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]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
```

```
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

import pickle

from tqdm import tqdm
import os

from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

1.1 Reading Data

In [2]:

```
project_data = pd.read_csv('train_data.csv',nrows=10000)
resource_data = pd.read_csv('resources.csv')
```

In [3]:

```
project_data.shape
```

Out[3]:

```
(10000, 17)
```

In [4]:

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

Out[4]:

```
1    8500
0    1500
Name: project_is_approved, dtype: int64
```

In [5]:

```
resource_data.shape
```

Out[5]:

```
(1541272, 4)
```

In [6]:

```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
```

```
Number of data points in train data (10000, 17)
--------------------------------------------------
```

```
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [7]:

```
# 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[7]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_s
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2	
7176	79341	p091436	bb2599c4a114d211b3381abe9f899bf8	Mrs.	OH	2016-04-27 07:24:47	Grades PreK-2	Math

In [8]:

```
print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

```
Number of data points in train data (1541272, 4)
['id' 'description' 'quantity' 'price']
```

Out[8]:

	id	description	quantity	price
0	p233245	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.00
1	p069063	Bouncy Bands for Desks (Blue support pipes)	3	14.95

1.2 preprocessing of `project_subject_categories`

In [9]:

```
print(project_data['project_subject_categories'].head(5))
```

```
473                        Applied Learning
7176      Math & Science, Applied Learning
5145                    Literacy & Language
2521                    Literacy & Language
5364    Applied Learning, Music & The Arts
Name: project_subject_categories, dtype: object
```

In [10]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

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

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

In [11]:

```
print(project_data['clean_categories'].head(5))
```

```
473                  AppliedLearning
7176    Math_Science AppliedLearning
5145                Literacy_Language
2521                Literacy_Language
5364      AppliedLearning Music_Arts
Name: clean_categories, dtype: object
```

1.3 preprocessing of `project_subject_subcategories`

In [12]:

```
print(project_data['project_subject_subcategories'].head(5))
```

```
473                         Early Development
7176    Applied Sciences, Early Development
5145                                Literacy
2521          Literacy, Literature & Writing
5364     College & Career Prep, Visual Arts
Name: project_subject_subcategories, dtype: object
```

In [13]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
```

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

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

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

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

In [14]:

```
print(project_data['clean_subcategories'].head(5))
```

```
473                     EarlyDevelopment
7176    AppliedSciences EarlyDevelopment
5145                            Literacy
2521         Literacy Literature_Writing
5364       College_CareerPrep VisualArts
Name: clean_subcategories, dtype: object
```

1.4 preprocessing of `school_state`

In [15]:

```
my_counter = Counter()
for word in project_data['school_state'].values:
    my_counter.update(word.split())

state_dict = dict(my_counter)
sorted_state_dict = dict(sorted(state_dict.items(), key=lambda kv: kv[1]))
```

In [16]:

```
print(project_data['school_state'].head(5))
```

```
473     GA
7176    OH
5145    CA
2521    NJ
5364    CO
Name: school_state, dtype: object
```

1.5 preprocessing of `project_grade_category`

In [17]:

```
preproc = []
# tqdm is for printing the status bar
for sent in project_data['project_grade_category']:
    sent = sent.replace('Grades ', '')
```

```
    sent = sent.replace('PreK-2', 'PreKto2')
    sent = sent.replace('3-5', '3to5')
    sent = sent.replace('6-8', '6to8')
    sent = sent.replace('9-12', '9to12')
    preproc.append(sent)
project_data['project_grade_category']=preproc
```

In [18]:

```
my_counter = Counter()
for word in project_data['project_grade_category'].values:
    my_counter.update(word.split())

grade_dict = dict(my_counter)
sorted_grade_dict = dict(sorted(grade_dict.items(), key=lambda kv: kv[1]))
```

In [19]:

```
print(project_data['project_grade_category'].head(5))
```

```
473     PreKto2
7176    PreKto2
5145       3to5
2521    PreKto2
5364       6to8
Name: project_grade_category, dtype: object
```

1.6 preprocessing of `teacher_prefix`

In [20]:

```
print(type(project_data['teacher_prefix']))
```

```
<class 'pandas.core.series.Series'>
```

In [21]:

```
project_data['teacher_prefix'] = project_data['teacher_prefix'].astype(str)
preproc = []
# tqdm is for printing the status bar
for sent in project_data['teacher_prefix']:
    sent = sent.replace('Mr.', 'Mr')
    sent = sent.replace('Mrs.', 'Mrs')
    sent = sent.replace('Dr.', 'Dr')
    sent = sent.replace('Ms.', 'Ms')
    sent = sent.replace('nan','Dr')
    preproc.append(sent)
project_data['teacher_prefix']=preproc
```

In [22]:

```
#['Teacher', 'Mrs.', 'Dr.', 'Mr.', 'Ms.']
project_data['teacher_prefix']=project_data['teacher_prefix'].fillna('')
my_counter = Counter()
for word in project_data['teacher_prefix'].values:
    my_counter.update(word.split())

teacher_dict = dict(my_counter)
sorted_teacher_dict = dict(sorted(teacher_dict.items(), key=lambda kv: kv[1]))
```

In [23]:

```
print(project_data['teacher_prefix'].head(5))
```

```
473     Mrs
7176    Mrs
5145    Mrs
2521     Ms
```

```
5364      Mr
Name: teacher_prefix, dtype: object
```

1.3 Preprocessing of Essays

In [24]:

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

In [25]:

```
# 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 [26]:

```
# 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 [27]:

```
# Combining all the above stundents
from tqdm import tqdm
```

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

```
100%|██████████| 10000/10000 [00:05<00:00, 1725.32it/s]
```

In [28]:

```
# after preprocesing
preprocessed_essays[2000]
```

Out[28]:

```
'feel truly blessed teach diverse dynamic group fifth graders love learn fifth graders incredibly c
reative inquisitive energetic deserve classroom thrive try provide learning environment beneficial
students students needs classroom vary greatly research shows students engaged comfortable
flexible environment likely take learning risks observing researching flexible seating truly belie
ve students become even better learners given opportunity choose seat comfortable using know stude
nts greatly benefit design classroom atmosphere year ask like sit completing challenging task woul
d respond something comfortable however expect students sit hard chairs best work assignments asse
ssments one day alone could observe class engaged small cooperative group discussions exploring sc
ientific investigations using critical thinking skill math centers scrawled floor offering wobble
chairs floor cushions chair stability cushions students gives opportunity choose seat comfortable
learning items help increase focus encourage students learn benefiting active sitting offering cho
ice important role student success help project students comfortable engage fully learning
experience dive deeper education nannan'
```

In [29]:

```
project_data['essay']=preprocessed_essays
```

1.4 Preprocessing of `project_title`

In [30]:

```
# similarly you can preprocess the titles also
preprocessed_titles = []
# tqdm is for printing the status bar
for sentence in tqdm(project_data['project_title'].values):
    sent = decontracted(sentence)
    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.append(sent.lower().strip())
```

```
100%|██████████| 10000/10000 [00:00<00:00, 35534.53it/s]
```

In [31]:

```
project_data['project_title']=preprocessed_titles
```

Sentiment Analysis of essays

In [32]:

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

```
analyser = SentimentIntensityAnalyzer()

neg = []
pos = []
neu = []
compound = []

for a in tqdm(project_data["essay"]) :
    b = analyser.polarity_scores(a)['neg']
    c = analyser.polarity_scores(a)['pos']
    d = analyser.polarity_scores(a)['neu']
    e = analyser.polarity_scores(a)['compound']
    neg.append(b)
    pos.append(c)
    neu.append(d)
    compound.append(e)
```

```
100%|██████████| 10000/10000 [00:59<00:00, 167.59it/s]
```

In [33]:

```
project_data["pos"] = pos
```

In [34]:

```
project_data["neg"] = neg
```

In [35]:

```
project_data["neu"] = neu
```

In [36]:

```
project_data["compound"] = compound
```

Number of Words in Title

In [37]:

```
title_word_count = []

for a in project_data["project_title"] :
    b = len(a.split())
    title_word_count.append(b)

project_data["title_word_count"] = title_word_count
```

Number of Words in Essays

In [38]:

```
essay_word_count = []

for a in project_data["essay"] :
    b = len(a.split())
    essay_word_count.append(b)

project_data["essay_word_count"] = essay_word_count
```

1.5 Preparing data for models

In [39]:

```
project_data.columns
```

Out[39]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_title', 'project_essay_1',
       'project_essay_2', 'project_essay_3', 'project_essay_4',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'pos', 'neg', 'neu',
       'compound', 'title_word_count', 'essay_word_count'],
      dtype='object')
```

we are going to consider

- school_state : categorical data
- clean_categories : categorical data
- clean_subcategories : categorical data
- project_grade_category : categorical data
- teacher_prefix : categorical data

- project_title : text data
- text : text data
- project_resource_summary: text data (optinal)

- quantity : numerical (optinal)
- teacher_number_of_previously_posted_projects : numerical
- price : numerical

In [40]:

```
project_data['text']=project_data["essay"].map(str) +project_data["project_title"].map(str)
```

Assignment 11: TruncatedSVD

- step 1 Select the top 2k words from essay text and project_title (concatinate essay text with project title and then find the top 2k words) based on their `idf_` values
- step 2 Compute the co-occurance matrix with these 2k words, with window size=5 (ref)
- step 3 Use TruncatedSVD on calculated co-occurance matrix and reduce its dimensions, choose the number of components (`n_components`) using elbow method

  > - The shape of the matrix after TruncatedSVD will be 2000*n, i.e. each row represents a vector form of the corresponding word.
  > - Vectorize the essay text and project titles using these word vectors. (while vectorizing, do ignore all the words which are not in top 2k words)

- step 4 Concatenate these truncatedSVD matrix, with the matrix with features
  - **school_state** : categorical data
  - **clean_categories** : categorical data
  - **clean_subcategories** : categorical data
  - **project_grade_category** :categorical data
  - **teacher_prefix** : categorical data
  - **quantity** : numerical data
  - **teacher_number_of_previously_posted_projects** : numerical data
  - **price** : numerical data
  - **sentiment score's of each of the essay** : numerical data
  - **number of words in the title** : numerical data
  - **number of words in the combine essays** : numerical data
  - **word vectors calculated in** step 3 : numerical data
- step 5: Apply GBDT on matrix that was formed in step 4 of this assignment, **DO REFER THIS BLOG: XGBOOST DMATRIX**
- **step 6:Hyper parameter tuning (Consider any two hyper parameters)**
  - **Find the best hyper parameter which will give the maximum AUC value**
  - **Find the best hyper paramter using k-fold cross validation or simple cross validation data**
  - **Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning**

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.**

TruncatedSVD

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

In [41]:

```
y = project_data['project_is_approved']
print(y.shape)
```

```
(10000,)
```

In [42]:

```
project_data.drop(['project_is_approved'],axis=1,inplace=True)
```

In [43]:

```
X=project_data
print(X.shape)
```

```
(10000, 24)
```

In [44]:

```
#train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
```

2.2 Make Data Model Ready: encoding eassay, and project_title

In [45]:

```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
print("="*100)
```

```
(6700, 24) (6700,)
(3300, 24) (3300,)
====================================================================================================
```

Encoding of Text Data

2.3 Make Data Model Ready: encoding numerical and categorical features

Vectorizing Numerical features

In [46]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
```

In [47]:

```
price_data.head(5)
```

Out[47]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21
2	p000003	298.97	4
3	p000004	1113.69	98
4	p000005	485.99	8

In [48]:

```
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 [49]:

```
X_train=X_train.fillna(0)
X_test=X_test.fillna(0)
```

Normalizing the numerical features: Price

In [50]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['price'].values.reshape(-1,1))
X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(-1,1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(-1,1))
print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(6700, 1) (6700,)
(3300, 1) (3300,)
====================================================================================================
```

Normalizing the numerical features: Number of previously posted projects

In [51]:

```
normalizer = Normalizer()
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_train_project_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'
].values.reshape(-1,1))
X_test_project_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].
values.reshape(-1,1))
print("After vectorizations")
print(X_train_project_norm.shape, y_train.shape)
print(X_test_project_norm.shape, y_test.shape)
```

```
print("="*100)
```

```
After vectorizations
(6700, 1) (6700,)
(3300, 1) (3300,)
====================================================================================================
```

Normalizing the numerical features: Title word Count

In [52]:

```
normalizer = Normalizer()
normalizer.fit(X_train['title_word_count'].values.reshape(-1,1))
X_train_title_norm = normalizer.transform(X_train['title_word_count'].values.reshape(-1,1))
X_test_title_norm = normalizer.transform(X_test['title_word_count'].values.reshape(-1,1))
print("After vectorizations")
print(X_train_title_norm.shape, y_train.shape)
print(X_test_title_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(6700, 1) (6700,)
(3300, 1) (3300,)
====================================================================================================
```

Normalizing the numerical features: Essay word Count

In [53]:

```
normalizer = Normalizer()
normalizer.fit(X_train['essay_word_count'].values.reshape(-1,1))
X_train_essay_norm = normalizer.transform(X_train['essay_word_count'].values.reshape(-1,1))
X_test_essay_norm = normalizer.transform(X_test['essay_word_count'].values.reshape(-1,1))
print("After vectorizations")
print(X_train_essay_norm.shape, y_train.shape)
print(X_test_essay_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(6700, 1) (6700,)
(3300, 1) (3300,)
====================================================================================================
```

Normalizing the numerical features: Essay Sentiments-Positive

In [54]:

```
normalizer = Normalizer()
normalizer.fit(X_train['pos'].values.reshape(-1,1))
essay_sent_pos_train = normalizer.transform(X_train['pos'].values.reshape(-1,1))
essay_sent_pos_test = normalizer.transform(X_test['pos'].values.reshape(-1,1))
print("After vectorizations")
print(essay_sent_pos_train.shape, y_train.shape)
print(essay_sent_pos_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(6700, 1) (6700,)
(3300, 1) (3300,)
====================================================================================================
```

Normalizing the numerical features: Essay Sentiments-Negative

In [55]:

```
normalizer = Normalizer()
normalizer.fit(X_train['neg'].values.reshape(-1,1))
essay_sent_neg_train = normalizer.transform(X_train['neg'].values.reshape(-1,1))
essay_sent_neg_test = normalizer.transform(X_test['neg'].values.reshape(-1,1))
print("After vectorizations")
print(essay_sent_neg_train.shape, y_train.shape)
print(essay_sent_neg_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(6700, 1) (6700,)
(3300, 1) (3300,)
====================================================================================================
```

Normalizing the numerical features: Essay Sentiments-Neutral

In [56]:

```
normalizer = Normalizer()
normalizer.fit(X_train['neu'].values.reshape(-1,1))
essay_sent_neu_train = normalizer.transform(X_train['neu'].values.reshape(-1,1))
essay_sent_neu_test = normalizer.transform(X_test['neu'].values.reshape(-1,1))
print("After vectorizations")
print(essay_sent_neu_train.shape, y_train.shape)
print(essay_sent_neu_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(6700, 1) (6700,)
(3300, 1) (3300,)
====================================================================================================
```

Normalizing the numerical features: Essay Sentiments-Compound

In [57]:

```
normalizer = Normalizer()
normalizer.fit(X_train['compound'].values.reshape(-1,1))
essay_sent_comp_train = normalizer.transform(X_train['compound'].values.reshape(-1,1))
essay_sent_comp_test = normalizer.transform(X_test['compound'].values.reshape(-1,1))
print("After vectorizations")
print(essay_sent_comp_train.shape, y_train.shape)
print(essay_sent_comp_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(6700, 1) (6700,)
(3300, 1) (3300,)
====================================================================================================
```

Vectorizing Categorical features

- **school_state : categorical data**
- **clean_categories : categorical data**
- **clean_subcategories : categorical data**
- **project_grade_category : categorical data**
- **teacher_prefix : categorical data**

Vectorizing Categorical features: project grade category

In [58]:

```
from sklearn.feature_extraction.text import CountVectorizer
```

In [59]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_grade_dict.keys()), lowercase=False, binary=Tr
ue)
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(6700, 4) (6700,)
(3300, 4) (3300,)
['9to12', '6to8', '3to5', 'PreKto2']
====================================================================================================
```

Vectorizing Categorical features: teacher prefix

In [60]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_teacher_dict.keys()), lowercase=False, binary=
True)
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(6700, 5) (6700,)
(3300, 5) (3300,)
['Dr', 'Teacher', 'Mr', 'Ms', 'Mrs']
====================================================================================================
```

Vectorizing Categorical features: school state

In [61]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_state_dict.keys()), lowercase=False, binary=Tr
ue)
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(6700, 51) (6700,)
(3300, 51) (3300,)
['VT', 'WY', 'ND', 'MT', 'NH', 'DE', 'SD', 'RI', 'NE', 'AK', 'NM', 'ME', 'DC', 'HI', 'WV', 'ID', 'I
A', 'KS', 'AR', 'MN', 'MS', 'OR', 'CO', 'KY', 'NV', 'MD', 'AL', 'TN', 'CT', 'WI', 'UT', 'VA', 'WA',
'NJ', 'MA', 'AZ', 'LA', 'OK', 'MO', 'IN', 'OH', 'PA', 'MI', 'GA', 'SC', 'IL', 'NC', 'FL', 'TX', 'NY
', 'CA']
====================================================================================================
```

Vectorizing Categorical features: clean categories

In [62]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True
)
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_cat_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_test_cat_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_cat_ohe.shape, y_train.shape)
print(X_test_cat_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(6700, 9) (6700,)
(3300, 9) (3300,)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
====================================================================================================
```

Vectorizing Categorical features: clean subcategories

In [63]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=
True)
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_sub_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
X_test_sub_ohe = vectorizer.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_sub_ohe.shape, y_train.shape)
print(X_test_sub_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(6700, 30) (6700,)
(3300, 30) (3300,)
['Economics', 'FinancialLiteracy', 'CommunityService', 'ForeignLanguages', 'Extracurricular',
'ParentInvolvement', 'Civics_Government', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'CharacterEducation', 'PerformingArts', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'ESL', 'Health_LifeScience',
'EarlyDevelopment', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness',
'AppliedSciences', 'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
====================================================================================================
```

Applying Truncated SVD on TFIDF of Text Data(Best 2000

features)

In [64]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

TFIDF of Text

In [65]:

```
vectorizer = TfidfVectorizer()
```

In [66]:

```
text_tfidf = vectorizer.fit_transform(project_data["text"])
```

In [67]:

```
print("After vectorization")
print(text_tfidf.shape)
```

```
After vectorization
(10000, 25556)
```

Select Top 2000 features

In [68]:

```
index = np.argsort(vectorizer.idf_)
text_features = vectorizer.get_feature_names()
top_features = [text_features[i] for i in index[:20]]# Top 20 features
print(top_features)
```

```
['students', 'school', 'learning', 'classroom', 'not', 'learn', 'help', 'many', 'need', 'work', 'c
ome', 'use', 'love', 'day', 'able', 'also', 'class', 'make', 'year', 'new']
```

In [69]:

```
# Top 2000 features
top_features = [text_features[i] for i in index[:2000]]
print(len(top_features))
```

```
2000
```

In [70]:

```
vocab = sorted(top_features)
```

Create Co-occurence Matrix

In [71]:

```
# https://github.com/vineet28/Truncated-SVD-Donors-
Choose/blob/master/11_DonorsChoose_TruncatedSVD.ipynb
window = 5
occ_matrix_2000 = np.zeros((2000,2000))
for row in tqdm(project_data["text"].values):
    words_in_row = row.split()
    for index,word in enumerate(words_in_row):
        if word in top_features:
            for j in range(max(index-window,0),min(index+window,len(words_in_row)-1) + 1):
                if words_in_row[j] in top_features:
                    occ_matrix_2000[top_features.index(word),top_features.index(words_in_row[j])] +
```

```
= 1
                else:
                    pass
        else:
            pass
```

```
100%|██████████| 10000/10000 [04:01<00:00, 41.39it/s]
```

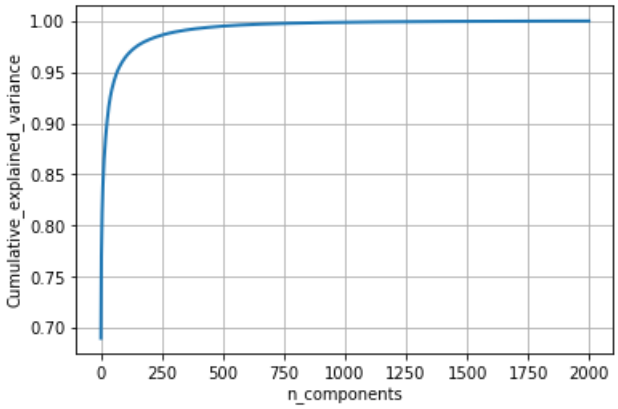
Apply SVD on Co-occurence Matrix

In [72]:

```
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import StandardScaler
svd = TruncatedSVD(n_components = 1999)
svd_2000 = svd.fit_transform(occ_matrix_2000)

percentage_var_explained = svd.explained_variance_ / np.sum(svd.explained_variance_);
cum_var_explained = np.cumsum(percentage_var_explained)
plt.figure(figsize=(6, 4))

plt.clf()
plt.plot(cum_var_explained, linewidth=2)
plt.axis('tight')
plt.grid()
plt.xlabel('n_components')
plt.ylabel('Cumulative_explained_variance')
plt.show()
```



In [73]:

```
svd = TruncatedSVD(n_components = 250)##98% variance preserved
svd_2000 = svd.fit_transform(occ_matrix_2000)
```

In [74]:

```
svd_2000.shape
```

Out[74]:

```
(2000, 250)
```

Avg-W2V on Essays and Titles(from SVD Matrix)

Essay Train

In [75]:

```
train_w2v_vectors_essays = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_train["essay"].values): # for each essay in training data
    vector = np.zeros(250) # as word vectors are of zero length
```

```
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in top_features:
            i=top_features.index(word)
            vector += svd_2000[i]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_w2v_vectors_essays.append(vector)
print("train  vector")
print(len(train_w2v_vectors_essays))
print(len(train_w2v_vectors_essays[0]))
print('='*50)
```

```
100%|██████████| 6700/6700 [00:13<00:00, 491.91it/s]
```

```
train  vector
6700
250
==================================================
```

Essay Test

In [76]:

```
test_w2v_vectors_essays = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_test["essay"].values): # for each essay in training data
    vector = np.zeros(250) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in top_features:
            i=top_features.index(word)
            vector += svd_2000[i]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_w2v_vectors_essays.append(vector)
print("train  vector")
print(len(test_w2v_vectors_essays))
print(len(test_w2v_vectors_essays[0]))
print('='*50)
```

```
100%|██████████| 3300/3300 [00:06<00:00, 492.99it/s]
```

```
train  vector
3300
250
==================================================
```

Title Train

In [77]:

```
train_w2v_vectors_titles = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_train["project_title"].values): # for each essay in training data
    vector = np.zeros(250) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in top_features:
            i=top_features.index(word)
            vector += svd_2000[i]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_w2v_vectors_titles.append(vector)
print("train  vector")
```

```
print(len(train_w2v_vectors_titles))
print(len(train_w2v_vectors_titles[0]))
print('='*50)
```

```
100%|██████████| 6700/6700 [00:00<00:00, 10512.63it/s]
```

```
train  vector
6700
250
==================================================
```

Title Test

In [78]:

```
test_w2v_vectors_titles = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_test["project_title"].values): # for each essay in training data
    vector = np.zeros(250) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in top_features:
            i=top_features.index(word)
            vector += svd_2000[i]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_w2v_vectors_titles.append(vector)
print("train  vector")
print(len(test_w2v_vectors_titles))
print(len(test_w2v_vectors_titles[0]))
print('='*50)
```

```
100%|██████████| 3300/3300 [00:00<00:00, 10239.65it/s]
```

```
train  vector
3300
250
==================================================
```

GBDT

Creating Data Matrix

In [79]:

```
from scipy.sparse import hstack
X_tr = hstack((train_w2v_vectors_essays,train_w2v_vectors_titles,X_train_state_ohe,
X_train_teacher_ohe, X_train_grade_ohe,X_train_cat_ohe,X_train_sub_ohe,
X_train_price_norm,X_train_project_norm,X_train_title_norm,X_train_essay_norm,essay_sent_pos_train
,essay_sent_neg_train,essay_sent_neu_train,essay_sent_comp_train)).tocsr()
X_te = hstack((test_w2v_vectors_essays,test_w2v_vectors_titles,X_test_state_ohe,
X_test_teacher_ohe, X_test_grade_ohe,X_test_cat_ohe,X_test_sub_ohe,
X_test_price_norm,X_test_project_norm,X_test_title_norm,X_test_essay_norm,essay_sent_pos_test,essa
y_sent_neg_test,essay_sent_neu_test,essay_sent_comp_test)).tocsr()

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(6700, 607) (6700,)
(3300, 607) (3300,)
====================================================================================================
```

Hyperparameter Tuning: Simple for loop (if you are having memory limitations use this)

In [80]:

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

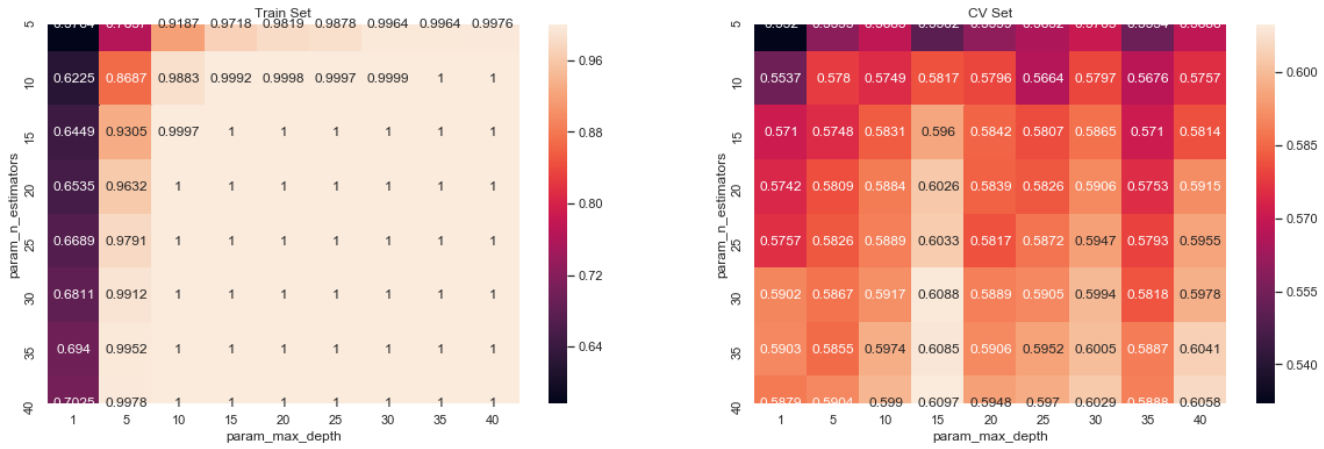
In [83]:

```
%%time
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
ens=XGBClassifier()
params = {'max_depth': [1,5,10,15,20,25,30,35,40], 'n_estimators': [5, 10,15,20,25,30,35,40]}
classifier = GridSearchCV(ens, params, cv=3, scoring='roc_auc',return_train_score=True)
select = classifier.fit(X_tr, y_train)
```

```
CPU times: user 1h 11min 18s, sys: 16.6 s, total: 1h 11min 34s
Wall time: 1h 11min 39s
```

In [84]:

```
import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(classifier.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(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
print(max_scores1.mean_train_score)
print(max_scores1.mean_test_score)
```




```
param_max_depth            1         5         10        15        20  \
param_n_estimators
5                   0.576395  0.763668  0.918744  0.971803  0.981880
10                  0.622545  0.868703  0.988313  0.999185  0.999762
```

```
15                  0.644944  0.930531  0.999684  0.999998  1.000000
20                  0.653515  0.963224  0.999999  1.000000  1.000000
25                  0.668905  0.979086  1.000000  1.000000  1.000000
30                  0.681081  0.991184  1.000000  1.000000  1.000000
35                  0.693993  0.995196  1.000000  1.000000  1.000000
40                  0.702453  0.997816  1.000000  1.000000  1.000000

param_max_depth           25        30        35        40
param_n_estimators
5                   0.987779  0.996435  0.996414  0.997576
10                  0.999665  0.999896  0.999966  0.999980
15                  0.999999  1.000000  1.000000  1.000000
20                  1.000000  1.000000  1.000000  1.000000
25                  1.000000  1.000000  1.000000  1.000000
30                  1.000000  1.000000  1.000000  1.000000
35                  1.000000  1.000000  1.000000  1.000000
40                  1.000000  1.000000  1.000000  1.000000
param_max_depth           1         5         10        15        20  \
param_n_estimators
5                   0.531952  0.559304  0.568497  0.550227  0.559899
10                  0.553697  0.578012  0.574906  0.581657  0.579639
15                  0.571034  0.574802  0.583132  0.595954  0.584235
20                  0.574151  0.580906  0.588420  0.602597  0.583896
25                  0.575677  0.582572  0.588878  0.603264  0.581737
30                  0.590191  0.586725  0.591654  0.608814  0.588938
35                  0.590267  0.585520  0.597438  0.608497  0.590567
40                  0.587896  0.590397  0.599009  0.609677  0.594763

param_max_depth           25        30        35        40
param_n_estimators
5                   0.565231  0.570345  0.553400  0.568628
10                  0.566377  0.579712  0.567567  0.575655
15                  0.580726  0.586456  0.571006  0.581429
20                  0.582556  0.590632  0.575264  0.591457
25                  0.587197  0.594656  0.579279  0.595511
30                  0.590528  0.599365  0.581776  0.597776
35                  0.595223  0.600517  0.588686  0.604077
40                  0.597026  0.602881  0.588756  0.605817
```

In [89]:

```
best_max_depth=1
best_n_estimators=35
```

Train Model

In [90]:

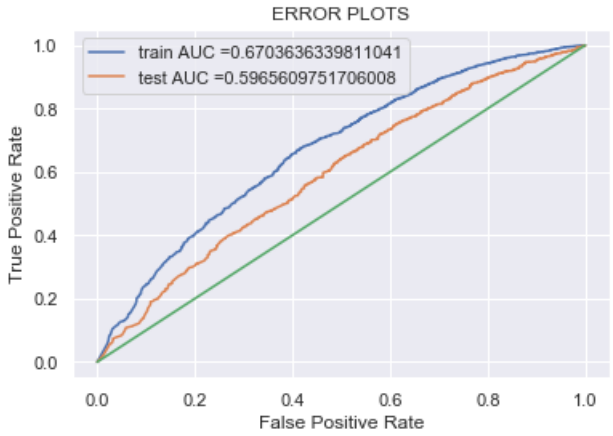
```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
classif = XGBClassifier(max_depth = best_max_depth, n_estimators = best_n_estimators)
classif.fit(X_tr, y_train)
y_train_pred = classif.predict_proba(X_tr)
y_test_pred = classif.predict_proba(X_te)
```

Performance of Model(AUC)

In [91]:

```
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred[:,1])
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred[:,1])
x=[0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.plot(x,x)
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ERROR PLOTS")
plt.grid(True)
plt.show()
```

```
plt.show()
```



Confusion Matrix

In [92]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):
    global predictions1
    t = threshould[np.argmax(fpr*(1-tpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    predictions1=predictions
    return predictions1
```

In [93]:

```
print("Train confusion matrix")
conf_matr_df_train_2=pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred[:,1],tr_thresholds
,train_fpr,train_fpr)),range(2),range(2))
conf_matr_df_train_2
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499997524813742 for threshold 0.836
```

Out[93]:

	0	1
0	503	502
1	1549	4146

In [94]:

```
#conf_matr_df_train_2[1][0]
print("Test confusion matrix")
conf_matr_df_test_2=pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred[:,1],tr_thresholds,tes
t_fpr,test_fpr)),range(2),range(2))
conf_matr_df_test_2
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2499989796959494 for threshold 0.844
```

Out[94]:

	0	1
0	285	210
1	1259	1546

In [ ]: