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

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
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 = 40000)
resource_data = pd.read_csv('resources.csv')
```

In [3]:

```
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 (40000, 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 [4]:

```
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[4]:

	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

In [5]:

```
# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes-for-all-groups-in
-one-step
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()

# join two dataframes in python:
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

In [6]:

```
# #Sampling down the data
project_data = project_data.sample(frac=0.5)
```

1.2 preprocessing of `project_subject_categories`

In [7]:

```
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]))
```

1.3 preprocessing of `project_subject_subcategories`

In [8]:

```
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]))
```

1.3 Text preprocessing

In [9]:

```
# 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 [10]:

```
project_data.head(2)
```

Out[10]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade
14515	65070	p034311	fa7c3df53e6e819835de9bcbbdea4cbc	Ms.	RI	2017-04-19 09:11:00	G
39447	45502	p110053	763d3d978dbb57d9bf9a4cc1be49e554	Mrs.	FL	2016-09-16 15:50:46	G

In [11]:

```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

In [12]:

```
# printing some random reviews
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
print("="*50)
print(project_data['essay'].values[20000])
print("="*50)
```

```
My students are energetic adolescents who have a thirst for learning and an infectious energy. The
y are eager to learn, they are kind and all around curious. \r\n\r\nOur school prides itself on th
e motto, \"all about the kids\" and the students and families know it. Our kids are so important t
```

```
o us and because of that they try do their best every day! \r\n\r\nMany of our students come from
low income families. Some often have difficult home lives. It is my goal as their teacher to make
my classroom a safe space for learning and exploration. \r\n\r\nMy students have been working hard
on collaboration this school year. We've structured our classroom in a way so students can
collaborate in small groups without being confined to a desk.\r\n\r\nOne of the greatest skills fo
r the \"real world\" is the art of collaboration and communication. It is very important that I pr
ovide my 29 students the opportunity to work alongside one another, share ideas and push each othe
r to deepen their learning. By replacing desks with tables, I can provide students the physical sp
ace to create a community of learners. We are fortunate enough to have received a table already th
is year from generous donors and this project will allow every group to have a new table to learn
from. This project is also a huge investment for my classroom as students in future years will be
able to benefit from this space.nannan
==================================================
My third grade classroom is one of fun and rigor. Students will spend this year learning the found
ational and grade level skills to prepare them for Read to Achieve, Maps Performance Exams, and En
d of Grade testing. The students are 3rd graders in an urban North Carolina elementary school.  \r
\nMy school is a Title I Focus school in North Carolina that serves students that come from high p
overty environments. \r\nMany of the students that I serve come from families that are not able to
purchase books for my third graders to read at home. I use small groups to reach my students
through interventions as I would love to help them increase their love for reading! Scholars are e
ager to investigate literature in fun and exciting ways. Scholars enjoy working in small groups to
collaborate and challenge their minds.I am requesting a leveled library with levels H-U, which
includes 5 books on each level fiction and nonfiction. Your contributions to our classroom leveled
library will help my students increase their reading fluency and comprehension. With your
contributions my students will be able to make real world connections, utilize the books to
identify characters, story elements, make inferences, reference the text to answer reading
comprehension questions, and locate text features. As the students are exposed to more reading mat
erial this will increase their background knowledge, build reading comprehension, increase their c
onfidence in reading, and impact their love of reading and learning.\r\n\r\nYour donation to our c
lassroom library will help my third graders develop a love for reading and show them the
importance of becoming lifelong learners.\r\nProviding my class with a leveled library will help t
hem grow as readers and increase their confidence when it comes to reading.nannan
==================================================
I work in a high poverty Title I school.\r\nMy school has the highest number of free and reduced-p
rice lunches in our very large district. My students may come from impoverished homes but they are
like any other kids- VERY motivated by technology and can quickly navigate computer programs and a
pps new to them. The students I work with specifically have significant difficulties communicating
their thoughts in an age appropriate manner because of  difficulties with pronouncing their words
or difficulties utilizing and understanding the complexities of the English language.
Additionally, I have several students who have autism. Many studies have shown the benefits of usi
ng technology to teach children with autism to better communicate.My students need Chromebooks in
the Speech and Language Therapy room to motivate them and to provide them with the latest and most
up to date communication programs and apps .\r\n\r\nMy students' communications skills are are my
priority.\r\nThe Chromebooks will not only excite my students, it will open up a brand new way in
which they can learn! My students as with any students are highly motivated by technology. There a
re always new and innovative speech and language apps that are being produced and the Chromebooks
will give them a front row seat to these apps that are specific to their individual needs.  Many o
f the apps will help my student to learn proper grammar, parts of speech, syntax, semantics,
asking/answering specific types of questions as well as helping them pronounce specific sounds in
isolation/syllables all the way up to conversational speech.nannan
==================================================


---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-12-d9b433748f1b> in <module>
      6 print(project_data['essay'].values[1000])
      7 print("="*50)
----> 8 print(project_data['essay'].values[20000])
      9 print("="*50)

IndexError: index 20000 is out of bounds for axis 0 with size 20000
```

In [13]:

```
# 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 [14]:

```
sent = decontracted(project_data['essay'].values[19000])
print(sent)
print("="*50)
```

```
Dr. Seuss said, "The more that you read, the more things you will know. The more that you learn, t
he more places you will go."I want to help my students go the farthest they can in life. When they
walk in my classroom, they feel welcome, safe and empowered.I teach at a Title 1 school. Ninety-si
x percent of our student population is free of reduced lunch. Almost all of our students are livin
g at or below the poverty line. Many of my students face economic hardships everyday. Some live in
temporary housing. Most of my students come from single parent families. I want to teach my studen
ts to love reading and learning. My students need to become life long learners in order to lesson
and possibly reverse the hardships they endure. Technology is the future and I want my children to
be prepared for it.My vision for my classroom is an environment where students take ownership of t
heir learning. I want to do this through the use of an iPad. This technology in my classroom will
be used to lead small groups, individualized instruction, and instill a love of reading and
math.\r\n\r\nThis donation will improve my students ownership of their learning, technology at the
ir fingertips, and a lifetime of learning.\r\nAll these things will help my students improve both
academically and socially. \r\n\r\nThe case will help protect the iPad.Having an iPad in my classr
oom will motivate students' learning and provide opportunities for them to sharpen their reading,
writing and math skills. It will also build their knowledge of technology as well as their skills
when using it. They will become more cooperative and engaging learners.
==================================================
```

In [15]:

```
# \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks-python/
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

```
Dr. Seuss said, "The more that you read, the more things you will know. The more that you learn, t
he more places you will go."I want to help my students go the farthest they can in life. When they
walk in my classroom, they feel welcome, safe and empowered.I teach at a Title 1 school. Ninety-si
x percent of our student population is free of reduced lunch. Almost all of our students are livin
g at or below the poverty line. Many of my students face economic hardships everyday. Some live in
temporary housing. Most of my students come from single parent families. I want to teach my studen
ts to love reading and learning. My students need to become life long learners in order to lesson
and possibly reverse the hardships they endure. Technology is the future and I want my children to
be prepared for it.My vision for my classroom is an environment where students take ownership of t
heir learning. I want to do this through the use of an iPad. This technology in my classroom will
be used to lead small groups, individualized instruction, and instill a love of reading and math.
This donation will improve my students ownership of their learning, technology at their
fingertips, and a lifetime of learning.  All these things will help my students improve both acade
mically and socially.     The case will help protect the iPad.Having an iPad in my classroom will
motivate students' learning and provide opportunities for them to sharpen their reading, writing a
nd math skills. It will also build their knowledge of technology as well as their skills when
using it. They will become more cooperative and engaging learners.
```

In [16]:

```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
Dr Seuss said The more that you read the more things you will know The more that you learn the mor
e places you will go I want to help my students go the farthest they can in life When they walk in
my classroom they feel welcome safe and empowered I teach at a Title 1 school Ninety six percent o
f our student population is free of reduced lunch Almost all of our students are living at or belo
w the poverty line Many of my students face economic hardships everyday Some live in temporary hou
sing Most of my students come from single parent families I want to teach my students to love read
ing and learning My students need to become life long learners in order to lesson and possibly rev
```

erse the hardships they endure Technology is the future and I want my children to be prepared for it My vision for my classroom is an environment where students take ownership of their learning I want to do this through the use of an iPad This technology in my classroom will be used to lead sm all groups individualized instruction and instill a love of reading and math This donation will im prove my students ownership of their learning technology at their fingertips and a lifetime of lea rning All these things will help my students improve both academically and socially The case will help protect the iPad Having an iPad in my classroom will motivate students learning and provide o pportunities for them to sharpen their reading writing and math skills It will also build their kn owledge of technology as well as their skills when using it They will become more cooperative and engaging learners

In [17]:

```
# 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 [18]:

```
# 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 not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████| 20000/20000
[00:14<00:00, 1369.29it/s]
```

In [19]:

```
# after preprocesing
preprocessed_essays[19000]
```

Out[19]:

'dr seuss said the read things know the learn places go i want help students go farthest life when walk classroom feel welcome safe empowered i teach title 1 school ninety six percent student popul ation free reduced lunch almost students living poverty line many students face economic hardships everyday some live temporary housing most students come single parent families i want teach studen

```
everyday some five temporary housing most students come single parent families i want teach studen
ts love reading learning my students need become life long learners order lesson possibly reverse
hardships endure technology future i want children prepared my vision classroom environment studen
ts take ownership learning i want use ipad this technology classroom used lead small groups indivi
dualized instruction instill love reading math this donation improve students ownership learning t
echnology fingertips lifetime learning all things help students improve academically socially the
case help protect ipad having ipad classroom motivate students learning provide opportunities shar
pen reading writing math skills it also build knowledge technology well skills using they become c
ooperative engaging learners'
```

1.4 Preprocessing of `project_title`

In [20]:

```
# similarly you can preprocess the titles also
# similarly you can preprocess the titles also
# similarly you can preprocess the titles also
# similarly you can preprocess the titles also
from tqdm import tqdm
preprocessed_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['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.lower() not in stopwords)
    preprocessed_titles.append(sent.lower().strip())

preprocessed_titles[1]
```

```
100%|██████████████████████████████████████████████████████████████████████████| 20000/20000
[00:00<00:00, 27768.75it/s]
```

Out[20]:

```
'reaching new heights'
```

In [21]:

```
preprocessed_essays[19000]

project_data['clean_essays'] = preprocessed_essays
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
```

In [22]:

```
project_data['clean_titles'] = preprocessed_titles
```

1.5 Preparing data for models

In [23]:

```
project_data.columns
```

Out[23]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_title',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'price', 'quantity', 'clean_categories', 'clean_subcategories', 'essay',
       'clean_essays', 'clean_titles'],
      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
```

1.5.1 Vectorizing Categorical data

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

In [24]:

```
# Done in TSVD section
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [25]:

```
# Done in TSVD section
```

1.5.2.2 TFIDF vectorizer

In [26]:

```
# Done in TSVD section
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [27]:

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

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

Loading Glove Model
```

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

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

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

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

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

words_courpus = {}
words_glove = set(model.keys())
for i in words:
    if i in words_glove:
        words_courpus[i] = model[i]
print("word 2 vec length", len(words_courpus))


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

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


'''
```

Out[27]:

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

In [28]:

```
# 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())
```

In [29]:

```
# Done in TSVD section
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [30]:

```
# Done in TSVD section
```

1.5.3 Vectorizing Numerical features

In [31]:

```
# Done in TSVD section
```

1.5.4 Merging all the above features

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

In [32]:

```
# Done in TSVD section
```

In [33]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

Computing Sentiment Scores

In [34]:

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

# import nltk
# nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest students w
ith the biggest enthusiasm \
for learning my students learn in many different ways using all of our senses and multiple intelli
gences i use a wide range\
of techniques to help all my students succeed students in my class come from a variety of differen
t backgrounds which makes\
for wonderful sharing of experiences and cultures including native americans our school is a carin
g community of successful \
learners which can be seen through collaborative student project based learning in and out of the
classroom kindergarteners \
in my class love to work with hands on materials and have many different opportunities to practice
a skill before it is\
mastered having the social skills to work cooperatively with friends is a crucial aspect of the ki
ndergarten curriculum\
montana is the perfect place to learn about agriculture and nutrition my students love to role pla
y in our pretend kitchen\
in the early childhood classroom i have had several kids ask me can we try cooking with real food
i will take their idea \
and create common core cooking lessons where we learn important math and writing concepts while co
oking delicious healthy \
food for snack time my students will have a grounded appreciation for the work that went into maki
ng the food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies this project woul
d expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade apple
sauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also create our own cook
books to be printed and \
shared with families students will gain math and literature skills as well as a life long enjoymen
t for healthy cooking \
```

```
nannan'
ss = sid.polarity_scores(for_sentiment)

for k in ss:
    print('{0}: {1}, '.format(k, ss[k]), end='')

# we can use these 4 things as features/attributes (neg, neu, pos, compound)
# neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975
```

```
neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975,
```

In [35]:

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

# import nltk
nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()
sentiment_titles=[]

for sentance in tqdm(project_data['essay'].values):
    ss = sid.polarity_scores(sentance)
    sentiment_titles.append(ss)
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\DELL\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
100%|██████████████████████████████████████████████████████████████████████████| 20000/20000 [01:
07<00:00, 294.34it/s]
```

In [36]:

```
sentiment_neg=[]
sentiment_neu=[]
sentiment_pos=[]
sentiment_compound=[]

for i in sentiment_titles:
    for j,k in i.items():
        if(j=='neg'):
            sentiment_neg.append(k)
        else:
            if(j=='neu'):
                sentiment_neu.append(k)
            else:
                if(j=='pos'):
                    sentiment_pos.append(k)
                else:
                    if(j=='compound'):
                        sentiment_compound.append(k)
```

In [37]:

```
project_data['sentiment_neg'] = sentiment_neg
project_data['sentiment_neu'] = sentiment_neu
project_data['sentiment_pos'] = sentiment_pos
project_data['sentiment_compound'] = sentiment_compound
```

In [38]:

```
#https://stackoverflow.com/questions/49984905/count-number-of-words-per-row
project_data['words_title'] = project_data['project_title'].str.split().str.len()
```

In [39]:

```
#https://stackoverflow.com/questions/49984905/count-number-of-words-per-row
project_data['words_essay'] = project_data['essay'].str.split().str.len()
```

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**

In [40]:

```
import sys
import math

import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score

# you might need to install this one
import xgboost as xgb

class XGBoostClassifier():
    def __init__(self, num_boost_round=10, **params):
        self.clf = None
        self.num_boost_round = num_boost_round
        self.params = params
        self.params.update({'objective': 'multi:softprob'})

    def fit(self, X, y, num_boost_round=None):
        num_boost_round = num_boost_round or self.num_boost_round
        self.label2num = {label: i for i, label in enumerate(sorted(set(y)))}
        dtrain = xgb.DMatrix(X, label=[self.label2num[label] for label in y])
        self.clf = xgb.train(params=self.params, dtrain=dtrain, num_boost_round=num_boost_round, ve
rbose_eval=1)

    def predict(self, X):
        num2label = {i: label for label, i in self.label2num.items()}
        Y = self.predict_proba(X)
        y = np.argmax(Y, axis=1)
        return np.array([num2label[i] for i in y])

    def predict_proba(self, X):
        dtest = xgb.DMatrix(X)
```

```
        return self.clf.predict(dtest)

    def score(self, X, y):
        Y = self.predict_proba(X)[:,1]
        return roc_auc_score(y, Y)

    def get_params(self, deep=True):
        return self.params

    def set_params(self, **params):
        if 'num_boost_round' in params:
            self.num_boost_round = params.pop('num_boost_round')
        if 'objective' in params:
            del params['objective']
        self.params.update(params)
        return self


clf = XGBoostClassifier(eval_metric = 'auc', num_class = 2, nthread = 4,)
#################################################################
#                  Change from here                             #
#################################################################
parameters = {
    'num_boost_round': [100, 250, 500],
    'eta': [0.05, 0.1, 0.3],
    'max_depth': [6, 9, 12],
    'subsample': [0.9, 1.0],
    'colsample_bytree': [0.9, 1.0],
}

clf = GridSearchCV(clf, parameters)
X = np.array([[1,2], [3,4], [2,1], [4,3], [1,0], [4,5]])
Y = np.array([0, 1, 0, 1, 0, 1])
clf.fit(X, Y)

#print(clf.cv_results_)
score = max(clf.cv_results_, key=lambda x: x[1])
print('score:', score)
```

```
score: std_fit_time
```

2. TruncatedSVD

2.1 Selecting top 2000 words from `essay` and `project_title`

In [41]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label

project_data["tiles_essays"] = project_data["clean_titles"].map(str) +\
                        project_data["clean_essays"].map(str)
```

In [42]:

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

In [43]:

```
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)
```

```
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

In [44]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vect = TfidfVectorizer(ngram_range = (1,1) , max_features = 2000)
tfidf_train = tfidf_vect.fit_transform (X_train['tiles_essays'])
```

In [45]:

```
top_2000 = tfidf_vect.get_feature_names()
```

2.2 Computing Co-occurance matrix

In [46]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label

from tqdm import tqdm
n_neighbor = 5
occ_matrix_2000 = np.zeros((2000,2000))
for row in tqdm(X_train["tiles_essays"].values):
    words_in_row = row.split()
    for index,word in enumerate(words_in_row):
        if word in top_2000:
            for j in range(max(index-n_neighbor,0),min(index+n_neighbor,len(words_in_row)-1) + 1):
                if words_in_row[j] in top_2000:
                    occ_matrix_2000[top_2000.index(word),top_2000.index(words_in_row[j])] += 1
                else:
                    pass
        else:
            pass
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 8978/8978 [13
:04<00:00, 11.45it/s]
```

2.3 Applying TruncatedSVD and Calculating Vectors for `essay` and `project_title`

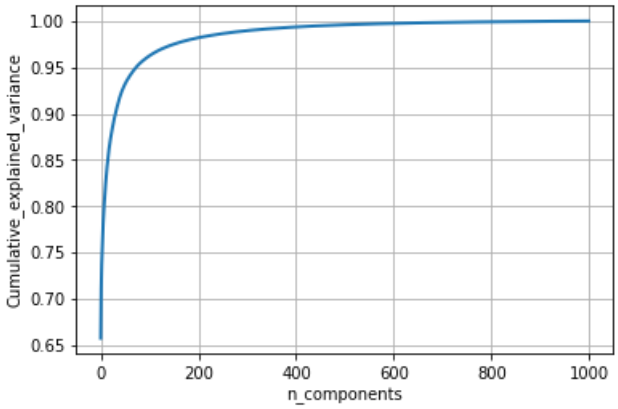
In [47]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

In [48]:

```
#https://chrisalbon.com/machine_learning/feature_engineering/select_best_number_of_components_in_t
```

```
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import StandardScaler
svd = TruncatedSVD(n_components = 1000)
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 [49]:

```
svd = TruncatedSVD(n_components = 150)
svd_2000 = svd.fit_transform(occ_matrix_2000)
```

In [50]:

```
my_dict={}
my_dict = dict.fromkeys(top_2000 , 1)
```

In [51]:

```
svd_2000_list=svd_2000.tolist()
```

In [52]:

```
my_dict = { i : svd_2000_list[i] for i in range(0, len(svd_2000_list) ) }
```

In [53]:

```
first2pairs = {k: my_dict[k] for k in list(my_dict)[:1]}
```

In [54]:

```
glove_words =  set(my_dict.keys())
```

In [55]:

```
train_w2v_vectors_essays = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_train["tiles_essays"].values): # for each essay in training data
    vector = np.zeros(150) # 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 glove_words:
            vector += my_dict[word]
            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%|████████████████████████████████████████████████████████████████| 8978/8978
[00:00<00:00, 24227.84it/s]
```

```
train  vector
8978
150
==================================================
```

In [56]:

```
test_w2v_vectors_essays = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_test["tiles_essays"].values): # for each essay in training data
    vector = np.zeros(150) # 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 glove_words:
            vector += my_dict[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_w2v_vectors_essays.append(vector)
print("test  vector")
print(len(test_w2v_vectors_essays))
print(len(test_w2v_vectors_essays[0]))
print('='*50)
```

```
100%|████████████████████████████████████████████████████████████████| 6600/6600
[00:00<00:00, 22105.09it/s]
```

```
test  vector
6600
150
==================================================
```

In [57]:

```
cv_w2v_vectors_essays = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_cv["tiles_essays"].values): # for each essay in training data
    vector = np.zeros(150) # 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 glove_words:
            vector += my_dict[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_w2v_vectors_essays.append(vector)
print("cv  vector")
print(len(cv_w2v_vectors_essays))
print(len(cv_w2v_vectors_essays[0]))
print('='*50)
```

```
100%|████████████████████████████████████████████████████████████████| 4422/4422
[00:00<00:00, 23281.97it/s]
```

```
cv  vector
4422
150
==================================================
```

In [58]:

```
# Changing list to numpy arrays
train_w2v_vectors_essays = np.array(train_w2v_vectors_essays)
test_w2v_vectors_essays = np.array(test_w2v_vectors_essays)
cv_w2v_vectors_essays = np.array(cv_w2v_vectors_essays)
```

In [59]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values)


X_train_clean_cat_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_cv_clean_cat_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_clean_cat_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_clean_cat_ohe.shape, y_train.shape)
print(X_cv_clean_cat_ohe.shape, y_cv.shape)
print(X_test_clean_cat_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(8978, 9) (8978,)
(4422, 9) (4422,)
(6600, 9) (6600,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
====================================================================================================
```

In [60]:

```
#Encoding project_subject_subcategories
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values)


X_train_clean_subcat_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_clean_subcat_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_clean_subcat_ohe = vectorizer.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_clean_subcat_ohe.shape, y_train.shape)
print(X_cv_clean_subcat_ohe.shape, y_cv.shape)
print(X_test_clean_subcat_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(8978, 30) (8978,)
(4422, 30) (4422,)
(6600, 30) (6600,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
====================================================================================================
```

In [61]:

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

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
grade_dict = dict(my_counter)
sorted_grade_dict = dict(sorted(grade_dict.items(), key=lambda kv: kv[1]))
```

In [62]:

```
#one hot encoding the catogorical features: project_grade_category
vectorizer = CountVectorizer(vocabulary=list(sorted_grade_dict.keys()), lowercase=False, binary=Tr
ue)
vectorizer.fit(X_train['project_grade_category'].values)

X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vectorizer.transform(X_cv['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_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(8978, 5) (8978,)
(4422, 5) (4422,)
(6600, 5) (6600,)
['9-12', '6-8', '3-5', 'PreK-2', 'Grades']
====================================================================================================
```

In [63]:

```
#Encoding school state
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values)

X_train_state_ohe = vectorizer.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer.transform(X_cv['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_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(8978, 51) (8978,)
(4422, 51) (4422,)
(6600, 51) (6600,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm',
'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv
', 'wy']
====================================================================================================
```

In [64]:

```
#one hot encoding for teacher_prefix
from collections import Counter
my_counter = Counter()
for word in project_data['teacher_prefix'].values:
    my_counter.update(str(word).split())


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

vectorizer = CountVectorizer(vocabulary=list((sorted_teacher_dict.keys())), lowercase=False,
binary=True)
vectorizer.fit(project_data['teacher_prefix'].values.astype('U'))


X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values.astype('U'))
```

```
X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values.astype('U'))
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values.astype('U'))

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

```
{'Dr.': 1, 'nan': 2, 'Teacher': 441, 'Mr.': 1914, 'Ms.': 7259, 'Mrs.': 10383}
After vectorizations
(8978, 6) (8978,)
(4422, 6) (4422,)
(6600, 6) (6600,)
['Dr.', 'nan', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
====================================================================================================
```

In [65]:

```
#Normalising the numerical feature
from sklearn.preprocessing import StandardScaler
standard_vec = StandardScaler(with_mean = False)
# 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.
standard_vec.fit(X_train['price'].values.reshape(-1,1))

X_train_price_std = standard_vec.transform(X_train['price'].values.reshape(-1,1))
X_cv_price_std = standard_vec.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_std = standard_vec.transform(X_test['price'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_price_std.shape, y_train.shape)
print(X_cv_price_std.shape, y_cv.shape)
print(X_test_price_std.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(8978, 1) (8978,)
(4422, 1) (4422,)
(6600, 1) (6600,)
====================================================================================================
```

In [66]:

```
#Normalising the numerical feature-no of words in essay
from sklearn.preprocessing import StandardScaler
standard_vec = StandardScaler(with_mean = False)
# 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.
standard_vec.fit(X_train['words_essay'].values.reshape(-1,1))

X_train_words_essay_std = standard_vec.transform(X_train['words_essay'].values.reshape(-1,1))
X_cv_words_essay_std = standard_vec.transform(X_cv['words_essay'].values.reshape(-1,1))
X_test_words_essay_std = standard_vec.transform(X_test['words_essay'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_words_essay_std.shape, y_train.shape)
print(X_cv_words_essay_std.shape, y_cv.shape)
print(X_test_words_essay_std.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
```

```
(8978, 1) (8978,)
(4422, 1) (4422,)
(6600, 1) (6600,)
====================================================================================================
```

In [67]:

```
#Normalising the numerical feature-no of words in titles
from sklearn.preprocessing import StandardScaler
standard_vec = StandardScaler(with_mean = False)
# 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.
standard_vec.fit(X_train['words_title'].values.reshape(-1,1))

X_train_words_title_std = standard_vec.transform(X_train['words_title'].values.reshape(-1,1))
X_cv_words_title_std = standard_vec.transform(X_cv['words_title'].values.reshape(-1,1))
X_test_words_title_std = standard_vec.transform(X_test['words_title'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_words_title_std.shape, y_train.shape)
print(X_cv_words_title_std.shape, y_cv.shape)
print(X_test_words_title_std.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(8978, 1) (8978,)
(4422, 1) (4422,)
(6600, 1) (6600,)
====================================================================================================
```

In [68]:

```
#Normalising the numerical feature-sentiment score of neg words
from sklearn.preprocessing import StandardScaler
standard_vec = StandardScaler(with_mean = False)
# 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.
standard_vec.fit(X_train['sentiment_neg'].values.reshape(-1,1))

X_train_sentiment_neg_std = standard_vec.transform(X_train['sentiment_neg'].values.reshape(-1,1))
X_cv_sentiment_neg_std = standard_vec.transform(X_cv['sentiment_neg'].values.reshape(-1,1))
X_test_sentiment_neg_std = standard_vec.transform(X_test['sentiment_neg'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_sentiment_neg_std.shape, y_train.shape)
print(X_cv_sentiment_neg_std.shape, y_cv.shape)
print(X_test_sentiment_neg_std.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(8978, 1) (8978,)
(4422, 1) (4422,)
(6600, 1) (6600,)
====================================================================================================
```

In [69]:

```
#Normalising the numerical feature-sentiment score of neutral words
from sklearn.preprocessing import StandardScaler
standard_vec = StandardScaler(with_mean = False)
# 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.
standard_vec.fit(X_train['sentiment_neu'].values.reshape(-1,1))

X_train_sentiment_neu_std = standard_vec.transform(X_train['sentiment_neu'].values.reshape(-1,1))
X_cv_sentiment_neu_std = standard_vec.transform(X_cv['sentiment_neu'].values.reshape(-1,1))
X_test_sentiment_neu_std = standard_vec.transform(X_test['sentiment_neu'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_sentiment_neu_std.shape, y_train.shape)
print(X_cv_sentiment_neu_std.shape, y_cv.shape)
print(X_test_sentiment_neu_std.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(8978, 1) (8978,)
(4422, 1) (4422,)
(6600, 1) (6600,)
====================================================================================================
```

In [70]:

```
#Normalising the numerical feature-sentiment score of positive words
from sklearn.preprocessing import StandardScaler
standard_vec = StandardScaler(with_mean = False)
# 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.
standard_vec.fit(X_train['sentiment_pos'].values.reshape(-1,1))

X_train_sentiment_pos_std = standard_vec.transform(X_train['sentiment_pos'].values.reshape(-1,1))
X_cv_sentiment_pos_std = standard_vec.transform(X_cv['sentiment_pos'].values.reshape(-1,1))
X_test_sentiment_pos_std = standard_vec.transform(X_test['sentiment_pos'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_sentiment_pos_std.shape, y_train.shape)
print(X_cv_sentiment_pos_std.shape, y_cv.shape)
print(X_test_sentiment_pos_std.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(8978, 1) (8978,)
(4422, 1) (4422,)
(6600, 1) (6600,)
====================================================================================================
```

In [71]:

```
#Normalising the numerical feature-sentiment score of compound words
from sklearn.preprocessing import StandardScaler
standard_vec = StandardScaler(with_mean = False)
# 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.
standard_vec.fit(X_train['sentiment_compound'].values.reshape(-1,1))

X_train_sentiment_compound_std =
standard_vec.transform(X_train['sentiment_compound'].values.reshape(-1,1))
X_cv_sentiment_compound_std = standard_vec.transform(X_cv['sentiment_compound'].values.reshape(-1,
1))
X_test_sentiment_compound_std = standard_vec.transform(X_test['sentiment_compound'].values.reshape
(-1,1))
```

```
print("After vectorizations")
print(X_train_sentiment_compound_std.shape, y_train.shape)
print(X_cv_sentiment_compound_std.shape, y_cv.shape)
print(X_test_sentiment_compound_std.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(8978, 1) (8978,)
(4422, 1) (4422,)
(6600, 1) (6600,)
====================================================================================================
```

In [72]:

```
#Normalising the numerical features: teacher_number_of_previously _posted_projects
from sklearn.preprocessing import StandardScaler
standard_vec = StandardScaler(with_mean = False)
# 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.
standard_vec.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

X_train_projects_std =
standard_vec.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1
))
X_cv_projects_std = standard_vec.transform(X_cv['teacher_number_of_previously_posted_projects'].va
lues.reshape(-1,1))
X_test_projects_std = standard_vec.transform(X_test['teacher_number_of_previously_posted_projects'
].values.reshape(-1,1))

print("After vectorizations")
print(X_train_projects_std.shape, y_train.shape)
print(X_cv_projects_std.shape, y_cv.shape)
print(X_test_projects_std.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(8978, 1) (8978,)
(4422, 1) (4422,)
(6600, 1) (6600,)
====================================================================================================
```

In [73]:

```
#Normalising numerical features: quantity
from sklearn.preprocessing import StandardScaler
standard_vec = StandardScaler(with_mean = False)
# 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.
standard_vec.fit(X_train['quantity'].values.reshape(-1,1))

X_train_qty_std = standard_vec.transform(X_train['quantity'].values.reshape(-1,1))
X_cv_qty_std = standard_vec.transform(X_cv['quantity'].values.reshape(-1,1))
X_test_qty_std = standard_vec.transform(X_test['quantity'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_qty_std.shape, y_train.shape)
print(X_cv_qty_std.shape, y_cv.shape)
print(X_test_qty_std.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(8978, 1) (8978,)
(4422, 1) (4422,)
(6600, 1) (6600,)
```

```
==========================================================================================================
```

2.4 Merge the features from step 3 and step 4

In [74]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

In [75]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((train_w2v_vectors_essays,X_train_sentiment_compound_std,X_train_sentiment_pos_std,X
_train_sentiment_neu_std,X_train_sentiment_neg_std,X_train_words_title_std,X_train_words_essay_std
,X_train_clean_cat_ohe,X_train_clean_subcat_ohe, X_train_state_ohe, X_train_teacher_ohe,
X_train_grade_ohe, X_train_price_std,X_train_projects_std,X_train_qty_std)).tocsr()
X_cr =
hstack((cv_w2v_vectors_essays,X_cv_sentiment_compound_std,X_cv_sentiment_pos_std,X_cv_sentiment_neu
_std,X_cv_sentiment_neg_std,X_cv_words_title_std,X_cv_words_essay_std,X_cv_clean_cat_ohe,X_cv_clean
_subcat_ohe, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_price_std,X_cv_projects_std,X_c
v_qty_std)).tocsr()
X_te =
hstack((test_w2v_vectors_essays,X_test_sentiment_compound_std,X_test_sentiment_pos_std,X_test_senti
ment_neu_std,X_test_sentiment_neg_std,X_test_words_title_std,X_test_words_essay_std,X_test_clean_ca
t_ohe,X_test_clean_subcat_ohe, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe,
X_test_price_std,X_test_projects_std,X_test_qty_std)).tocsr()

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

```
Final Data matrix
(8978, 260) (8978,)
(4422, 260) (4422,)
(6600, 260) (6600,)
==========================================================================================================
```

2.5 Apply XGBoost on the Final Features from the above section

https://xgboost.readthedocs.io/en/latest/python/python_intro.html

In [76]:

```
# No need to split the data into train and test(cv)
# use the Dmatrix and apply xgboost on the whole data
# please check the Quora case study notebook as reference

# please write all the code with proper documentation, and proper titles for each subsection
```

```
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label

def plot_roc(classifier, X_train, y_train, X_test, y_test):
    from sklearn.metrics import roc_curve, auc
    fpr = dict()
    tpr = dict()
    roc_auc = dict()

    '''TEST DATA ROC CURVE'''
    #Use probability scores to compute the ROC Curve
    class_probabilities = classifier.predict_proba(X_test)
    y_probs = class_probabilities[:,1]
    fpr["Test"], tpr["Test"], threshold = roc_curve(y_test, y_probs)
    roc_auc["Test"] = auc(fpr["Test"], tpr["Test"])

    '''TRAIN DATA ROC CURVE'''
    #Use probability scores to compute the ROC Curve
    class_probabilities = classifier.predict_proba(X_train)
    y_probs = class_probabilities[:,1]
    fpr["Train"], tpr["Train"], threshold = roc_curve(y_train, y_probs)
    roc_auc["Train"] = auc(fpr["Train"], tpr["Train"])

    plt.figure(figsize=(15,10))
    linewidth = 2
    plt.plot(fpr["Test"], tpr["Test"], color='green', lw=linewidth, label='ROC curve Test Data (are
a = %0.2f)' % roc_auc["Test"])
    plt.plot(fpr["Train"], tpr["Train"], color='red', lw=linewidth, label='ROC curve Train Data (ar
ea = %0.2f)' % roc_auc["Train"])
    plt.plot([0, 1], [0, 1], color='navy', lw=linewidth, linestyle='--', label='Baseline ROC curve
(area = 0.5)')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.legend(loc="lower right")
    plt.show()
```

In [77]:

```
def top_features(trained_clf, vect_object):
    print("Word Cloud Showing the top 50 most important features.")
    '''Get the most important features for the given input vector.'''
    features = trained_clf.feature_importances_
    top_features_index = (-features).argsort()              #Note : Putting a - sign indicates the
indexes will be sorted in descending order.
    top_feat = np.take(vect_object.get_feature_names(), top_features_index[:50])
    corpus = " "
    for i in top_feat:
        corpus = corpus + " " + i
    wordcloud = WordCloud(background_color='black',
                          width=800,
                          height=450)
    wordcloud.generate(corpus)
    fig = plt.figure(1, figsize=(15, 15), facecolor='k')
    plt.axis('off')
    plt.imshow(wordcloud, aspect='equal')
    plt.tight_layout(pad=0)
    plt.show()
```

In [78]:

```
#Fit this model with the best value of hyperparameter obtained.
def performance(best_clf, vectorizationType, X_train_vec, y_train, X_test_vec, y_test, X_calib_vec,
y_calib, max_depth, n_estimators): #optimal_hp should have 2 hyperparameters.
```

```
    print("============================================================")

#This function is used to visualize the results in a 2D plot.


def plot_auc_2d(param1_val, param2_val, mean_auc_train, mean_auc_cv, results, param_names):

    #Sort the AUC scores according to the parameter n_estimators and plot AUC vs n_estimators
    param1_val = [results['params'][i][param_names[0]] for i in range(0,len(results['params']))] #n
_estimators
    mean_auc_train_sorted = [i for _,i in sorted(zip(param1_val,mean_auc_train))]
    mean_auc_cv_sorted = [i for _,i in sorted(zip(param1_val,mean_auc_cv))]
    param1_val = sorted(param1_val)

    labels=["Train AUC","Test AUC"]
    y_axes = [mean_auc_train_sorted, mean_auc_cv_sorted]

    color_map = ['red','green']
    plt.figure(figsize=(10,6))
    for index in range(0,len(labels)):
        plt.plot(param1_val, y_axes[index], color=color_map[index], label=labels[index])
    plt.xlabel('Number of estimators (n_estimators)')
    plt.ylabel('AUC Scores')
    plt.title("Comparison between Train and Cross Validate AUC scores with n_estimators.")
    plt.legend()
    plt.show()

    #Sort the AUC scores according to the parameter max_depth and plot AUC vs max_depth
    param2_val = [results['params'][i][param_names[1]] for i in range(0,len(results['params']))] #m
ax_depth
    mean_auc_train_sorted = [i for _,i in sorted(zip(param2_val,mean_auc_train))]
    mean_auc_cv_sorted = [i for _,i in sorted(zip(param2_val,mean_auc_cv))]
    param2_val = sorted(param2_val)

    labels=["Train AUC","Test AUC"]
    y_axes = [mean_auc_train_sorted, mean_auc_cv_sorted]

    color_map = ['red','green']
    plt.figure(figsize=(10,6))
    for index in range(0,len(labels)):
        plt.plot(param2_val, y_axes[index], color=color_map[index], label=labels[index])
    plt.xlabel('Max Depths (max_depth)')
    plt.ylabel('AUC Scores')
    plt.title("Comparison between Train and Cross Validate AUC scores with max_depth.")
    plt.legend()
    plt.show()


#Utility function to report best scores, https://scikit-
learn.org/stable/auto_examples/model_selection/plot_randomized_search.html
def report(results, n_top=3):
    for i in range(1, n_top + 1):
        candidates = np.flatnonzero(results['rank_test_score'] == i)
        for candidate in candidates:
            print("Model with rank: {0}".format(i))
            print("Mean validation score: {0:.3f} (std: {1:.3f})".format(
                  results['mean_test_score'][candidate],
                  results['std_test_score'][candidate]))
            print("Parameters: {0}".format(results['params'][candidate]))
            print("")

#This function is used to obtain the grid values and call other functions to visualize the result
of Grid Search.
def plot_results(rsearch_cv):
    '''This function is used to plot the curve for mean squared errors vs alpha values and obtain
the optimal
    value of the hyperparameters'''

    results = rsearch_cv.cv_results_

    #Get the Hyperparameter names in a dictionary
    param_names = []
    for i in results["params"][0].keys():
        param_names.append(str(i))

    param1_val = [results['params'][i][param_names[0]] for i in range(0,len(results['params']))] #m
ax_depth
```

```
    param2_val = [results['params'][i][param_names[1]] for i in range(0,len(results['params']))] #m
in_sample_split

    mean_auc_train =  results['mean_train_score']
    mean_auc_cv =  results['mean_test_score']

    #Get the index position corresponding to the highest AUC score and use it to find the values o
f best hyperparameters
    index = list(mean_auc_cv).index(max(list(mean_auc_cv)))
    best_param1 = param1_val[index] #n_estimators
    best_param2 = param2_val[index] #max_depth

    print("\nThe best value of hyperparameters are: \n{} = {} \n{} = {}".format(param_names[0],best
_param1,param_names[1],best_param2))

    #Compare Train and Test AUC in a 2D graph to determine overfitting and underfitting
    plot_auc_2d(param1_val, param2_val, mean_auc_train, mean_auc_cv, results, param_names)
```

In [79]:

```
from xgboost import XGBClassifier
def get_XGBDT_RandSearchCV(vectorizationType, X_train, y_train, X_test, y_test):
    '''This function will determine the best hyperparameters using TimeSeriesSplit CV and
RandomSearchCV, using 10 fold cross validation. '''
    from sklearn.model_selection import TimeSeriesSplit
    tuned_parameters = {'max_depth': [2,4,6,8,10],
                        'n_estimators' : [50,100,150,200,250]}
    base_estimator = XGBClassifier(learning_rate=0.01, objective='binary:logistic', random_state=0)


    #Run randomized search
    n_iter_search = 30
    rsearch_cv = RandomizedSearchCV(base_estimator,
                                    random_state=0,
                                    param_distributions=tuned_parameters,
                                    n_iter=n_iter_search,
                                    scoring='roc_auc',
                                    n_jobs=-1)
    rsearch_cv.fit(X_train,y_train)
    print("Best estimator obtained from CV data: \n", rsearch_cv.best_estimator_)
    print("Best Score : ", rsearch_cv.best_score_)
    plot_results(rsearch_cv)
    return (rsearch_cv)
```

In [80]:

```
from datetime import datetime as dt
def XGBOOST_CLF(X_train_vectors, y_train, X_test_vectors, y_test, X_calib_vectors, y_calib, vectori
zationType):
    '''This  function will determine the best estimators for each model and use them to call sever
al other functions
    which trains the model and measure the performance of the model and plot the final results etc
.'''

    print("\nUSING RandomSearchCV TO DETERMINE THE HYPERPARAMETERS. ")
    print("*****************************************************")

    st_t = dt.now()
    rsearch_cv = get_XGBDT_RandSearchCV(vectorizationType, X_train_vectors, y_train, X_test_vectors
, y_test)
    print("Time taken to complete Hyperparameter Search : ",dt.now()-st_t)
    print("\nScore Reports with Ranks and STD\n")
    report(rsearch_cv.cv_results_)
    best_estimator = rsearch_cv.best_estimator_
    max_depth = rsearch_cv.best_estimator_.max_depth
    n_estimators = rsearch_cv.best_estimator_.n_estimators
    trained_clf = performance(best_estimator, vectorizationType, X_train_vectors, y_train, X_test_v
ectors, y_test, X_calib_vectors, y_calib, max_depth, n_estimators)
    return (trained_clf)
```

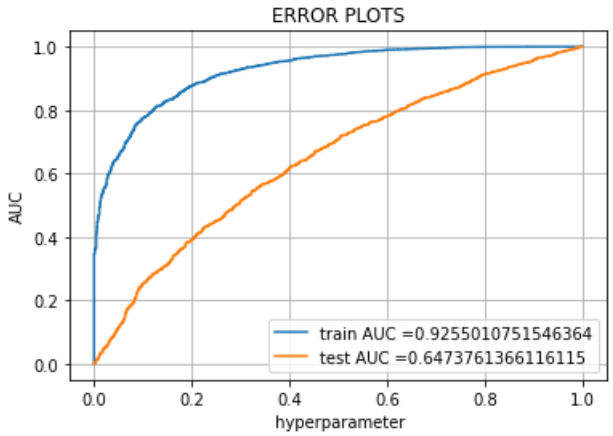
In [81]:

```
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 [91]:

```
%%time
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
from sklearn.linear_model import LogisticRegression
from sklearn.calibration import CalibratedClassifierCV
from sklearn import tree

dtc=clf = XGBClassifier(n_estimators=200,max_depth=5)
dtc.fit(X_tr, y_train)


y_train_pred = batch_predict(dtc, X_tr)
y_test_pred = batch_predict(dtc, X_te)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

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.legend()
plt.xlabel(" hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

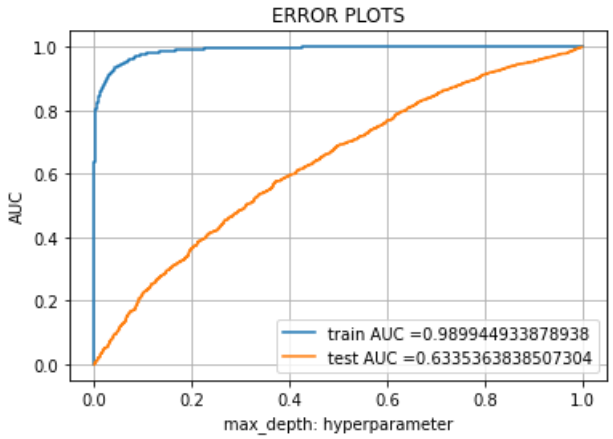


Wall time: 2.22 s

In [83]:

```
%%time
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
from sklearn.linear_model import LogisticRegression
from sklearn.calibration import CalibratedClassifierCV
from sklearn import tree

dtc=clf = XGBClassifier(n_estimators=500,max_depth=5)
```

```
dtc.fit(X_tr, y_train)


y_train_pred = batch_predict(dtc, X_tr)
y_test_pred = batch_predict(dtc, X_te)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

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.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```




```
Wall time: 5.41 s
```

In [84]:

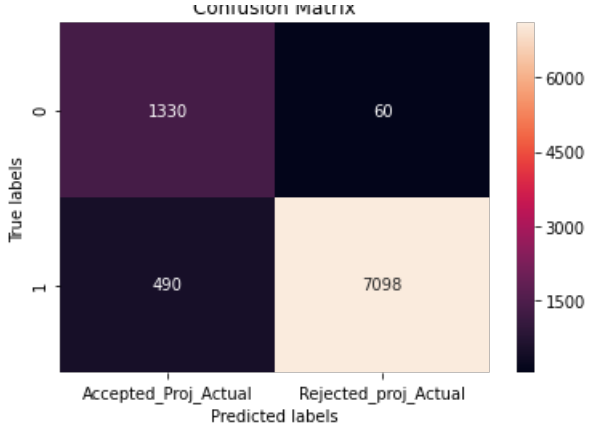
```
import seaborn as sns
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(tpr*(1-fpr))]

    # (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)),np.argmax(tpr*(1-fpr)), "for thresho
ld",threshould[np.argmax(tpr*(1-fpr))], np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [85]:

```
import seaborn as sns
import matplotlib.pyplot as plt
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)),
annot=True, fmt="d", ax = ax)
# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['Accepted_Proj_Actual', 'Rejected_proj_Actual']); ax.yaxis.set_ticklabels
(['Accepted_Proj_Pred', 'Rejected_proj_pred']);
```

```
the maximum value of tpr*(1-fpr) 0.895046324563965 152 for threshold 0.7835566 0.784
```

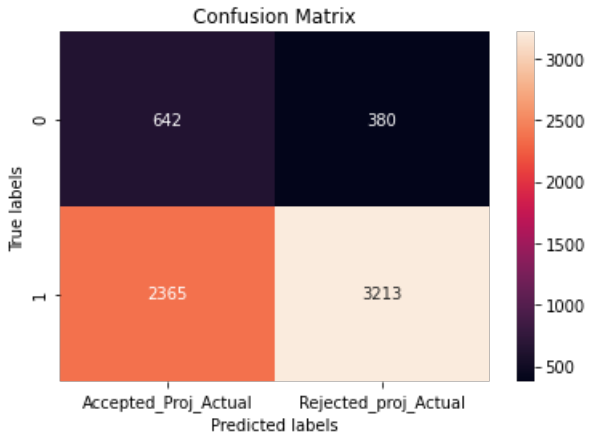
Confusion Matrix



In [86]:

```
import seaborn as sns
import matplotlib.pyplot as plt
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_tpr)), anno
t=True, fmt="d", ax = ax)
# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['Accepted_Proj_Actual', 'Rejected_proj_Actual']); ax.yaxis.set_ticklabels
(['Accepted_Proj_Pred', 'Rejected_proj_pred']);
```

```
the maximum value of tpr*(1-fpr) 0.3618398109991797 687 for threshold 0.89355725 0.894
```



In [101]:

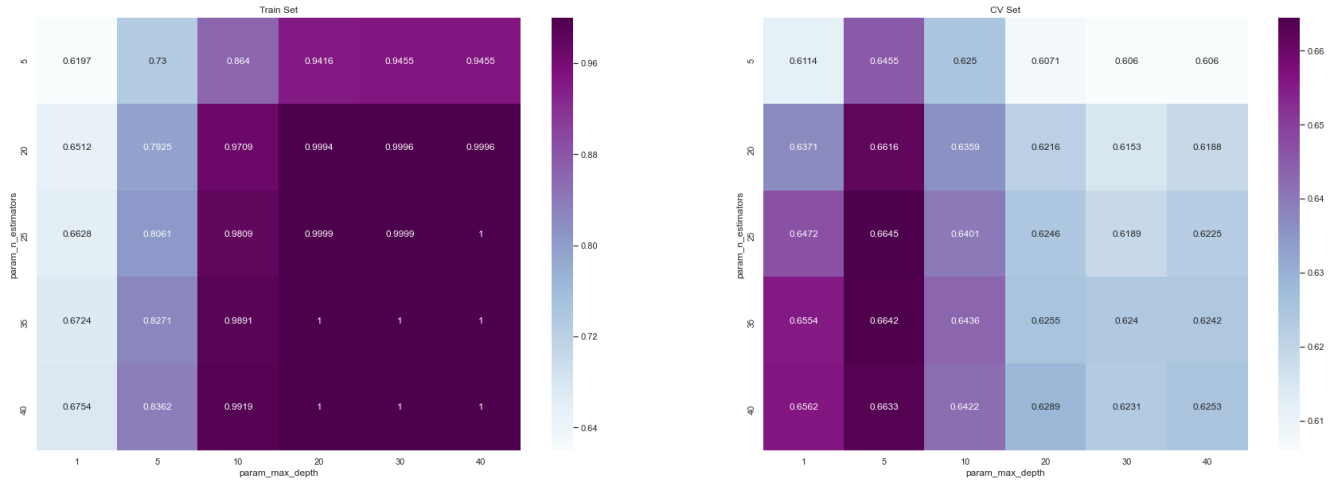
```
%%time
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
import xgboost
import seaborn as sb


XG = xgboost.XGBClassifier(scale_pos_weight=1, n_jobs = -1)
parameters = {'max_depth':[1,5,10,20,30,40] , 'n_estimators':[5,20,25,35,40]}
XGB = GridSearchCV(XG, parameters, cv=3, scoring='roc_auc',return_train_score=True)
XGB.fit(X_tr, y_train)
print('Best estimator', XGB.best_estimator_)
print('Best score', XGB.best_score_)
```

```
Best estimator XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=5,
              min_child_weight=1, missing=None, n_estimators=25, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
Best score 0.6644550182388971
```

```
Wall time: 54.4 s
```

In [102]:

```
XGB = pd.DataFrame.from_dict(XGB.cv_results_)
max_scores_avg = XGB.groupby(['param_n_estimators',
                              'param_max_depth']).max()
max_scores_avg = max_scores_avg.unstack()[['mean_test_score', 'mean_train_score']]
#https://towardsdatascience.com/using-3d-visualizations-to-tune-hyperparameters-of-ml-models-with-python-ba2885eab2e9
import seaborn as sns; sns.set()

fig, ax = plt.subplots(1,2, figsize=(30,10))
sns.heatmap(max_scores_avg.mean_train_score, annot = True, fmt='.4g',cmap= "BuPu", ax=ax[0])
sns.heatmap(max_scores_avg.mean_test_score, annot = True, fmt='.4g',cmap="BuPu", ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
print(max_scores_avg.mean_train_score)
print(max_scores_avg.mean_test_score)
```



param_max_depth	1	5	10	20	30	40
param_n_estimators						
5	0.619735	0.729986	0.863985	0.941565	0.945457	0.945457
20	0.651173	0.792469	0.970920	0.999361	0.999577	0.999610
25	0.662781	0.806150	0.980909	0.999890	0.999948	0.999956
35	0.672402	0.827136	0.989094	0.999995	0.999999	1.000000
40	0.675388	0.836197	0.991872	0.999999	1.000000	1.000000

param_max_depth	1	5	10	20	30	40
param_n_estimators						
5	0.611367	0.645509	0.624984	0.607130	0.605968	0.605968
20	0.637104	0.661619	0.635938	0.621588	0.615251	0.618771
25	0.647229	0.664455	0.640100	0.624632	0.618876	0.622451
35	0.655396	0.664210	0.643596	0.625490	0.623986	0.624240
40	0.656168	0.663265	0.642201	0.628851	0.623112	0.625333

3. Conclusion

In [84]:

```
# Please write down few lines about what you observed from this assignment.
```

In [85]:

```
from prettytable import PrettyTable

x_pretty_table = PrettyTable()
x_pretty_table.field_names = ["Model Type","max_depth","n_estimators","Train-AUC","Test-AUC"]

x_pretty_table.add_row(["XGB",200,5,0.84,0.69])
x_pretty_table.add_row([ "XGB",500,5,0.93,0.67])
print(x_pretty_table)
```

```
+------------+-----------+--------------+-----------+----------+
| Model Type | max_depth | n_estimators | Train-AUC | Test-AUC |
+------------+-----------+--------------+-----------+----------+
|    XGB     |    200    |      5       |    0.84   |   0.69   |
|    XGB     |    500    |      5       |    0.93   |   0.67   |
+------------+-----------+--------------+-----------+----------+
```

In [86]:

```
import pandas as pd
def co_occurance_matrix(input_text,top_words,window_size):
    co_occur = pd.DataFrame(index=top_words, columns=top_words)

    for row,nrow in zip(top_words,range(len(top_words))):
        for colm,ncolm in zip(top_words,range(len(top_words))):
            count = 0
            if row == colm:
                co_occur.iloc[nrow,ncolm] = count
            else:
                for single_essay in input_text:
                    essay_split = single_essay.split(" ")
                    max_len = len(essay_split)
                    top_word_index = [index for index, split in enumerate(essay_split) if row in sp
it]
                    for index in top_word_index:
                        if index == 0:
                            count = count + essay_split[:window_size + 1].count(colm)
                        elif index == (max_len -1):
                            count = count + essay_split[-(window_size + 1):].count(colm)
                        else:
                            count = count + essay_split[index + 1 : (index + window_size + 1)].count
(colm)
                            if index < window_size:
                                count = count + essay_split[: index].count(colm)
                            else:
                                count = count + essay_split[(index - window_size): index].count(col
)
                co_occur.iloc[nrow,ncolm] = count

    return co_occur
```

In [87]:

```
corpus = ["abc def ijk pqr", "pqr klm opq", "lmn pqr xyz abc def pqr abc"]
words = ["abc","pqr","def"]
window_size =2

result = co_occurance_matrix(corpus,words,window_size)
print(result)
```

```
     abc  pqr  def
abc    0    3    3
pqr    3    0    2
def    3    2    0
```

In [ ]:

```

```