, tpr_Test, label="TFIDF_W2V AUC Test (for best estimator and depth)",
color='orange')
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("TFIDF_W2V ROC PLOTS")
plt.show()
print("-"*115)
print("AUC Train (for best estimator and depth) =", auc(fpr_train, tpr_train))
print("AUC Test (for best estimator and depth) =", auc(fpr_Test, tpr_Test))

TFIDF_W2V_GBDT_AUC=round(auc(fpr_Test, tpr_Test)*100)
pred1 = TFIDF_W2V_GBDT_O_CLF.predict(TFIDF_W2V_Train)
pred2 = TFIDF_W2V_GBDT_O_CLF.predict(TFIDF_W2V_Test)
```



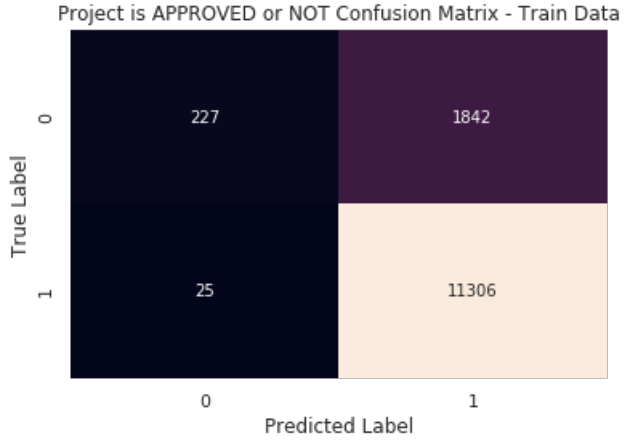

```
---------------------------------------------------------------------------------------------------------
------------
AUC Train (for best estimator and depth) = 0.8512879225966361
AUC Test (for best estimator and depth) = 0.6758116057811026
CPU times: user 613 ms, sys: 5.01 ms, total: 618 ms
Wall time: 614 ms
```

In [47]:

```
#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.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Train Data')
```

Out[47]:

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



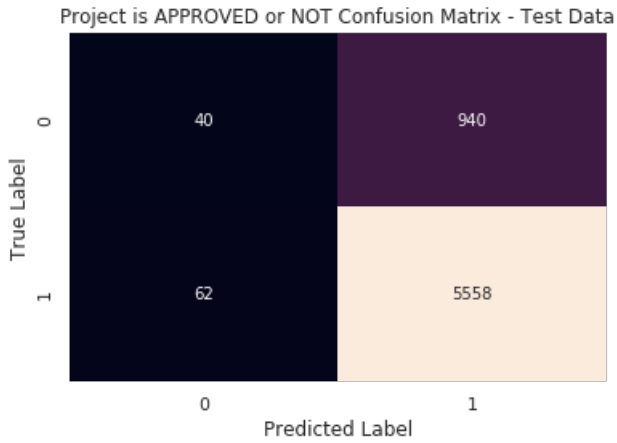
Observations for Train data: True Positives - 11306, True Negatives - 227, False Positives - 1842, False Negatives - 25.

In [48]:

```
#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
Test = confusion_matrix(Y_Test, pred2)
sns.heatmap(Test,annot=True,cbar=False,fmt='g')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Test Data')
```

Out[48]:

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



Observations for Test data: True Positives - 5558, True Negatives - 40, False Positives - 940, False Negatives - 32.

Conclusion

In [52]:

```
# Please compare all your models using Prettytable library

pt = PrettyTable()
pt.field_names= ("S.No","Vectorizer", "Model", "No of Estimators","Max Depth" , "Test AUC(%)")
pt.add_row(["1","BOW", "Random Forest","1000","10", "68.36"])
pt.add_row(["2","TFIDF", "Random Forest", "1000","10", "68.73"])
pt.add_row(["3","AVG_W2V", "Random Forest","1000","6", "66.59"])
pt.add_row(["4","TFIDF_W2V", "Random Forest", "500","5", "67.37"])
print(pt)

pt = PrettyTable()
pt.field_names= ("S.No","Vectorizer", "Model", "No of Estimators","Max Depth", "Test AUC(%)")
pt.add_row(["1","BOW", "Gradient Boosted Decision Tree","1000","2", "68.08"])
pt.add_row(["2","TFIDF", "Gradient Boosted Decision Tree", "500","2", "66.05"])
pt.add_row(["3","AVG_W2V", "Gradient Boosted Decision Tree","150","2", "66.21"])
pt.add_row(["4","TFIDF_W2V", "Gradient Boosted Decision Tree", "300","2", "67.58"])
print(pt)
```

```
+------+------------+---------------+------------------+-----------+-------------+
| S.No | Vectorizer |     Model     | No of Estimators | Max Depth | Test AUC(%) |
+------+------------+---------------+------------------+-----------+-------------+
|  1   |    BOW     | Random Forest |       1000       |     10    |    68.36    |
|  2   |   TFIDF    | Random Forest |       1000       |     10    |    68.73    |
|  3   |  AVG_W2V   | Random Forest |       1000       |     6     |    66.59    |
|  4   | TFIDF_W2V  | Random Forest |       500        |     5     |    67.37    |
+------+------------+---------------+------------------+-----------+-------------+
+------+------------+-------------------------------+------------------+-----------+-------------+
| S.No | Vectorizer |             Model             | No of Estimators | Max Depth | Test AUC(%) |
+------+------------+-------------------------------+------------------+-----------+-------------+
|  1   |    BOW     | Gradient Boosted Decision Tree |       1000       |     2     |    68.08    |
```

2	TFIDF	Gradient Boosted Decision Tree	500	2	66.05
3	AVG_W2V	Gradient Boosted Decision Tree	150	2	66.21
4	TFIDF_W2V	Gradient Boosted Decision Tree	300	2	67.58

SUMMARY:

1. Sets using BOW and TFIDF (for RF and GBDT) use 50000 datapoints and sets using AvgW2V and TFIDF_W2V (for RF and GBDT) use 20000 datapoints due to time and space constraints.
2. Both the models RF and GBDT using all four vectorizers yield almost same Test AUC scores.
3. In comparison with the RF models the GBDT model takes up considerably more space and time.
4. For the majority 'no_of_estimators' and 'max_depth' is lower for GBDT when compared to RF models.
5. Both the models - RF and GBDT using BOW, RF using TFIDF vectorization performs silghtly better than their counterparts.