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`

Feature	Description
	• `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*
`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."

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
os.chdir('C:/Users/kingsubham27091995/Desktop/AppliedAiCouse/DonorsChoose')
```

1.1 Reading Data

In [2]:

```
project_data = pd.read_csv('train_data.csv',nrows=50000)
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 (50000, 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]:

```
# 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[5]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_categ
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5

In [6]:

```
project_grade_category = []

for i in range(len(project_data)):
    a = project_data["project_grade_category"][i].replace(" ", "_")
    project_grade_category.append(a)
```

In [7]:

```
project_grade_category[0:5]
```

Out[7]:

```
['Grades_PreK-2', 'Grades_6-8', 'Grades_6-8', 'Grades_PreK-2', 'Grades_PreK-2']
```

In [8]:

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

In [9]:

```
project_data["project_grade_category"] = project_grade_category
```

In [10]:

```
project_data.head(5)
```

Out[10]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_subject_ca
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Applied Learning
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Literacy & Language
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-27 01:10:09	Math & Science, Hist Civics
23374	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	CA	2016-04-27 02:04:15	Literacy & Language
49228	57854	p099430	4000cfe0c8b2df75a218347c1765e283	Ms.	IL	2016-04-27 07:19:44	Literacy & Language

1.2 preprocessing of `project_subject_categories`

In [11]:

```
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['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 [12]:

```
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]))
```

1.4 Clean Titles (Text preprocessing)

In [13]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
```

```
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "d
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [14]:

```
# 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 [15]:

```
clean_titles = []

for titles in tqdm(project_data["project_title"]):
    title = decontracted(titles)
    title = title.replace('\\r', ' ')
    title = title.replace('\\"', ' ')
    title = title.replace('\\n', ' ')
    title = re.sub('[^A-Za-z0-9]+', ' ', title)
    title = ' '.join(f for f in title.split() if f not in stopwords)
    clean_titles.append(title.lower().strip())
```

```
100%|██████████████████████████████████| 50000/50000 [00:01<00:00, 26736.08it/s]
```

In [16]:

```
project_data["clean_titles"] = clean_titles
```

In [17]:

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

1.5 Introducing new feature "Number of Words in Title"

In [18]:

```
title_word_count = []
```

In [19]:

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

In [20]:

```
project_data["title_word_count"] = title_word_count
```

In [21]:

```
project_data.head(5)
```

Out[21]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_essay_1	p
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	I recently read an article about giving studen...	I t in so
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	My students crave challenge, they eat obstacle...	W pu el so
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-27 01:10:09	It's the end of the school year. Routines have...	M de ch m c.
23374	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	CA	2016-04-27 02:04:15	Never has society so rapidly changed. Technolo...	O A Ju S
49228	57854	p099430	4000cfe0c8b2df75a218347c1765e283	Ms.	IL	2016-04-27 07:19:44	My students yearn for a classroom environment ...	I I pr te in

1.6 Combine 4 Project essays into 1 Essay

In [22]:

```
# 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)
```

1.7 Clean Essays (Text preprocessing)

In [23]:

```
clean_essay = []

for ess in tqdm(project_data["essay"]):
    ess = decontracted(ess)
    ess = ess.replace('\\r', ' ')
    ess = ess.replace('\\"', ' ')
    ess = ess.replace('\\n', ' ')
    ess = re.sub('[^A-Za-z0-9]+', ' ', ess)
    ess = ' '.join(f for f in ess.split() if f not in stopwords)
    clean_essay.append(ess.lower().strip())
```

```
100%|██████████| 50000/50000 [00:41<00:00, 1214.82it/s]
```

In [24]:

```
project_data["clean_essays"] = clean_essay
```

In [25]:

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

1.8 Introducing new feature "Number of Words in Essay"

In [26]:

```
essay_word_count = []
```

In [27]:

```
for ess in project_data["clean_essays"] :
    c = len(ess.split())
    essay_word_count.append(c)
```

In [28]:

```
project_data["essay_word_count"] = essay_word_count
```

In [29]:

```
project_data.head(5)
```

Out[29]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_essay_1	p
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	I recently read an article about giving studen...	I t in so
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	My students crave challenge, they eat obstacle...	W pu el so
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-27 01:10:09	It's the end of the school year. Routines have...	M de ch m c.
23374	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	CA	2016-04-27 02:04:15	Never has society so rapidly changed. Technolo...	O A Ju S
49228	57854	p099430	4000cfe0c8b2df75a218347c1765e283	Ms.	IL	2016-04-27 07:19:44	My students yearn for a classroom environment ...	I I pr te in

1.9 Calculate Sentiment Scores for the essays

In [30]:

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

In [31]:

```
analyser = SentimentIntensityAnalyzer()
```

In [32]:

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

for a in tqdm(project_data["clean_essays"]) :
    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%|██████████████████████████████████| 50000/50000 [08:58<00:00, 92.81it/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
```

In [37]:

```
project_data.head(5)
```

Out[37]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_essay_1	pr
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	I recently read an article about giving studen...	I f in sc
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	My students crave challenge, they eat obstacle...	W pu el sc
						2016-	It's the end of the	M de

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_essay_1	
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	04-27 01:1[illegible]	school year.	
23374	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	CA	2016-04-27 02:04:15	Never has society so rapidly changed. Technolo...	O A Ju S
49228	57854	p099430	4000cfe0c8b2df75a218347c1765e283	Ms.	IL	2016-04-27 07:19:44	My students yearn for a classroom environment ...	I I pr te in

5 rows × 24 columns

1.10 Test - Train Split

In [38]:

```
# train test split

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(project_data,
project_data['project_is_approved'], test_size=0.33, stratify = project_data['project_is_approved'
])
```

Preparing data for models

In [39]:

```
project_data.columns
```

Out[39]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'project_grade_category', 'clean_categories', 'clean_subcategories',
       'clean_titles', 'title_word_count', 'clean_essays', 'essay_word_count',
       'pos', 'neg', 'neu', 'compound'],
      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
```

2.1 Vectorizing Text data

A) Bag of Words (BOW) with min_df=10

Bag of words - Train Data - Essays

In [40]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).

vectorizer_bow_essay = CountVectorizer(min_df=10)

vectorizer_bow_essay.fit(X_train["clean_essays"])

text_bow_train = vectorizer_bow_essay.transform(X_train["clean_essays"])

print("Shape of matrix after one hot encoding ",text_bow_train.shape)
```

```
Shape of matrix after one hot encoding  (33500, 10424)
```

Bag of words - Test Data - Essays

In [41]:

```
text_bow_test = vectorizer_bow_essay.transform(X_test["clean_essays"])
print("Shape of matrix after one hot encoding ",text_bow_test.shape)
```

```
Shape of matrix after one hot encoding  (16500, 10424)
```

Bag of words - Train Data - Titles

In [42]:

```
vectorizer_bow_title = CountVectorizer(min_df=10)

vectorizer_bow_title.fit(X_train["clean_titles"])

title_bow_train = vectorizer_bow_title.transform(X_train["clean_titles"])
print("Shape of matrix after one hot encoding ",title_bow_train.shape)
```

```
Shape of matrix after one hot encoding  (33500, 1645)
```

Bag of words - Test Data - Titles

In [43]:

```
title_bow_test = vectorizer_bow_title.transform(X_test["clean_titles"])
print("Shape of matrix after one hot encoding ",title_bow_test.shape)
```

```
Shape of matrix after one hot encoding  (16500, 1645)
```

B) TFIDF vectorizer with min_df=10

TFIDF - Train Data - Essays

In [44]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer_tfidf_essay = TfidfVectorizer(min_df=10)
vectorizer_tfidf_essay.fit(X_train["clean_essays"])
```

```
text_tfidf_train = vectorizer_tfidf_essay.transform(X_train["clean_essays"])
print("Shape of matrix after one hot encoding ",text_tfidf_train.shape)
```

```
Shape of matrix after one hot encoding  (33500, 10424)
```

TFIDF - Test Data - Essays

In [45]:

```
text_tfidf_test = vectorizer_tfidf_essay.transform(X_test["clean_essays"])
print("Shape of matrix after one hot encoding ",text_tfidf_test.shape)
```

```
Shape of matrix after one hot encoding  (16500, 10424)
```

TFIDF - Train Data - Titles

In [46]:

```
vectorizer_tfidf_titles = TfidfVectorizer(min_df=10)

vectorizer_tfidf_titles.fit(X_train["clean_titles"])
title_tfidf_train = vectorizer_tfidf_titles.transform(X_train["clean_titles"])
print("Shape of matrix after one hot encoding ",title_tfidf_train.shape)
```

```
Shape of matrix after one hot encoding  (33500, 1645)
```

TFIDF - Test Data - Titles

In [47]:

```
title_tfidf_test = vectorizer_tfidf_titles.transform(X_test["clean_titles"])
print("Shape of matrix after one hot encoding ",title_tfidf_test.shape)
```

```
Shape of matrix after one hot encoding  (16500, 1645)
```

C) Using Pretrained Models : AVG W2V

In [48]:

```
'''
# 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[48]:

```
'\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 [49]:

```
# 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())
```

Train - Essays

In [50]:

```
# average Word2Vec
# compute average word2vec for each review.

avg_w2v_vectors_train = [];

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

```
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train.append(vector)

print(len(avg_w2v_vectors_train))
print(len(avg_w2v_vectors_train[0]))
```

```
100%|████████████████████████████████| 33500/33500 [00:14<00:00, 2280.22it/s]
```

```
33500
300
```

Test - Essays

In [51]:

```
# average Word2Vec
# compute average word2vec for each review.

avg_w2v_vectors_test = [];

for sentence in tqdm(X_test["clean_essays"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test.append(vector)

print(len(avg_w2v_vectors_test))
print(len(avg_w2v_vectors_test[0]))
```

```
100%|████████████████████████████████| 16500/16500 [00:07<00:00, 2275.40it/s]
```

```
16500
300
```

Train - Titles

In [52]:

```
# Similarly you can vectorize for title also

avg_w2v_vectors_titles_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train["clean_titles"]): # for each title
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_titles_train.append(vector)

print(len(avg_w2v_vectors_titles_train))
print(len(avg_w2v_vectors_titles_train[0]))
```

```
100%|████████████████████████████████| 33500/33500 [00:00<00:00, 40139.36it/s]
```

```
33500
300
```

Test - Titles

In [53]:

```
# Similarly you can vectorize for title also

avg_w2v_vectors_titles_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test["clean_titles"]): # for each title
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_titles_test.append(vector)

print(len(avg_w2v_vectors_titles_test))
print(len(avg_w2v_vectors_titles_test[0]))
```

```
100%|██████████████████████████████████| 16500/16500 [00:00<00:00, 38979.40it/s]
```

```
16500
300
```

D) Using Pretrained Models: TFIDF weighted W2V

Train - Essays

In [54]:

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

In [55]:

```
# Tfidf Word2Vec
# compute Tfidf word2vec for each review.
tfidf_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train["clean_essays"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_train.append(vector)

print(len(tfidf_w2v_vectors_train))
print(len(tfidf_w2v_vectors_train[0]))
```

```
100%|██████████████████████████████████| 33500/33500 [01:40<00:00, 333.96it/s]
```

```
33500
300
```

Test - Essays

In [56]:

```
# compute Tfidf word2vec for each review.

tfidf_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test["clean_essays"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_test.append(vector)

print(len(tfidf_w2v_vectors_test))
print(len(tfidf_w2v_vectors_test[0]))
```

```
100%|████████████████████████████████████| 16500/16500 [00:49<00:00, 333.55it/s]
```

```
16500
300
```

Train - Titles

In [57]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train["clean_titles"])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [58]:

```
# compute average word2vec for each review.

tfidf_w2v_vectors_titles_train = [];

for sentence in tqdm(X_train["clean_titles"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_titles_train.append(vector)

print(len(tfidf_w2v_vectors_titles_train))
print(len(tfidf_w2v_vectors_titles_train[0]))
```

```
100%|████████████████████████████████████| 33500/33500 [00:01<00:00, 20802.56it/s]
```

```
33500
300
```

Test - Titles

In [59]:

```
# compute average word2vec for each review.

tfidf_w2v_vectors_titles_test = [];

for sentence in tqdm(X_test["clean_titles"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_titles_test.append(vector)

print(len(tfidf_w2v_vectors_titles_test))
print(len(tfidf_w2v_vectors_titles_test[0]))
```

```
100%|██████████████████████████████████| 16500/16500 [00:00<00:00, 19842.31it/s]
```

```
16500
300
```

2.2 Vectorizing Numerical features

In [60]:

```
# 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()
price_data.head(2)
```

Out[60]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21

In [61]:

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

A) Price

In [62]:

```
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))

price_train = normalizer.transform(X_train['price'].values.reshape(-1,1))
price_test = normalizer.transform(X_test['price'].values.reshape(-1,1))

print("After vectorizations")
print(price_train.shape, y_train.shape)
print(price_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

B) Quantity

In [63]:

```
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['quantity'].values.reshape(-1,1))

quantity_train = normalizer.transform(X_train['quantity'].values.reshape(-1,1))
quantity_test = normalizer.transform(X_test['quantity'].values.reshape(-1,1))

print("After vectorizations")
print(quantity_train.shape, y_train.shape)
print(quantity_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

C) Number of Projects previously proposed by Teacher

In [64]:

```
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['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

prev_projects_train = normalizer.transform(X_train['teacher_number_of_previously_posted_projects']
.values.reshape(-1,1))
prev_projects_test = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].v
alues.reshape(-1,1))

print("After vectorizations")
print(prev_projects_train.shape, y_train.shape)
```

```
print(prev_projects_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

D) Title word Count

In [65]:

```
normalizer = Normalizer()

normalizer.fit(X_train['title_word_count'].values.reshape(-1,1))

title_word_count_train = normalizer.transform(X_train['title_word_count'].values.reshape(-1,1))
title_word_count_test = normalizer.transform(X_test['title_word_count'].values.reshape(-1,1))

print("After vectorizations")
print(title_word_count_train.shape, y_train.shape)
print(title_word_count_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

E) Essay word Count

In [66]:

```
normalizer = Normalizer()

normalizer.fit(X_train['essay_word_count'].values.reshape(-1,1))

essay_word_count_train = normalizer.transform(X_train['essay_word_count'].values.reshape(-1,1))
essay_word_count_test = normalizer.transform(X_test['essay_word_count'].values.reshape(-1,1))

print("After vectorizations")
print(essay_word_count_train.shape, y_train.shape)
print(essay_word_count_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

F) Essay Sentiments - pos

In [67]:

```
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
(33500, 1) (33500,)
(16500, 1) (16500,)
==================================================================================================
```

G) Essay Sentiments - neg

In [68]:

```
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
(33500, 1) (33500,)
(16500, 1) (16500,)
==================================================================================================
```

H) Essay Sentiments - neu

In [69]:

```
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
(33500, 1) (33500,)
(16500, 1) (16500,)
==================================================================================================
```

I) Essay Sentiments - compound

In [70]:

```
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
```

```
(33500, 1) (33500,)
(16500, 1) (16500,)
==========================================================================================================
```

2.3 Response coding for Categorical Data

In [71]:

```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing

def get_gv_fea_dict(alpha, feature, df):

    value_count = X_train[feature].value_counts()
    gv_dict = dict()

    # denominator will contain the number of time that particular feature occured in whole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to perticular class
        # vec is 2 dimensional vector
        vec = []
        for k in range(1,3):

            cls_cnt = X_train.loc[(X_train['project_is_approved']==k) & (X_train[feature]==i)]
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 20*alpha))

        gv_dict[i]=vec
    return gv_dict

def get_gv_feature(alpha, feature, df):

    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = X_train[feature].value_counts()

    gv_fea = []

    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/2,1/2])
    return gv_fea
```

School State- Response Coding

In [72]:

```
# alpha is used for laplace smoothing
alpha = 1

train_school_state = np.array(get_gv_feature(alpha, "school_state", X_train))
test_school_state = np.array(get_gv_feature(alpha, "school_state", X_test))
```

In [73]:

```
print(train_school_state.shape)
print(test_school_state.shape)
```

```
(33500, 2)
(16500, 2)
```

In [74]:

```
train_school_state
```

Out[74]:

```
array([[0.84210526, 0.02631579],
       [0.81730769, 0.01923077],
       [0.81789639, 0.00523286],
       ...,
       [0.85328023, 0.002096  ],
       [0.83763838, 0.01845018],
       [0.84538653, 0.00623441]])
```

Clean Categories - Response Coding

In [75]:

```
# alpha is used for laplace smoothing
alpha = 1

train_clean_categories = np.array(get_gv_feature(alpha, "clean_categories", X_train))
test_clean_categories = np.array(get_gv_feature(alpha, "clean_categories", X_test))
```

In [76]:

```
print(train_clean_categories.shape)
print(test_clean_categories.shape)
```

```
(33500, 2)
(16500, 2)
```

Clean Sub-Categories - Response Coding

In [77]:

```
# alpha is used for laplace smoothing
alpha = 1

train_clean_subcategories = np.array(get_gv_feature(alpha, "clean_subcategories", X_train))
test_clean_subcategories = np.array(get_gv_feature(alpha, "clean_subcategories", X_test))
```

In [78]:

```
print(train_clean_subcategories.shape)
print(test_clean_subcategories.shape)
```

```
(33500, 2)
(16500, 2)
```

Project_Grade_Category - Response Coding

In [79]:

```
# alpha is used for laplace smoothing
alpha = 1

train_project_grade_category = np.array(get_gv_feature(alpha, "project_grade_category", X_train))
test_project_grade_category = np.array(get_gv_feature(alpha, "project_grade_category", X_test))
```

In [80]:

```
print(train_project_grade_category.shape)
print(test_project_grade_category.shape)
```

```
(33500, 2)
(16500, 2)
```

Teacher Prefix - Response Coding

Teacher Prefix - Response Coding

In [81]:

```
# alpha is used for laplace smoothing
alpha = 1

train_teacher_prefix = np.array(get_gv_feature(alpha, "teacher_prefix", X_train))
test_teacher_prefix = np.array(get_gv_feature(alpha, "teacher_prefix", X_test))
```

In [82]:

```
print(train_teacher_prefix.shape)
print(test_teacher_prefix.shape)
```

```
(33500, 2)
(16500, 2)
```

Assignment 9: RF and GBDT

Response Coding: Example

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

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

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

     or

   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure

     seaborn heat maps with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**
   - You can choose either of the plotting techniques: 3d plot or heat map
   - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.

   - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points

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

Note: Data Leakage

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

Set 1: Categorical, Numerical features + Project_title(BOW) + Preprocessed_essay (BOW with min_df=10)

In [109]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = hstack((train_clean_categories, train_clean_subcategories, train_school_state,
train_project_grade_category, train_teacher_prefix,price_train, quantity_train,
prev_projects_train, title_word_count_train, essay_word_count_train, essay_sent_pos_train,
essay_sent_neg_train, essay_sent_neu_train, essay_sent_comp_train, title_bow_train, text_bow_train
)).tocsr()
X_te = hstack((test_clean_categories, test_clean_subcategories, test_school_state,
test_project_grade_category, test_teacher_prefix,price_test, quantity_test, prev_projects_test,
title_word_count_test, essay_word_count_test, essay_sent_pos_test, essay_sent_neg_test, essay_sent_
neu_test, essay_sent_comp_test, title_bow_test, text_bow_test)).tocsr()
```

In [89]:

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

```
Final Data matrix
(33500, 12088) (33500,)
(16500, 12088) (16500,)
====================================================================================================
```

A) RandomizedSearchCV (K fold Cross Validation)

In [86]:

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
```

Applying Random Forest

In [ ]:

```
rf = RandomForestClassifier()

parameters = {'n_estimators': [10, 100, 500, 1000], 'max_depth':[10, 50, 100, 500, 1000]}

clf = RandomizedSearchCV(rf, parameters, cv= 3, scoring='roc_auc')

clf.fit(X_tr, y_train)
```

In [92]:

```
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
print(clf.best_params_)
```

```
{'n_estimators': 1000, 'max_depth': 10}
```

In [93]:

```
train_auc
```

Out[93]:

```
array([0.94923064, 0.99999757, 0.99999947, 0.99999998, 0.99999998,
       0.80450695, 0.99967616, 0.99947546, 0.99995109, 0.83801762])
```

In [94]:

```
cv_auc
```

Out[94]:

```
array([0.5891218 , 0.61176301, 0.63110397, 0.60439054, 0.59222454,
       0.66091569, 0.54791642, 0.64469848, 0.66353631, 0.67222748])
```

Plot for Train & Cross Validation Data

In [105]:

```
import plotly.plotly as py
import plotly.graph_objs as go
```

In [96]:

```
import plotly
plotly.tools.set_credentials_file(username='Subham27091995', api_key='dDPzdxmhgzisjZHpBrwL')
```

In [97]:

```
x1 = [0.94923064, 0.99999757, 0.99999947, 0.99999998, 0.99999998,
       0.80450695, 0.99967616, 0.99947546, 0.99995109, 0.83801762]
```

In [98]:

```
x2 = [0.5891218 , 0.61176301, 0.63110397, 0.60439054, 0.59222454,
       0.66091569, 0.54791642, 0.64469848, 0.66353631, 0.67222748]
```

In [100]:

```
y1 = pd.Series([10,100,500,1000,10,100,500,1000,5,10], index = x1)
```

In [101]:

```
z1 = pd.Series([10,10,50,50,100,100,500,500,1000,1000],index = x1)
```

In [102]:

```
trace1 = go.Scatter3d(
    x=x1, y=y1, z=z1,
    name = 'Train',
    marker=dict(
        size=4,
```

```
        colorscale='Viridis',
    ),
    line=dict(
        color='#1f77b4',
        width=1
    )
)

trace2 = go.Scatter3d(
    x=x2, y=y1, z=z1,
    name = 'Test',
    marker=dict(
        size=4,
        colorscale='Viridis',
    ),
    line=dict(
        color='#b45c1f',
        width=1
    )
)
```

In [103]:

```
data = [trace1, trace2]
```

In [104]:

```
layout = dict(
    width=800,
    height=700,
    autosize=False,
    title='Hyper Parameter Tuning -- Random Forests - BOW',
    scene=dict(
        xaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        yaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        zaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        camera=dict(
            up=dict(
                x=0,
                y=0,
                z=1
            ),
            eye=dict(
                x=-1.7428,
                y=1.0707,
                z=0.7100,
            )
        ),
        aspectratio = dict( x=1, y=1, z=0.7 ),
        aspectmode = 'manual'
    ),
)
```

In [105]:

```
fig = dict(data=data, layout=layout)

py.iplot(fig, filename='Random-Forests-a', height=700)
```

Out[105]:

Observations :

1) Number of estimators as 1000, performs decently on both Train as well as Cross Validation Data.

2) 10 as the value for maximum depth is considered.

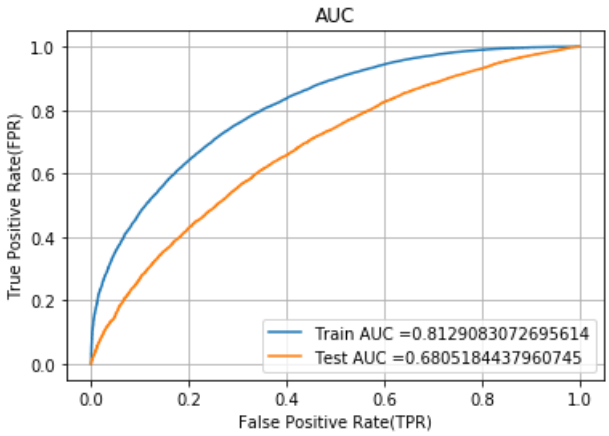
B) Train the model using the best hyper parameter value

In [83]:

```
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 [107]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

model = RandomForestClassifier(max_depth = 10, n_estimators = 1000)

model.fit(X_tr, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(model, X_tr)
y_test_pred = batch_predict(model, 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("False Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



C) Confusion Matrix

In [108]:

```
def predict(proba, threshould, fpr, tpr):

    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)
    return predictions
```

Train Data

In [109]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499999979647145 for threshold 0.837
[[ 5542  5541]
 [ 6263 55850]]
```

In [110]:

```
conf_matr_df_train_1_rf = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred,
tr_thresholds, train_fpr, train_fpr)), range(2),range(2))
```

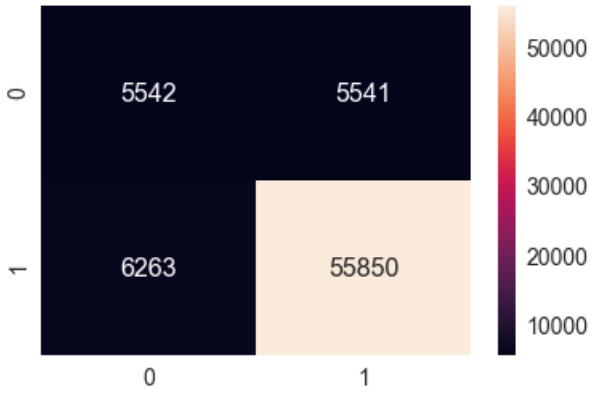
```
the maximum value of tpr*(1-fpr) 0.2499999979647145 for threshold 0.837
```

In [111]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_1_rf, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[111]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x8ccb93e278>
```



Test data

In [112]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.845
[[ 3338  2121]
 [10729 19864]]
```

In [113]:

```
conf_matr_df_test_1_rf = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds,
test_fpr, test_fpr)), range(2),range(2))
```

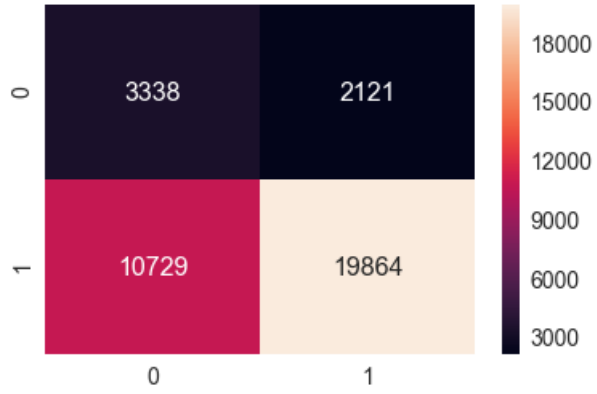
```
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.845
```

In [114]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_1_rf, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[114]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x8c80360898>
```



Applying GBDT

In [91]:

```
from sklearn.ensemble import GradientBoostingClassifier
gbdt = GradientBoostingClassifier()

parameters = {'n_estimators': [10, 100, 500, 1000], 'max_depth':[10, 50, 100, 500, 1000]}

clf = RandomizedSearchCV(gbdt, parameters, cv= 3, scoring='roc_auc',n_jobs=-1)

clf.fit(X_tr, y_train)
```

Out[91]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
          estimator=GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_sampl...      subsample=1.0, tol=0.0001,
validation_fraction=0.1,
              verbose=0, warm_start=False),
          fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'n_estimators': [10, 100, 500, 1000], 'max_depth': [10, 50, 100, 500
, 1000]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring='roc_auc', verbose=0)
```

In [92]:

```
train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']
print(clf.best_params_)
```

```
{'n_estimators': 500, 'max_depth': 10}
```

Observations :

1) Number of estimators as 500, performs decently on both Train as well as Cross Validation Data.

2) 10 as the value for maximum depth is considered.

In [93]:

```
train_auc
```

Out[93]:

```
array([1.        , 0.98522027, 0.84030395, 1.        , 0.99963858,
       1.        , 1.        , 0.99959273, 1.        , 1.        ])
```

In [94]:

```
cv_auc
```

Out[94]:

```
array([0.57805871, 0.67238116, 0.639614  , 0.52707894, 0.55004282,
       0.52047871, 0.57801874, 0.6794619 , 0.57886902, 0.61772489])
```

Plot for Train & Cross Validation Data

In [102]:

```
import plotly
plotly.tools.set_credentials_file(username='Subham27091995', api_key='wfHUdo9zCFSMJacH3jHt')
```

In [96]:

```
x1 = [1.        , 0.98522027, 0.84030395, 1.        , 0.99963858,
       1.        , 1.        , 0.99959273, 1.        , 1.        ]
```

In [97]:

```
x2 = [0.57805871, 0.67238116, 0.639614  , 0.52707894, 0.55004282,
       0.52047871, 0.57801874, 0.6794619 , 0.57886902, 0.61772489]
```

In [98]:

```
y1 = pd.Series([10,100,500,1000,10,100,500,1000,5,10], index = x1)
```

In [99]:

```
z1 = pd.Series([10,10,50,50,100,100,500,500,1000,1000],index = x1)
```

In [100]:

```
trace1 = go.Scatter3d(
    x=x1, y=y1, z=z1,
    name = 'Train',
    marker=dict(
        size=4,
        colorscale='Viridis',
    ),
    line=dict(
        color='#1f77b4',
        width=1
    )
)

trace2 = go.Scatter3d(
    x=x2, y=y1, z=z1,
    name = 'Test',
    marker=dict(
        size=4,
        colorscale='Viridis',
    ),
    line=dict(
        color='#b45c1f',
        width=1
    )
```

```
)
```

In [101]:

```
data = [trace1, trace2]
```

In [103]:

```
layout = dict(
    width=800,
    height=700,
    autosize=False,
    title='Hyper Parameter Tuning -- Random Forests - BOW2',
    scene=dict(
        xaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        yaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        zaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        camera=dict(
            up=dict(
                x=0,
                y=0,
                z=1
            ),
            eye=dict(
                x=-1.7428,
                y=1.0707,
                z=0.7100,
            )
        ),
        aspectratio = dict( x=1, y=1, z=0.7 ),
        aspectmode = 'manual'
    ),
)
```

In [106]:

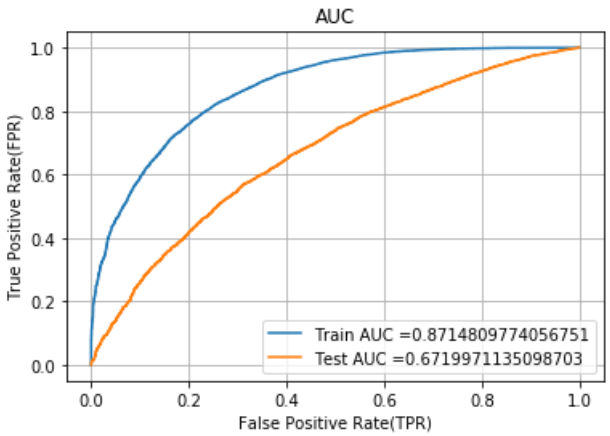
```
fig = dict(data=data, layout=layout)

py.iplot(fig, filename='GBDT-bow', height=700)
```

Out[106]:

B) Train the model using the best hyper parameter value

In [95]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

model = GradientBoostingClassifier(max_depth = 10, n_estimators = 500)

model.fit(X_tr, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(model, X_tr)
y_test_pred = batch_predict(model, 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("False Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



C) Confusion Matrix

In [110]:

```
def predict(proba, threshould, fpr, tpr):

    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)
    return predictions
```

Train Data

In [111]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.831
[[ 2584  2584]
 [ 1093 27239]]
```

In [112]:

```
conf_matr_df_train_2_rf = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred,
tr_thresholds, train_fpr, train_fpr)), range(2),range(2))
```

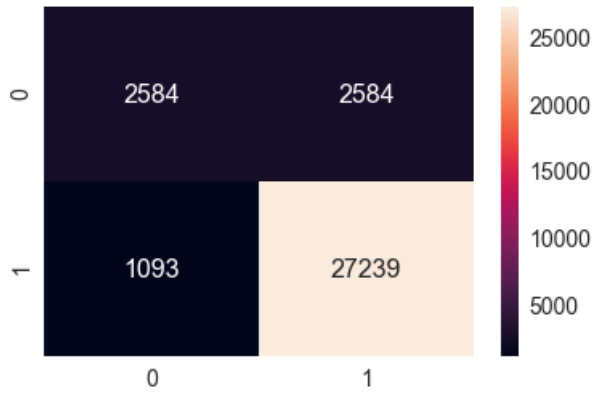
```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.831
```

In [113]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_2_rf, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[113]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xdd957035c0>
```



Test Data

In [114]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.84
[[ 1238  1308]
 [ 3471 10483]]
```

In [115]:

```
conf_matr_df_test_2_rf = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds,
test_fpr, test_fpr)), range(2),range(2))
```

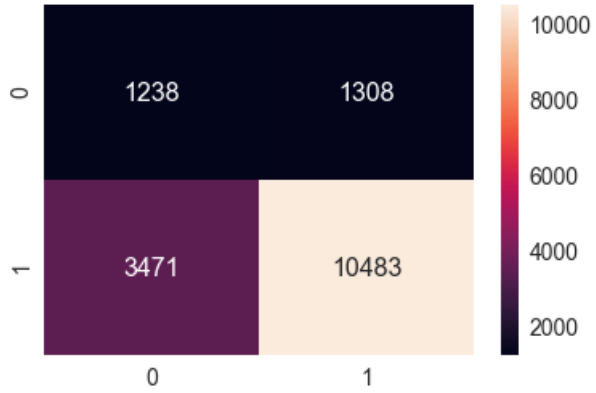
```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.84
```

In [116]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_2_rf, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[116]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xdd95f82b70>
```



Set 2 : Categorical, Numerical features + Project_title(TFIDF) + Preprocessed_essay (TFIDF min_df=10)

In [117]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = hstack((train_clean_categories, train_clean_subcategories, train_school_state,
train_project_grade_category, train_teacher_prefix,price_train, quantity_train,
prev_projects_train, title_word_count_train, essay_word_count_train, essay_sent_pos_train,
essay_sent_neg_train, essay_sent_neu_train, essay_sent_comp_train, title_tfidf_train,
text_tfidf_train)).tocsr()
X_te = hstack((test_clean_categories, test_clean_subcategories, test_school_state,
test_project_grade_category, test_teacher_prefix,price_test, quantity_test, prev_projects_test,
title_word_count_test, essay_word_count_test, essay_sent_pos_test, essay_sent_neg_test, essay_sent_
neu_test, essay_sent_comp_test, title_tfidf_test, text_tfidf_test)).tocsr()
```

In [118]:

```
print("Final Data matrix")
```

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

```
Final Data matrix
(33500, 12088) (33500,)
(16500, 12088) (16500,)
====================================================================================================
```

A) RandomizedSearchCV (K fold Cross Validation)

Applying Random Forest

In [171]:

```
rf = RandomForestClassifier()

parameters = {'n_estimators': [10, 100, 500, 1000], 'max_depth':[10, 50, 100, 500, 1000]}

clf = RandomizedSearchCV(rf, parameters, cv= 3, scoring='roc_auc',n_jobs=-1)

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']
```

In [172]:

```
print(clf.best_params_)
```

```
{'n_estimators': 1000, 'max_depth': 50}
```

Observations :

1) Number of estimators as 1000, performs decently on both Train as well as Cross Validation Data.

2) 50 as the value for maximum depth is considered.

In [173]:

```
train_auc
```

Out[173]:

```
array([0.99999998, 0.95793273, 0.69301964, 0.99999606, 0.99999998,
       0.99967026, 0.85562876, 0.99996606, 0.99999998, 0.99999998])
```

In [174]:

```
cv_auc
```

Out[174]:

```
array([0.68573045, 0.59117539, 0.62669499, 0.68506437, 0.65999251,
       0.58317326, 0.67538978, 0.68914466, 0.68438161, 0.66234867])
```

Plot for Train & Cross Validation Data

In [175]:

```
x1 = [0.99999998, 0.95793273, 0.69301964, 0.99999606, 0.99999998,
```

```
       0.99967026, 0.85562876, 0.99996606, 0.99999998, 0.99999998]
```

In [176]:

```
x2 = [0.68573045, 0.59117539, 0.62669499, 0.68506437, 0.65999251,
       0.58317326, 0.67538978, 0.68914466, 0.68438161, 0.66234867]
```

In [178]:

```
y1 = pd.Series([10,100,500,1000,10,100,500,1000,5,10], index = x1)
```

In [179]:

```
z1 = pd.Series([10,10,50,50,100,100,500,500,1000,1000],index = x1)
```

In [180]:

```
trace1 = go.Scatter3d(
    x=x1, y=y1, z=z1,
    name = 'Train',
    marker=dict(
        size=4,
        colorscale='Viridis',
    ),
    line=dict(
        color='#1f77b4',
        width=1
    )
)

trace2 = go.Scatter3d(
    x=x2, y=y1, z=z1,
    name = 'Test',
    marker=dict(
        size=4,
        colorscale='Viridis',
    ),
    line=dict(
        color='#b45c1f',
        width=1
    )
)
```

In [181]:

```
data = [trace1, trace2]
```

In [182]:

```
layout = dict(
    width=800,
    height=700,
    autosize=False,
    title='Hyper Parameter Tuning -- Random Forests - TFIDF',
    scene=dict(
        xaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        yaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        zaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
```

```
            backgroundcolor='rgb(230, 230,230)'
        ),
        camera=dict(
            up=dict(
                x=0,
                y=0,
                z=1
            ),
            eye=dict(
                x=-1.7428,
                y=1.0707,
                z=0.7100,
            )
        ),
        aspectratio = dict( x=1, y=1, z=0.7 ),
        aspectmode = 'manual'
    ),
)
```

In [183]:

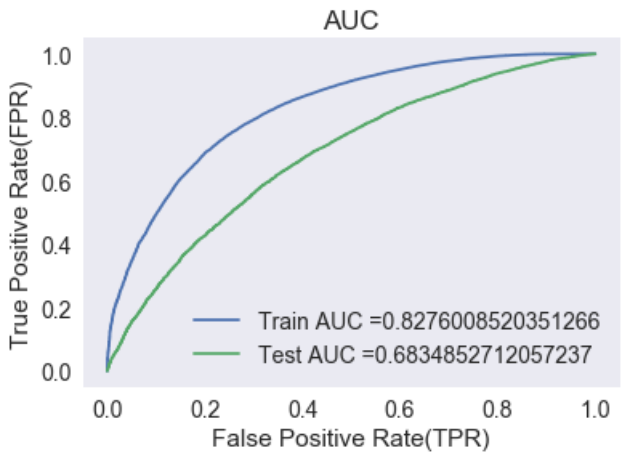
```
fig = dict(data=data, layout=layout)

py.iplot(fig, filename='Random-Forests-c', height=700)
```

Out[183]:

B) Train the model using the best hyper parameter value

In [184]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

model = RandomForestClassifier(max_depth = 10, n_estimators = 1000)

model.fit(X_tr, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(model, X_tr)
y_test_pred = batch_predict(model, 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("False Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



C) Confusion Matrix

In [185]:

```
def predict(proba, threshould, fpr, tpr):

    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)
    return predictions
```

Train Data

In [186]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
```

```
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499999979647145 for threshold 0.836
[[ 5542  5541]
 [ 5303 56810]]
```

In [187]:

```
conf_matr_df_train_2_rf = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred,
tr_thresholds, train_fpr, train_fpr)), range(2),range(2))
```

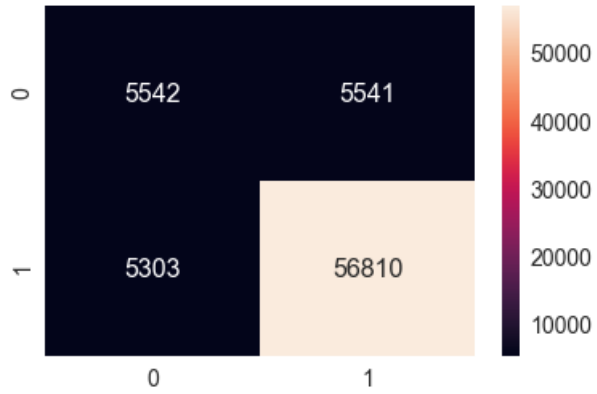
```
the maximum value of tpr*(1-fpr) 0.2499999979647145 for threshold 0.836
```

In [188]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_2_rf, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[188]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x8c80309470>
```



Test Data

In [189]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.847
[[ 3294  2165]
 [10231 20362]]
```

In [190]:

```
conf_matr_df_test_2_rf = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds,
test_fpr, test_fpr)), range(2),range(2))
```

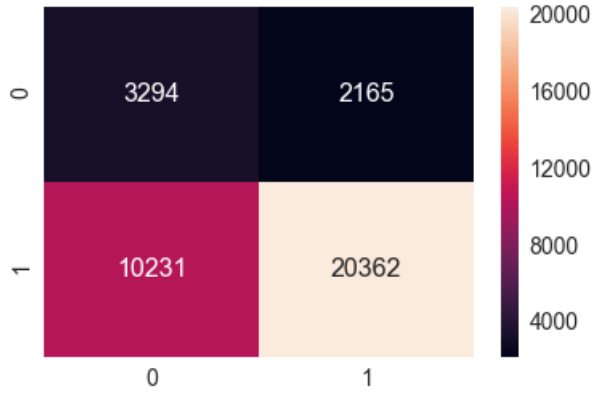
```
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.847
```

In [191]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_2_rf, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[191]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x8c8037ca20>
```



Applying GBDT

In [119]:

```
from sklearn.ensemble import GradientBoostingClassifier
gbdt = GradientBoostingClassifier()

parameters = {'n_estimators': [10, 100, 500, 1000], 'max_depth':[10, 50, 100, 500, 1000]}

clf = RandomizedSearchCV(gbdt, parameters, cv= 3, scoring='roc_auc',n_jobs=-1)

clf.fit(X_tr, y_train)
```

Out[119]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
          estimator=GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_sampl...      subsample=1.0, tol=0.0001,
validation_fraction=0.1,
              verbose=0, warm_start=False),
          fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'n_estimators': [10, 100, 500, 1000], 'max_depth': [10, 50, 100, 500
, 1000]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring='roc_auc', verbose=0)
```

In [120]:

```
train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']
print(clf.best_params_)
```

```
{'n_estimators': 100, 'max_depth': 10}
```

Observations :

1) Number of estimators as 100, performs decently on both Train as well as Cross Validation Data.

2) 10 as the value for maximum depth is considered.

In [121]:

```
train_auc
```

Out[121]:

```
array([1.        , 1.        , 0.84814726, 1.        , 1.        ,
       0.99999999, 1.        , 0.98985758, 1.        , 1.        ])
```

In [122]:

```
cv_auc
```

Out[122]:

```
array([0.59985359, 0.54047509, 0.63224996, 0.61783935, 0.63820281,
       0.65928622, 0.62913588, 0.66429253, 0.63733355, 0.6151373 ])
```

Plot for Train & Cross Validation Data

In [124]:

```
x1 = [1.        , 0.98522027, 0.84030395, 1.        , 0.99963858,
      1.        , 1.        , 0.99959273, 1.        , 1.        ]
```

In [125]:

```
x2 = [0.59985359, 0.54047509, 0.63224996, 0.61783935, 0.63820281,
      0.65928622, 0.62913588, 0.66429253, 0.63733355, 0.6151373 ]
```

In [126]:

```
y1 = pd.Series([10,100,500,1000,10,100,500,1000,5,10], index = x1)
```

In [127]:

```
z1 = pd.Series([10,10,50,50,100,100,500,500,1000,1000],index = x1)
```

In [128]:

```
trace1 = go.Scatter3d(
    x=x1, y=y1, z=z1,
    name = 'Train',
    marker=dict(
        size=4,
        colorscale='Viridis',
    ),
    line=dict(
        color='#1f77b4',
        width=1
    )
)

trace2 = go.Scatter3d(
    x=x2, y=y1, z=z1,
    name = 'Test',
    marker=dict(
        size=4,
        colorscale='Viridis',
    ),
    line=dict(
        color='#b45c1f',
        width=1
    )
)
```

In [129]:

```
data = [trace1, trace2]
```

In [130]:

```
layout = dict(
    width=800,
    height=700,
    autosize=False,
    title='Hyper Parameter Tuning -- Random Forests - TFIDF2',
    scene=dict(
        xaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        yaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        zaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        camera=dict(
            up=dict(
                x=0,
                y=0,
                z=1
            ),
            eye=dict(
                x=-1.7428,
                y=1.0707,
                z=0.7100,
            )
        ),
        aspectratio = dict( x=1, y=1, z=0.7 ),
        aspectmode = 'manual'
    ),
)
```

In [131]:

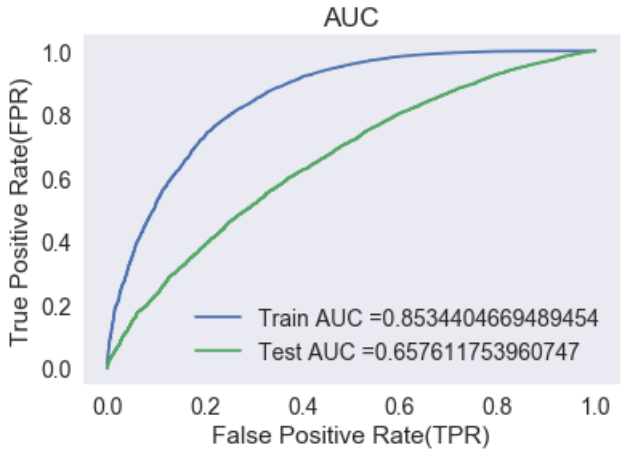
```
fig = dict(data=data, layout=layout)

py.iplot(fig, filename='GBDT-tfidf', height=700)
```

Out[131]:

B) Train the model using the best hyper parameter value

In [123]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

model = GradientBoostingClassifier(max_depth = 10, n_estimators = 100)

model.fit(X_tr, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(model, X_tr)
y_test_pred = batch_predict(model, 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("False Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



C) Confusion Matrix

In [132]:

```
def predict(proba, threshould, fpr, tpr):
```

```
    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)
    return predictions
```

Train Data

In [133]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499999625583491 for threshold 0.827
[[ 2585  2583]
 [ 1186 27146]]
```

In [134]:

```
conf_matr_df_train_2_rf = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred,
tr_thresholds, train_fpr, train_fpr)), range(2),range(2))
```

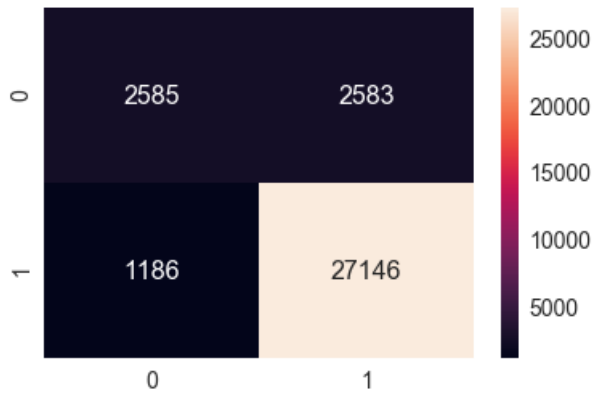
```
the maximum value of tpr*(1-fpr) 0.2499999625583491 for threshold 0.827
```

In [135]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_2_rf, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[135]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xdd9c4406a0>
```



Test Data

In [136]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.844
[[1383 1163]
 [4527 9427]]
```

In [137]:

```
conf_matr_df_test_2_rf = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds,
test_fpr, test_fpr)), range(2),range(2))
```

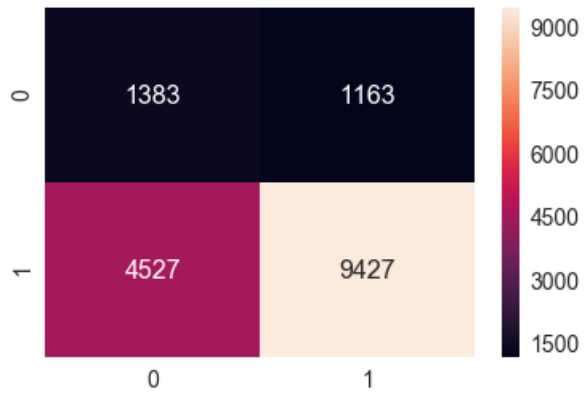
```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.844
```

In [138]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_2_rf, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[138]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xdd9c47ba90>
```



Set 3 : Categorical, Numerical features + Project_title(AVG W2V) + Preprocessed_essay (AVG W2V)

In [142]:

```
avg_w2v_vectors_train2d = np.array(avg_w2v_vectors_train)
avg_w2v_vectors_train2d.shape
```

Out[142]:

```
(33500, 300)
```

In [143]:

```
avg_w2v_vectors_titles_train2d = np.array(avg_w2v_vectors_titles_train)
avg_w2v_vectors_titles_train2d.shape
```

Out[143]:

```
(33500, 300)
```

In [144]:

```
avg_w2v_vectors_test2d = np.array(avg_w2v_vectors_test)
avg_w2v_vectors_test2d.shape
```

Out[144]:

```
(16500, 300)
```

In [145]:

```
avg_w2v_vectors_titles_test2d = np.array(avg_w2v_vectors_titles_test)
avg_w2v_vectors_titles_test2d.shape
```

Out[145]:

```
(16500, 300)
```

In [146]:

```
print(train_clean_categories.shape)
print(train_clean_subcategories.shape)
print( train_school_state.shape)
print(train_project_grade_category.shape)
print(train_teacher_prefix.shape)
print(price_train.shape)
print(quantity_train.shape)
print( prev_projects_train.shape)
print(title_word_count_train.shape)
print(essay_word_count_train.shape)
print(essay_sent_pos_train.shape)
print(essay_sent_neg_train.shape)
print(essay_sent_neu_train.shape)
print( essay_sent_comp_train.shape)
print(avg_w2v_vectors_titles_train2d.shape)
print(avg_w2v_vectors_train2d.shape)
```

```
(33500, 2)
(33500, 2)
(33500, 2)
(33500, 2)
(33500, 2)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 300)
(33500, 300)
```

In [78]:

```
print(type(essay_sent_neu_train))
```

```
<class 'numpy.ndarray'>
```

In [79]:

```
print(type(train_clean_subcategories))
```

```
<class 'numpy.ndarray'>
```

In [147]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = np.hstack((train_clean_categories, train_clean_subcategories, train_school_state,
train_project_grade_category, train_teacher_prefix,price_train, quantity_train,
prev_projects_train, title_word_count_train, essay_word_count_train, essay_sent_pos_train,
essay_sent_neg_train, essay_sent_neu_train, essay_sent_comp_train, avg_w2v_vectors_titles_train, a
vg_w2v_vectors_train))
X_te = np.hstack((test_clean_categories, test_clean_subcategories, test_school_state,
```

```
test_project_grade_category, test_teacher_prefix,price_test, quantity_test, prev_projects_test,
title_word_count_test, essay_word_count_test, essay_sent_pos_test, essay_sent_neg_test, essay_sent_
neu_test, essay_sent_comp_test, avg_w2v_vectors_titles_test, avg_w2v_vectors_test))
```

In [148]:

```
print(X_tr.shape)
print(X_te.shape)
print("="*100)
```

```
(33500, 619)
(16500, 619)
====================================================================================================
```

A) GridSearchCV (K fold Cross Validation)

Applying Random Forest

In [121]:

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()

parameters = {'n_esti
              mators': [10, 100, 500, 1000], 'max_depth':[10, 50, 100, 500, 1000]}

clf = RandomizedSearchCV(rf, parameters, cv= 3, scoring='roc_auc',n_jobs=-1)

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']
print(clf.best_params_)
```

```
{'n_estimators': 500, 'max_depth': 50}
```

In [122]:

```
train_auc
```

Out[122]:

```
array([1.        , 1.        , 1.        , 1.        , 0.99954974,
       1.        , 1.        , 1.        , 1.        , 1.        ])
```

In [92]:

```
cv_auc
```

Out[92]:

```
array([0.65410349, 0.61735948, 0.62606581, 0.66609313, 0.57406451,
       0.62579185, 0.62741507, 0.57205151, 0.59841727, 0.61227614])
```

Plot for Train & Cross Validation Data

In [95]:

```
import plotly
plotly.tools.set_credentials_file(username='Subham27091995', api_key='dDPzdxmhgzisjZHpBrwL')
```

In [96]:

```
import plotly.plotly as py
import plotly.graph_objs as go
```

In [97]:

```
x1 = [0.96755591, 1.         , 1.         , 0.97718481, 0.99958176,
       1.         , 1.         , 0.99946105, 0.88962584, 1.         ]
```

In [98]:

```
x2 = [0.65410349, 0.61735948, 0.62606581, 0.66609313, 0.57406451,
       0.62579185, 0.62741507, 0.57205151, 0.59841727, 0.61227614]
```

In [101]:

```
y1 = pd.Series([10,100,500,1000,10,100,500,1000,5,10], index = x1)
```

In [102]:

```
z1 = pd.Series([10,10,50,50,100,100,500,500,1000,1000],index = x1)
```

In [103]:

```
trace1 = go.Scatter3d(
    x=x1, y=y1, z=z1,
    name = 'Train',
    marker=dict(
        size=4,
        colorscale='Viridis',
    ),
    line=dict(
        color='#1f77b4',
        width=1
    )
)

trace2 = go.Scatter3d(
    x=x2, y=y1, z=z1,
    name = 'Test',
    marker=dict(
        size=4,
        colorscale='Viridis',
    ),
    line=dict(
        color='#b45c1f',
        width=1
    )
)
```

In [104]:

```
data = [trace1, trace2]
```

In [105]:

```
layout = dict(
    width=800,
    height=700,
    autosize=False,
    title='Hyper Parameter Tuning -- Random Forests - AVG W2V',
    scene=dict(
        xaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        yaxis=dict(
            gridcolor='rgb(255, 255, 255)'
```

```
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        zaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        camera=dict(
            up=dict(
                x=0,
                y=0,
                z=1
            ),
            eye=dict(
                x=-1.7428,
                y=1.0707,
                z=0.7100,
            )
        ),
        aspectratio = dict( x=1, y=1, z=0.7 ),
        aspectmode = 'manual'
    ),
)
```

In [106]:

```
fig = dict(data=data, layout=layout)

py.iplot(fig, filename='Random-Forests-e', height=700)
```

Out[106]:

Observations :

1) Number of estimators as 500 performs decently on both Train as well as Cross Validation Data.

2) 50 as the value for maximum depth is considered.

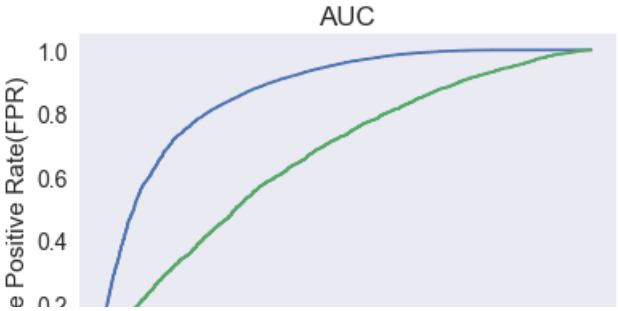
B) Train the model using the best hyper parameter value

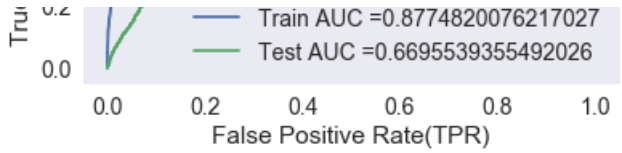
In [108]:

```
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 [139]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

model = RandomForestClassifier(max_depth = 10, n_estimators = 500)

model.fit(X_tr, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(model, X_tr)
y_test_pred = batch_predict(model, 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("False Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```





C) Confusion Matrix

In [110]:

```
def predict(proba, threshould, fpr, tpr):

    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)
    return predictions
```

Train Data

In [111]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999996255834908 for threshold 0.747
[[ 2583  2585]
 [    8 28324]]
```

In [112]:

```
conf_matr_df_train_3_rf = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred,
tr_thresholds, train_fpr, train_fpr)), range(2),range(2))
```

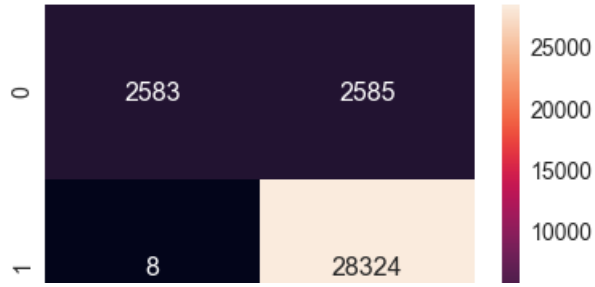
```
the maximum value of tpr*(1-fpr) 0.24999996255834908 for threshold 0.747
```

In [113]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_3_rf, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[113]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x931b5fc048>
```



Test Data

In [114]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.802
[[  725  1821]
 [ 1587 12367]]
```

In [115]:

```
conf_matr_df_test_3_rf = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds,
test_fpr, test_fpr)), range(2),range(2))
```

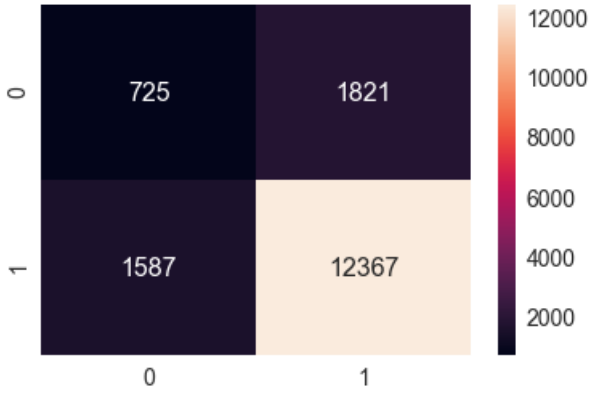
```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.802
```

In [116]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_3_rf, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[116]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x931bf25940>
```



Applying GBDT

In [149]:

```
from sklearn.ensemble import GradientBoostingClassifier
gbdt = GradientBoostingClassifier()

parameters = {'n_estimators': [10, 100, 500, 1000], 'max_depth':[10, 50, 100, 500, 1000]}

clf = RandomizedSearchCV(gbdt, parameters, cv= 3, scoring='roc_auc',n_jobs=-1)

clf.fit(X_tr, y_train)
```

Out[149]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
```

```
          estimator=GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_sampl...      subsample=1.0, tol=0.0001,
validation_fraction=0.1,
              verbose=0, warm_start=False),
          fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'n_estimators': [10, 100, 500, 1000], 'max_depth': [10, 50, 100, 500
, 1000]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring='roc_auc', verbose=0)
```

In [151]:

```
train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']
print(clf.best_params_)
```

```
{'n_estimators': 500, 'max_depth': 10}
```

Observations :

1) Number of estimators as 500, performs decently on both Train as well as Cross Validation Data.

2) 10 as the value for maximum depth is considered.

In [ ]:

```
train_auc
```

In [153]:

```
cv_auc
```

Out[153]:

```
array([0.57571938, 0.53152818, 0.53089381, 0.65613728, 0.57969573,
       0.57723286, 0.53263914, 0.58455066, 0.58162209, 0.57993513])
```

Plot for Train & Cross Validation Data

In [154]:

```
x1 = [0.99999998, 0.95793273, 0.69301964, 0.99999606, 0.99999998,
       0.99967026, 0.85562876, 0.99996606, 0.99999998, 0.99999998]
```

In [155]:

```
x2= [0.57571938, 0.53152818, 0.53089381, 0.65613728, 0.57969573,
       0.57723286, 0.53263914, 0.58455066, 0.58162209, 0.57993513]
```

In [156]:

```
y1 = pd.Series([10,100,500,1000,10,100,500,1000,5,10], index = x1)
```

In [157]:

```
z1 = pd.Series([10,10,50,50,100,100,500,500,1000,1000],index = x1)
```

In [158]:

```
trace1 = go.Scatter3d(
```

```
    x=x1, y=y1, z=z1,
    name = 'Train',
    marker=dict(
        size=4,
        colorscale='Viridis',
    ),
    line=dict(
        color='#1f77b4',
        width=1
    )
)

trace2 = go.Scatter3d(
    x=x2, y=y1, z=z1,
    name = 'Test',
    marker=dict(
        size=4,
        colorscale='Viridis',
    ),
    line=dict(
        color='#b45c1f',
        width=1
    )
)
```

In [159]:

```
data = [trace1, trace2]
```

In [160]:

```
layout = dict(
    width=800,
    height=700,
    autosize=False,
    title='Hyper Parameter Tuning -- GBDT- AVG W2V',
    scene=dict(
        xaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        yaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        zaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        camera=dict(
            up=dict(
                x=0,
                y=0,
                z=1
            ),
            eye=dict(
                x=-1.7428,
                y=1.0707,
                z=0.7100,
            )
        ),
        aspectratio = dict( x=1, y=1, z=0.7 ),
        aspectmode = 'manual'
    ),
)
```

In [162]:

```
import plotly
plotly.tools.set_credentials_file(username='Subham27091995', api_key='40QRf2k9LYb3nAcTUkSD')
```

In [163]:

```
fig = dict(data=data, layout=layout)

py.iplot(fig, filename='GBDT-avg', height=700)
```

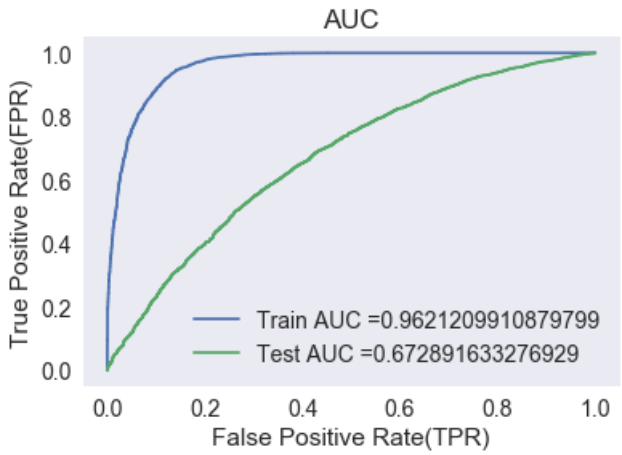
Out[163]:

B) Train the model using the best hyper parameter value

In [164]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

model = GradientBoostingClassifier(max_depth = 10, n_estimators = 1000)

model.fit(X_tr, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(model, X_tr)
y_test_pred = batch_predict(model, 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("False Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



Confusion Matrix

Train Data

In [165]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499915756285405 for threshold 0.747
[[ 2569  2599]
 [    2 28330]]
```

In [166]:

```
conf_matr_df_train_3_rf = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred,
tr_thresholds, train_fpr, train_fpr)), range(2),range(2))
```

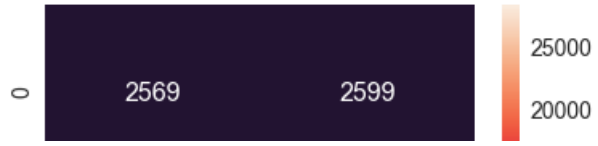
```
the maximum value of tpr*(1-fpr) 0.2499915756285405 for threshold 0.747
```

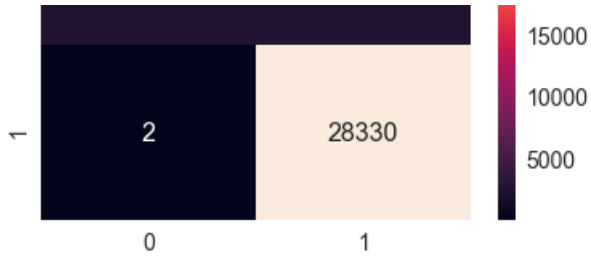
In [167]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_3_rf, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[167]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xdd9c53dfd0>
```





Test Data

In [168]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.794
[[  623  1923]
 [ 1084 12870]]
```

In [169]:

```
conf_matr_df_test_3_rf = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds,
test_fpr, test_fpr)), range(2),range(2))
```

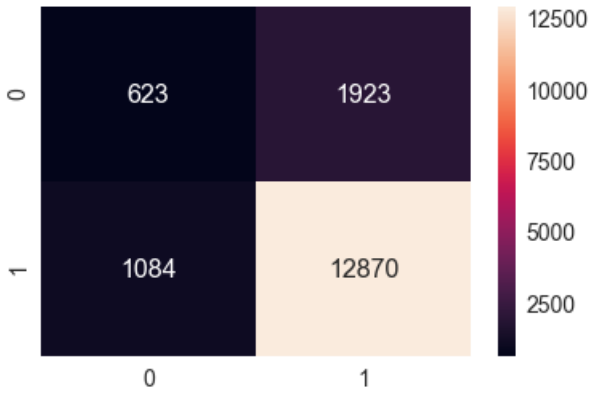
```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.794
```

In [170]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_3_rf, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[170]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xdda07ede48>
```



Set 4 : Categorical, Numerical features + Project_title(TFIDF W2V) + Preprocessed_essay (TFIDF W2V)

In [174]:

```
tfidf_w2v_vectors_train2d = np.array(tfidf_w2v_vectors_train)
tfidf_w2v_vectors_train2d.shape
```

Out[174]:

```
(33500, 300)
```

(33500, 300)

In [175]:

```
tfidf_w2v_vectors_titles_train2d = np.array(tfidf_w2v_vectors_titles_train)
tfidf_w2v_vectors_titles_train2d.shape
```

Out[175]:

(33500, 300)

In [176]:

```
tfidf_w2v_vectors_test2d = np.array(tfidf_w2v_vectors_test)
tfidf_w2v_vectors_test2d.shape
```

Out[176]:

(16500, 300)

In [177]:

```
tfidf_w2v_vectors_titles_test2d = np.array(tfidf_w2v_vectors_titles_test)
tfidf_w2v_vectors_titles_test2d.shape
```

Out[177]:

(16500, 300)

In [179]:

```
print(train_clean_categories.shape)
print(train_clean_subcategories.shape)
print( train_school_state.shape)
print(train_project_grade_category.shape)
print(train_teacher_prefix.shape)
print(price_train.shape)
print(quantity_train.shape)
print( prev_projects_train.shape)
print(title_word_count_train.shape)
print(essay_word_count_train.shape)
print(essay_sent_pos_train.shape)
print(essay_sent_neg_train.shape)
print(essay_sent_neu_train.shape)
print( essay_sent_comp_train.shape)
print(tfidf_w2v_vectors_titles_train2d.shape)
print(tfidf_w2v_vectors_train2d.shape)
```

```
(33500, 2)
(33500, 2)
(33500, 2)
(33500, 2)
(33500, 2)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 300)
(33500, 300)
```

In [180]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = np.hstack((train_clean_categories, train_clean_subcategories, train_school_state,
train_project_grade_category, train_teacher_prefix,price_train, quantity_train,
```

```
prev_projects_train, title_word_count_train, essay_word_count_train, essay_sent_pos_train,
essay_sent_neg_train, essay_sent_neu_train, essay_sent_comp_train, tfidf_w2v_vectors_titles_train,
tfidf_w2v_vectors_train))
X_te = np.hstack((test_clean_categories, test_clean_subcategories, test_school_state,
test_project_grade_category, test_teacher_prefix,price_test, quantity_test, prev_projects_test,
title_word_count_test, essay_word_count_test, essay_sent_pos_test, essay_sent_neg_test, essay_sent_
neu_test, essay_sent_comp_test, tfidf_w2v_vectors_titles_test, tfidf_w2v_vectors_test))
```

In [183]:

```
print(X_tr.shape)
print(X_te.shape)
```

```
(33500, 619)
(16500, 619)
```

A) GridSearchCV (K fold Cross Validation)

Applying Random Forest

In [185]:

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()

parameters = {'n_estimators': [10, 100, 500, 1000], 'max_depth':[10, 50, 100, 500, 1000]}

clf = RandomizedSearchCV(rf, parameters, cv= 3, scoring='roc_auc',n_jobs=-1)

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']
print(clf.best_params_)
```

```
{'n_estimators': 1000, 'max_depth': 10}
```

In [186]:

```
train_auc
```

Out[186]:

```
array([1.        , 1.        , 0.9704975 , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 0.97794206])
```

In [187]:

```
cv_auc
```

Out[187]:

```
array([0.62280622, 0.62429227, 0.6603915 , 0.62343682, 0.62443708,
       0.62352321, 0.62209853, 0.62454456, 0.61561862, 0.66963468])
```

Plot for Train & Cross Validation Data

In [188]:

```
x1 = [1.        , 1.        , 0.9704975 , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 0.97794206]
```

In [189]:

```
x2 = [0.62280622, 0.62429227, 0.6603915 , 0.62343682, 0.62443708,
       0.62352321, 0.62209853, 0.62454456, 0.61561862, 0.66963468]
```

In [190]:

```
y1 = pd.Series([10,100,500,1000,10,100,500,1000,5,10], index = x1)
```

In [191]:

```
z1 = pd.Series([10,10,50,50,100,100,500,500,1000,1000],index = x1)
```

In [192]:

```
trace1 = go.Scatter3d(
    x=x1, y=y1, z=z1,
    name = 'Train',
    marker=dict(
        size=4,
        colorscale='Viridis',
    ),
    line=dict(
        color='#1f77b4',
        width=1
    )
)

trace2 = go.Scatter3d(
    x=x2, y=y1, z=z1,
    name = 'Test',
    marker=dict(
        size=4,
        colorscale='Viridis',
    ),
    line=dict(
        color='#b45c1f',
        width=1
    )
)
```

In [193]:

```
data = [trace1, trace2]
```

In [194]:

```
layout = dict(
    width=800,
    height=700,
    autosize=False,
    title='Hyper Parameter Tuning -- Random Forests - TFIDF W2V',
    scene=dict(
        xaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        yaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        zaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        camera=dict(
```

```
            up=dict(
                x=0,
                y=0,
                z=1
            ),
            eye=dict(
                x=-1.7428,
                y=1.0707,
                z=0.7100,
            )
        ),
        aspectratio = dict( x=1, y=1, z=0.7 ),
        aspectmode = 'manual'
    ),
)
```

In [196]:

```
import plotly
plotly.tools.set_credentials_file(username='Subham27091995', api_key='ltlSgXuO5vzBGyXwFMyv')
```

In [197]:

```
fig = dict(data=data, layout=layout)

py.iplot(fig, filename='Random-Forests-g', height=700)
```

Out[197]:

Observations :

1) Number of estimators as 1000 performs decently on both Train as well as Cross Validation Data.

2) 10 as the value for maximum depth is considered.

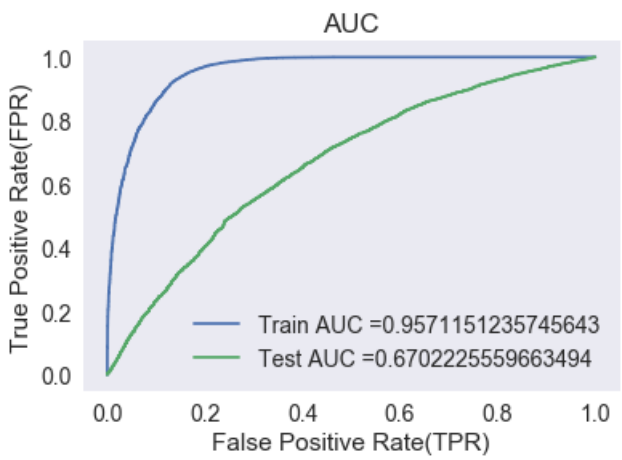
B) Train the model using the best hyper parameter value

In [198]:

```
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 [199]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

model = RandomForestClassifier(max_depth = 10, n_estimators = 1000)

model.fit(X_tr, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(model, X_tr)
y_test_pred = batch_predict(model, 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("False Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



C) Confusion Matrix

In [335]:

```
def predict(proba, threshould, fpr, tpr):

    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)
    return predictions
```

Train Data

In [336]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
('the maximum value of tpr*(1-fpr)', 0.23755274534005302, 'for threshold', 0.852)
[[ 4305  6778]
 [20813 41300]]
```

In [337]:

```
conf_matr_df_train_4_rf = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred,
tr_thresholds, train_fpr, train_fpr)), range(2),range(2))
```

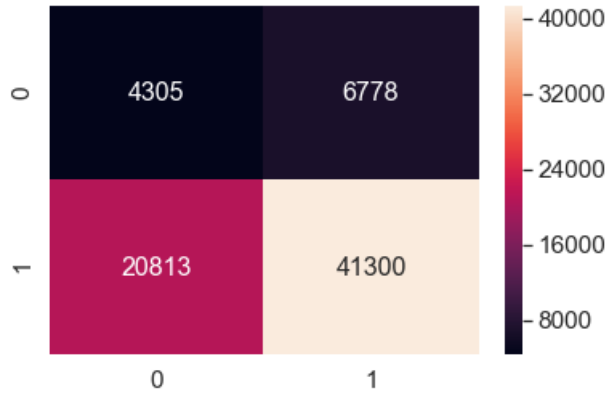
```
('the maximum value of tpr*(1-fpr)', 0.23755274534005302, 'for threshold', 0.852)
```

In [340]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_4_rf, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[340]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a8e2dac90>
```



Test Data

In [338]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
('the maximum value of tpr*(1-fpr)', 0.23880877084654542, 'for threshold', 0.944)
[[ 5457     2]
 [30582    11]]
```

In [339]:

```
conf_matr_df_test_4_rf = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds,
test_fpr, test_fpr)), range(2),range(2))
```

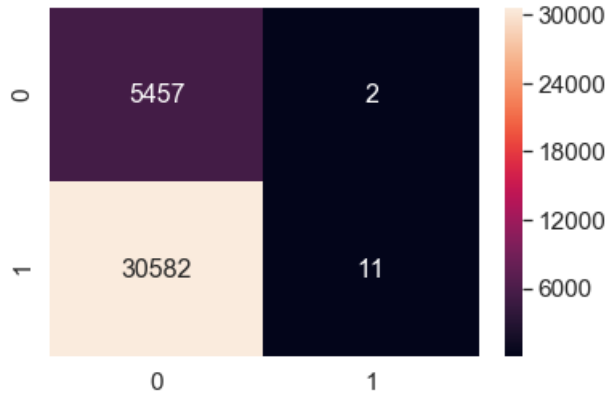
```
('the maximum value of tpr*(1-fpr)', 0.23880877084654542, 'for threshold', 0.944)
```

In [341]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_4_rf, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[341]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a8de9acd0>
```



Applying GBDT

In [201]:

```
from sklearn.ensemble import GradientBoostingClassifier
gbdt = GradientBoostingClassifier()

parameters = {'n_estimators': [10, 100, 500, 1000], 'max_depth':[10, 50, 100, 500, 1000]}

clf = RandomizedSearchCV(gbdt, parameters, cv= 3, scoring='roc_auc',n_jobs=-1)

clf.fit(X_tr, y_train)
```

Out[201]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
          estimator=GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_sampl          subsample=1.0, tol=0.0001
```

```
validation_fraction=0.1,
              verbose=0, warm_start=False),
          fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'n_estimators': [10, 100, 500, 1000], 'max_depth': [10, 50, 100, 500
, 1000]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring='roc_auc', verbose=0)
```

In [202]:

```
train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']
print(clf.best_params_)
```

```
{'n_estimators': 1000, 'max_depth': 10}
```

In [ ]:

```
train_auc
```

In [204]:

```
cv_auc
```

Out[204]:

```
array([0.57612373, 0.57713129, 0.57860349, 0.57947862, 0.57751162,
       0.66219175, 0.53456406, 0.65832634, 0.5781767 , 0.57628625])
```

Plot for Train & Cross Validation Data

In [205]:

```
x1 = [1.        , 1.        , 0.9704975 , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 0.97794206]
```

In [206]:

```
x2 = [0.57612373, 0.57713129, 0.57860349, 0.57947862, 0.57751162,
       0.66219175, 0.53456406, 0.65832634, 0.5781767 , 0.57628625]
```

In [207]:

```
z1 = pd.Series([10,10,50,50,100,100,500,500,1000,1000],index = x1)
```

In [208]:

```
y1 = pd.Series([10,100,500,1000,10,100,500,1000,5,10], index = x1)
```

In [209]:

```
trace1 = go.Scatter3d(
    x=x1, y=y1, z=z1,
    name = 'Train',
    marker=dict(
        size=4,
        colorscale='Viridis',
    ),
    line=dict(
        color='#1f77b4',
        width=1
    )
)

trace2 = go.Scatter3d(
    x=x2, y=y1, z=z1,
```

```
    name = 'Test',
    marker=dict(
        size=4,
        colorscale='Viridis',
    ),
    line=dict(
        color='#b45c1f',
        width=1
    )
)
```

In [210]:

```
data = [trace1, trace2]
```

In [211]:

```
layout = dict(
    width=800,
    height=700,
    autosize=False,
    title='Hyper Parameter Tuning -- GBDT - TFIDF-W2V',
    scene=dict(
        xaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        yaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        zaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        camera=dict(
            up=dict(
                x=0,
                y=0,
                z=1
            ),
            eye=dict(
                x=-1.7428,
                y=1.0707,
                z=0.7100,
            )
        ),
        aspectratio = dict( x=1, y=1, z=0.7 ),
        aspectmode = 'manual'
    ),
)
```

In [212]:

```
fig = dict(data=data, layout=layout)

py.iplot(fig, filename='GBDT - a', height=700)
```

Out[212]:

Observations :

1) 10 as the value for maximum depth is considered. Shallow trees generally perform well for GBDT.

2) 1000 as number of estimators is considered for training the final model.

B) Train the model using the best hyper parameter value

In [ ]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

model = GradientBoostingClassifier(max_depth = 5, n_estimators = 500)

model.fit(X_tr, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(model, X_tr)
y_test_pred = batch_predict(model, 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("False Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```

C) Confusion Matrix

In [503]:

```
def predict(proba, threshould, fpr, tpr):

    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)
    return predictions
```

Train Data

In [504]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
('the maximum value of tpr*(1-fpr)', 0.2499999979647145, 'for threshold', 0.844)
[[ 5542  5541]
 [10955 51158]]
```

In [505]:

```
conf_matr_df_train_1_gbdt = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred,
tr_thresholds, train_fpr, train_fpr)), range(2),range(2))
```

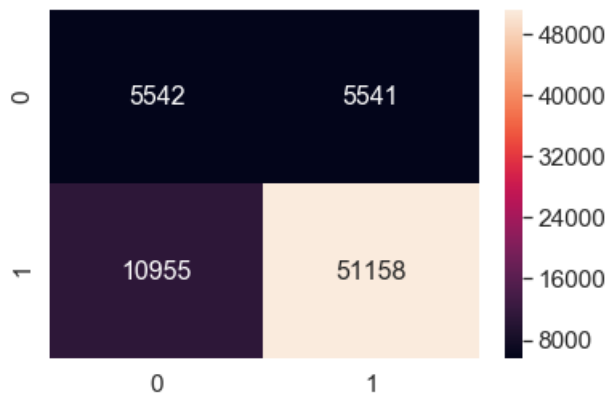
```
('the maximum value of tpr*(1-fpr)', 0.2499999979647145, 'for threshold', 0.844)
```

In [620]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_1_gbdt, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[620]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a42ff9350>
```



Test Data

In [506]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
('the maximum value of tpr*(1-fpr)', 0.24999999161092995, 'for threshold', 0.848)
[[ 3855  1604]
 [13949 16644]]
```

In [507]:

```
conf_matr_df_test_1_gbdt = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds
, test_fpr, test_fpr)), range(2),range(2))
```

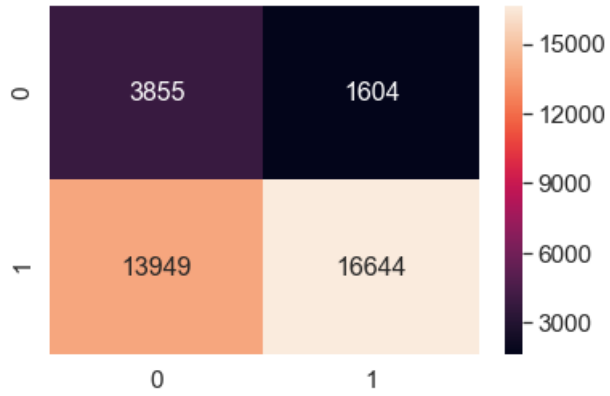
```
('the maximum value of tpr*(1-fpr)', 0.24999999161092995, 'for threshold', 0.848)
```

In [619]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_1_gbdt, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[619]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a43d586d0>
```



5. Conclusion

In [217]:

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

# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable

#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyperparameters(max depth,min samples split)", "Train AUC"
, "Test AUC"]

x.add_row(["BOW", "RF","(10, 1000)", 0.81, 0.68])
x.add_row(["TFIDF", "RF", "(50, 1000)", 0.82, 0.68])
x.add_row(["AVG W2V", "RF", "(50, 500)", 0.87, 0.67])
x.add_row(["TFIDF W2V", "RF", "(10, 1000)", 0.92, 0.68])
```

```
x.add_row(["----- ---", "----", "-----------------------------------------------", "---------", "--
---"])

x.add_row(["BOW", "GBDT","(10, 500)", 0.87, 0.67])
x.add_row(["TFIDF", "GBDT", "(10, 100)", 0.85, 0.65])
x.add_row(["AVG W2V", "GBDT", "(10, 500)", 0.90, 0.67])
x.add_row(["TFIDF W2V", "GBDT", "(10, 1000)", 0.90, 0.68])



print(x)
```

```
+------------+-------+-------------------------------------------------+-----------+----------+
| Vectorizer | Model |   Hyperparameters(max depth,min samples split)  | Train AUC | Test AUC |
+------------+-------+-------------------------------------------------+-----------+----------+
|    BOW     |   RF  |                    (10, 1000)                   |    0.81   |   0.68   |
|   TFIDF    |   RF  |                    (50, 1000)                   |    0.82   |   0.68   |
|  AVG W2V   |   RF  |                    (50, 500)                    |    0.87   |   0.67   |
| TFIDF W2V  |   RF  |                    (10, 1000)                   |    0.92   |   0.68   |
| ----- ---  |  ---- | ----------------------------------------------- | --------- | -------- |
|    BOW     |  GBDT |                    (10, 500)                    |    0.87   |   0.67   |
|   TFIDF    |  GBDT |                    (10, 100)                    |    0.85   |   0.65   |
|  AVG W2V   |  GBDT |                    (10, 500)                    |    0.9    |   0.67   |
| TFIDF W2V  |  GBDT |                    (10, 1000)                   |    0.9    |   0.68   |
+------------+-------+-------------------------------------------------+-----------+----------+
```