

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.

DonorsChoose

About the DonorsChoose Data Set

The `train.csv` data set provided by DonorsChoose contains the following features:

Feature	
<code>project_id</code>	A unique identifier for the proposed project
<code>project_title</code>	Title of the project
<code>project_grade_category</code>	Grade level of students for which the project is targeted
<code>project_art_category</code>	Art category
<code>project_teacher_id</code>	Teacher's unique identifier
<code>project_teacher_title</code>	Teacher's title
<code>project_teacher_years</code>	Teacher's years of experience
<code>project_school_id</code>	School's unique identifier
<code>project_school_latitude</code>	School's latitude
<code>project_school_longitude</code>	School's longitude
<code>project_approved</code>	Whether the project was approved

Feature	Description
id	A <code>project_id</code> value from the <code>train.csv</code> file. Example: p036502
description	Description 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
<code>project_is_approved</code>	A binary flag indicating whether DonorsChoose approved the project. A value of <code>0</code> indicates the project was not approved, and a value of <code>1</code> indicates the project was approved.



Notes on the Essay Data

Prior to May 17, 2016, the prompts for the essays were as follows:

- `__project_essay_1:` "Introduce us to your classroom"
- `__project_essay_2:` "Tell us more about your students"
- `__project_essay_3:` "Describe how your students will use the materials you're requesting"
- `__project_essay_3:` "Close by sharing why your project will make a difference"

Starting on May 17, 2016, the number of essays was reduced from 4 to 2, and the prompts for the first 2 essays were changed to the following:

- `__project_essay_1:` "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."
- `__project_essay_2:` "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

For all projects with `project_submitted_datetime` of 2016-05-17 and later, the values of `project_essay_3` and `project_essay_4` will be NaN.

```
In [4]: %matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import math
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 chart_studio import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter

from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

import dill #To store session variables
#https://stackoverflow.com/questions/34342155/how-to-pickle-or-store-jupyter-ipy
```

1.1 Reading Data

```
In [5]: resource_data = pd.read_csv('resources.csv')
project_data = pd.read_csv('train_data.csv')
```

```
In [6]: project_data_1=project_data[project_data['project_is_approved']==1]
project_data_0=project_data[project_data['project_is_approved']==0]

print(project_data_1.shape)
print(project_data_0.shape)

#Creating a dataset of 0.2k points containg points from both the classes
project_data = project_data_1[0:33458].append(project_data_0[0:16542])
print(project_data['project_is_approved'].value_counts())
print(project_data.shape)
```

```
(92706, 17)
(16542, 17)
1    33458
0    16542
Name: project_is_approved, dtype: int64
(50000, 17)
```

```
In [7]: 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 [8]: # how to replace elements in list python: https://stackoverflow.com/a/2582163/408
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/4970249/408
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/13148611/408
project_data = project_data[cols]

project_data.head(2)
```

Out[8]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-20 00:53:00
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-20 01:10:00

```
In [9]: 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[9]:

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

1.2 preprocessing of project_subject_categories

```

In [10]: categories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.co

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-
cat_list = []
for i in categories:
    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",
        if 'The' in j.split(): # this will split each of the category based on sp
            j=j.replace('The','') # if we have the words "The" we are going to re
        j = j.replace(' ', '') # we are placing all the ' '(space) with ''(empty)
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trail
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

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

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

```

1.3 preprocessing of project_subject_subcategories

```

In [11]: sub_categories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/22898595/4
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-while-keeping-the-words

sub_cat_list = []
for i in sub_categories:
    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 & Hunger"]
        if 'The' in j.split(): # this will split each of the category based on space
            j=j.replace('The', '') # if we have the words "The" we are going to remove them
        j = j.replace(' ', '') # we are placing all the ' ' (space) with '' (empty string)
        temp +=j.strip()+" #" "abc ".strip() will return "abc", remove the trailing space
    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/22898595/4
my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

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

```

1.3 Text preprocessing

```

In [12]: # merge two column text dataframe:
project_data["essay"] = project_data["project_essay_1"].map(str) + \
    project_data["project_essay_2"].map(str) + \
    project_data["project_essay_3"].map(str) + \
    project_data["project_essay_4"].map(str)

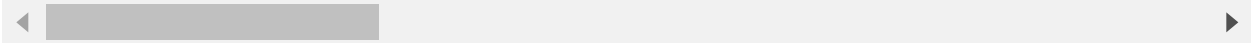
```



```
In [13]: project_data.head(2)
```

Out[13]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-20 00:53:00
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-20 01:10:00



```
In [14]: # printing some random reviews
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
```

60302it [00:19, 8079.97it/s]

I recently read an article about giving students a choice about how they learn. We already set goals; why not let them choose where to sit, and give them options of what to sit on? I teach at a low-income (Title 1) school. Every year, I have a class with a range of abilities, yet they are all the same age. They learn differently, and they have different interests. Some have ADHD, and some are fast learners. Yet they are eager and active learners that want and need to be able to move around the room, yet have a place that they can be comfortable to complete their work. We need a classroom rug that we can use as a class for reading time, and students can use during other learning times. I have also requested four Kore Kids wobble chairs and four Back Jack padded portable chairs so that students can still move during whole group lessons without disrupting the class. Having these areas will provide these little ones with a way to wiggle while working. Benjamin Franklin once said, "Tell me and I forget, teach me and I may remember, involve me and I learn." I want these children to be involved in their learning by having a choice on where to sit and how to learn, all by giving them options for comfortable flexible seating.

=====

A unit that has captivated my students and one that has forced them to seek out further resources on their own, is the Holocaust unit. This unit not only brought their critical thinking skills to life, but it brought out their passion, love, dislikes, and fears about wars and prejudices to light. My 8th graders students live in a high-poverty school district and live in a large, urban area. They are reluctant readers unless introduced to life-changing books. This book made my students work hard in improving their reading and writing skills. The Holocaust unit brought compassion and history to life. The students wanted to read ahead and learn about tolerance and discrimination. These materials will be used in-class. We were read, discuss, and think critically about the world event that still affects us. The Holocaust is part of our history and its victims and survivors deserve our knowledge and recognition of the hardships they endured. We will be researching the victims and survivors of the Holocaust, read non-fictional text, watch documentaries, and overall broaden our education on this historic event. This project will greatly benefit my students. It will not only help them academically and help prepare them for high school, but it will make them well-rounded individuals who better understand the power of tolerance and war. Please know that you have made a positive impact on my students and we sincerely thank you in advance.

=====

Why learn coding in the 5th grade? I teach science through STEM. Instead of using only spaghetti and marshmallows for engineering, I want the students to use coding. It is time to use interactive approaches to solving problems and testing ideas using real-life skills students may use in the future. My school is located in Jupiter, Florida, and we are an intermediate center, servicing only 3rd-5th grades. I teach 3 classes of science to 5th grade students. My students are a mix of gifted and advanced 10 and 11 year olds, of at which 20% have some type of learning challenge, such as ADD or autism. They all have insatiable thirsts for science. Most come to me with limited knowledge of science, but a tremendous understanding of technology. Most have a computer in their home and are familiar with tablets and smartphones. At least 1/3 of my students know Scratch and

d JavaScript programming.\r\nMy goal is to pair my students incredible knowledge of technology with science concepts to deepen their understandings of that concept. I also want to expose all of my students with coding since research has shown that more computer coders will be needed for future jobs than ever before.\r\nWhat I envision is the students working in groups using the specific coding device, Raspberry Pi, to create codes to manipulate the sensors. These will be attached to laptops at each table. In the beginning, I will use the device to teach basic coding to solve a problem. The students will be required to learn how to set up the motherboard during this process. Then I will move on to using it with my science content. One activity I found intriguing is the weather station sensors. The students work together to find a way to code for each of these sensors to turn on and off and collect, store, and manipulate the data. This will become a part of my weather unit. By pairing this type of technology with science, I feel my lesson then is reflecting how science works in the real world. Technology and science go hand in hand and I want my students to experience that one influences the other. I want them to experience that scientists use technology as a tool to further deepen their understanding of concepts. I also want both my boys and girls to learn and understanding coding as a viable future career.

In [15]: [# https://stackoverflow.com/a/47091490/4084039](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 [16]: sent = decontracted(project_data['essay'].values[2000])
print(sent)
print("="*50)
```

My school is in a low socio-economic area with a high ELL population. The students in my classroom do not have a lot of academic practice outside of the school day. They love coming to school everyday and are eager to learn. They work very hard and are so excited when they master new concepts. \r\n At my school site we strive to make the most of every minute during the school day in order to ensure students are able to learn and feel successful. We know that the time we have with them is very precious! I am asking for the mini white boards and reusable write and wipe pockets in order to help me monitor my students thinking and learning. Often times, when work is done on worksheets the feedback to students is not meaningful because it can take awhile to give each student individual feedback. The white boards and write and wipe pockets will give students a way to show written responses while we are gathered at the carpet together. This will allow me to give immediate feedback to students and then can modify their responses right then and there. This will lead to more meaningful learning and processing.nannan

=====

```
In [17]: # \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks
sent = sent.replace('\n', ' ')
sent = sent.replace('\r', ' ')
sent = sent.replace('\t', ' ')
print(sent)
```

My school is in a low socio-economic area with a high ELL population. The students in my classroom do not have a lot of academic practice outside of the school day. They love coming to school everyday and are eager to learn. They work very hard and are so excited when they master new concepts. At my school site we strive to make the most of every minute during the school day in order to ensure students are able to learn and feel successful. We know that the time we have with them is very precious! I am asking for the mini white boards and reusable write and wipe pockets in order to help me monitor my students thinking and learning. Often times, when work is done on worksheets the feedback to students is not meaningful because it can take awhile to give each student individual feedback. The white boards and write and wipe pockets will give students a way to show written responses while we are gathered at the carpet together. This will allow me to give immediate feedback to students and then can modify their responses right then and there. This will lead to more meaningful learning and processing.nannan

```
In [18]: #remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

My school is in a low socio economic area with a high ELL population The students in my classroom do not have a lot of academic practice outside of the school day They love coming to school everyday and are eager to learn They work very hard and are so excited when they master new concepts At my school site we strive to make the most of every minute during the school day in order to ensure students are able to learn and feel successful We know that the time we have with them is very precious I am asking for the mini white boards and reusable write and wipe pockets in order to help me monitor my students thinking and learning Often times when work is done on worksheets the feedback to students is not meaningful because it can take awhile to give each student individual feedback The white boards and write and wipe pockets will give students a way to show written responses while we are gathered at the carpet together This will allow me to give immediate feedback to students and then can modify their responses right then and there This will lead to more meaningful learning and processing

```
In [19]: # 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'll', 'you'd', 'your', 'yours', 'yourself', 'yourselves', 'he', 'she', 'she's", 'her', 'hers', 'herself', 'it', 'it's", 'its', 'itself', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'till', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 's', 't', 'can', 'will', 'just', 'don', 'don't", 'should', "should've", 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'won', "won't", 'wouldn', "wouldn't"]
```



```
In [21]: #adding a new column for the processed essay text
project_data['clean_essay']=preprocessed_essays
print(project_data.columns)

# after preprocessing
preprocessed_essays[2000]
```

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

```
Out[21]: 'school low socio economic area high ell population students classroom not lot
academic practice outside school day love coming school everyday eager learn wo
rk hard excited master new concepts school site strive make every minute school
day order ensure students able learn feel successful know time precious asking
mini white boards reusable write wipe pockets order help monitor students think
ing learning often times work done worksheets feedback students not meaningful
take awhile give student individual feed back white boards write wipe pockets g
ive students way show written responses gathered carpet together allow give imm
ediate feedback students modify responses right lead meaningful learning proces
sing nannan'
```

1.4.1 Preprocessing of `project_title`

```
In [22]: project_data.head(2)
```

```
Out[22]:
```

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-20 00:53:00
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-20 01:10:00

In [23]: *#Printing a few random review summaries*

```
for i in range(1,3000,1000):  
    sent = project_data['project_title'].values[i]  
    print(sent, '--- Row No:', i)  
    print("="*50)
```

Breakout Box to Ignite Engagement! --- Row No: 1

=====

Cozy Classroom Carpet for Learning --- Row No: 1001

=====

Community Circle Carpet: A Place to Call Home! --- Row No: 2001

=====


```
In [24]: # The above random records show that there are no URLs or HTML tags, but we will
```

```
from tqdm import tqdm #for status bar
from bs4 import BeautifulSoup #for html tags

preprocessed_title=[]

for title in tqdm(project_data['project_title'].values):
    # To remove urls - https://stackoverflow.com/a/40823105/4084039
    title = re.sub(r"http\S+", "", title)

    # To remove all HTML tags
    #https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-rem
    title = BeautifulSoup(title, 'lxml').get_text()

    # To split contractions - refer decontracted function defined above
    title = decontracted(title)

    # To remove alphanumerics (words with numbers in them) - https://stackoverflowfl
    title = re.sub("\S*\d\S*", "", title).strip()

    # To remove special characters - https://stackoverflow.com/a/5843547/4084039
    title = re.sub('[^A-Za-z]+', ' ', title)

    # To remove stop words from the summaries and convert to lowercase
    title = ' '.join(e.lower() for e in title.split() if e.lower() not in stopwords)
    preprocessed_title.append(title.strip())

#adding a new column for cleaned titles
project_data['clean_title']=preprocessed_title
print(project_data.columns)
```

In [25]:

```
#replacing Nan values with 'Unknown'
project_data['teacher_prefix']=project_data['teacher_prefix'].replace(np.nan, 'Unl
```

1.4.3 Combining resource_data with project_data

In [26]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).l
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

1.4.4 Adding word counts for Title and Essay

In [27]:

```
#https://stackoverflow.com/questions/54397096/how-to-do-word-count-on-pandas-data

project_data['title_wc'] = project_data['clean_title'].str.count(' ')+1

project_data['essay_wc'] = project_data['clean_essay'].str.count(' ')+1

project_data.columns
```

```
Out[27]: Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
               'Date', 'project_grade_category', 'project_title', 'project_essay_1',
               'project_essay_2', 'project_essay_3', 'project_essay_4',
               'project_resource_summary',
               'teacher_number_of_previously_posted_projects', 'project_is_approved',
               'clean_categories', 'clean_subcategories', 'essay', 'clean_essay',
               'clean_title', 'price', 'quantity', 'title_wc', 'essay_wc'],
              dtype='object')
```

1.4.5 Adding sentiment scores for each essay

In [28]: <http://t-redactyl.io/blog/2017/04/using-vader-to-handle-sentiment-analysis-with>

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

nltk.download('vader_lexicon')

project_data['senti_score'] = 0
project_data['senti_score'] = project_data['senti_score'].astype(float)

anlyzr = SentimentIntensityAnalyzer()

for index in project_data.index:
    project_data.at[index, 'senti_score'] = anlyzr.polarity_scores(project_data.at[index, 'essay_1'])

print(project_data.columns)
```

[nltk_data] Downloading package vader_lexicon to

[nltk_data] C:\Users\vinodhkumarb\AppData\Roaming\nltk_data...

[nltk_data] Package vader_lexicon is already up-to-date!

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
      'Date', 'project_grade_category', 'project_title', 'project_essay_1',
      'project_essay_2', 'project_essay_3', 'project_essay_4',
      'project_resource_summary',
      'teacher_number_of_previously_posted_projects', 'project_is_approved',
      'clean_categories', 'clean_subcategories', 'essay', 'clean_essay',
      'clean_title', 'price', 'quantity', 'title_wc', 'essay_wc',
      'senti_score'],
      dtype='object')
```

```
In [29]: import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
analyzer = SentimentIntensityAnalyzer()
neg=[];pos=[];neu=[]; compound = []

for i in tqdm(range(len(project_data['clean_essay']))):
    sentiment_scores = analyzer.polarity_scores(project_data['clean_essay'][i])
    neg.append(sentiment_scores['neg'])
    pos.append(sentiment_scores['pos'])
    neu.append(sentiment_scores['neu'])
    compound.append(sentiment_scores['compound'])
```

```
0%|
| 0/50000 [00:00<?, ?it/s]
0%|
| 51/50000 [00:00<01:39, 501.35it/s]
0%||
| 107/50000 [00:00<01:36, 515.38it/s]
0%||
| 159/50000 [00:00<01:36, 515.62it/s]
0%||
| 214/50000 [00:00<01:35, 521.41it/s]
1%||
| 267/50000 [00:00<01:35, 521.29it/s]
1%||
| 322/50000 [00:00<01:34, 526.95it/s]
1%||
| 370/50000 [00:00<01:37, 509.10it/s]
1%||
| 424/50000 [00:00<01:36, 515.44it/s]
1%||
```

```
In [30]: #new columns indicating the sentiment score of each project essay
project_data['neg'] = neg
project_data['neu'] = neu
project_data['pos'] = pos
project_data['compound'] = compound
```

1.5 Preparing data for models

```
In [31]: project_data.columns
```

```
Out[31]: Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
               'Date', 'project_grade_category', 'project_title', 'project_essay_1',
               'project_essay_2', 'project_essay_3', 'project_essay_4',
               'project_resource_summary',
               'teacher_number_of_previously_posted_projects', 'project_is_approved',
               'clean_categories', 'clean_subcategories', 'essay', 'clean_essay',
               'clean_title', 'price', 'quantity', 'title_wc', 'essay_wc',
               'senti_score', 'neg', 'neu', 'pos', '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

```
In [32]: project_data.drop(labels='senti_score',inplace=True,axis=1)
```

2.GBDT

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

```
In [33]: from sklearn.model_selection import train_test_split

import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)

#Checking if there are any values other than 0 and 1
project_data['project_is_approved'].unique()

#https://answers.dataiku.com/2352/split-dataset-by-stratified-sampling
df_train, df_test = train_test_split(project_data, test_size = 0.3, stratify=pro
print(df_train.shape,df_test.shape)
```

```
(35000, 28) (15000, 28)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

2.2.1 Vectorizing Categorical data using class probabilities (Response Coding)

```
In [34]: print(df_train.columns)
```

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

2.2.1.1 Feature encoding for categories

In [35]: [#https://stackoverflow.com/questions/3839729/count-unique-values-with-pandas-per](https://stackoverflow.com/questions/3839729/count-unique-values-with-pandas-per)

```
# Fetching unique value counts for each class
clean_cat_count = pd.DataFrame()
clean_cat_count[1] = df_train['clean_categories'].where(df_train['project_is_app
clean_cat_count[0] = df_train['clean_categories'].where(df_train['project_is_app

#Replacing nan value counts with zeros
clean_cat_count[1]=clean_cat_count[1].replace(np.nan,0)
clean_cat_count[0]=clean_cat_count[0].replace(np.nan,0)

#print(clean_cat_count)

#Calculating probs for each class
for i in clean_cat_count.iterrows():
    clean_cat_count['1_prob'] = clean_cat_count[1]/(clean_cat_count[1]+clean_cat_co
    clean_cat_count['0_prob'] = clean_cat_count[0]/(clean_cat_count[1]+clean_cat_co

#print(clean_cat_count)

#appending prob values to train data in a new column

for idx,j in clean_cat_count.iterrows():
    for idx,i in df_train.iterrows():
        if idx == df_train.at[idx, 'clean_categories']:
            df_train.at[idx, 'cat_1'] = clean_cat_count.at[idx, '1_prob']
            df_train.at[idx, 'cat_0'] = clean_cat_count.at[idx, '0_prob']

print(df_train.head(2))
```

	Unnamed: 0	id	teacher_id	teacher_prefix
21523	65163	p166386	211ee18bf18a4d132284f2de565ef18a	Mrs.
4975	102990	p194349	406518edf1c30d599de90e2055a697fc	Mrs.

	school_state	Date	project_grade_category
21523	KY	2016-09-09 17:28:19	Grades PreK-2
4975	GA	2016-06-26 22:40:41	Grades 3-5

	project_title
21523	Wiggle While You Work!
4975	Working in Small Groups Is a Big Deal

	project_essay_1
21523	As the librarian, I am privileged to work with...
4975	My students come from diverse backgrounds, and...

	project_essay_2	price
21523	My Project\r\nMy K-5 students are active, enth...	96.01
4975	My students will use this carpet each day duri...	546.08

	quantity	title_wc	essay_wc	neg	neu	pos	compound	cat_1
21523	8	2	113	0.043	0.581	0.376	0.9927	0.656489
4975	7	5	240	0.098	0.564	0.338	0.9973	0.651769

	cat_0
--	-------

```
21523  0.343511
4975   0.348231
```

```
[2 rows x 30 columns]
```



```
In [36]: df_train.isna().any()
```

```
Out[36]: Unnamed: 0      False
id          False
teacher_id  False
teacher_prefix False
school_state False
Date        False
project_grade_category False
project_title False
project_essay_1 False
project_essay_2 False
project_essay_3    True
project_essay_4    True
project_resource_summary False
teacher_number_of_previously_posted_projects False
project_is_approved False
clean_categories  False
clean_subcategories False
essay            False
clean_essay      False
clean_title      False
price           False
quantity        False
title_wc        False
essay_wc        False
neg            False
neu            False
pos            False
compound        False
cat_1          True
cat_0          True
dtype: bool
```



```
In [37]: #appending prob values to test data in a new column. Incase the class is not part
for idx,j in clean_cat_count.iterrows():
    for indx,i in df_test.iterrows():
        if idx == df_test.at[indx, 'clean_categories']:
            df_test.at[indx, 'cat_1'] = clean_cat_count.at[idx, '1_prob']
            df_test.at[indx, 'cat_0'] = clean_cat_count.at[idx, '0_prob']

df_test['cat_1']=df_test['cat_0'].replace(np.nan,0.5)
df_test['cat_0']=df_test['cat_0'].replace(np.nan,0.5)

print(df_test.head(2))
```

```

      Unnamed: 0      id      teacher_id teacher_prefix \
33973      112656 p208177 3fe3dd300b49b1bcb20c5ce7c770c9c3      Ms.
27635      131772 p217130 931402be87fef92c660e68d12cc108d9      Ms.

      school_state      Date project_grade_category \
33973      LA 2016-12-07 15:03:46      Grades 3-5
27635      TX 2016-10-16 21:20:21      Grades PreK-2

      project_title \
33973      Silence is Golden!
27635  Robotics in the classroom. Bring on the iPads!

      project_essay_1 \
33973  My school is an at risk school located in the ...
27635  As a teacher in a low-income/high poverty scho...

      project_essay_2 ...      price \
33973  Silence is golden! My students just received ...      134.90
27635  My students' lives will be changed, because th...      1226.45

      quantity title_wc  essay_wc      neg      neu      pos compound      cat_1 \
33973      3      2      118  0.076  0.579  0.345  0.9917  0.327586
27635      3      4      140  0.031  0.660  0.309  0.9911  0.382998

      cat_0
33973  0.327586
27635  0.382998

[2 rows x 30 columns]
```

```
In [38]: df_train['cat_1']=df_train['cat_1'].replace(np.nan,0.5)
df_train['cat_0']=df_train['cat_0'].replace(np.nan,0.5)
```

2.2.1.2 Feature encoding for subcategories

In [39]: [#https://stackoverflow.com/questions/3839729/count-unique-values-with-pandas-per](https://stackoverflow.com/questions/3839729/count-unique-values-with-pandas-per)

```
# Fetching unique value counts for each class
clean_subcat_count = pd.DataFrame()
clean_subcat_count[1] = df_train['clean_subcategories'].where(df_train['project_
clean_subcat_count[0] = df_train['clean_subcategories'].where(df_train['project_

#Replacing nan value counts with zeros
clean_subcat_count[1]=clean_subcat_count[1].replace(np.nan,0)
clean_subcat_count[0]=clean_subcat_count[0].replace(np.nan,0)

#print(clean_subcat_count)

#Calculating probs for each class
for i in clean_subcat_count.iterrows():
    clean_subcat_count['1_prob'] = clean_subcat_count[1]/(clean_subcat_count[1]+cle
    clean_subcat_count['0_prob'] = clean_subcat_count[0]/(clean_subcat_count[1]+cle

#print(clean_subcat_count)

#appending prob values to train data in a new column

for idx,j in clean_subcat_count.iterrows():
    for idx,i in df_train.iterrows():
        if idx == df_train.at[idx, 'clean_subcategories']:
            df_train.at[idx, 'subcat_1'] = clean_subcat_count.at[idx, '1_prob']
            df_train.at[idx, 'subcat_0'] = clean_subcat_count.at[idx, '0_prob']

print(df_test.head(2))
```

	Unnamed: 0	id	teacher_id	teacher_prefix	\
33973	112656	p208177	3fe3dd300b49b1bcb20c5ce7c770c9c3	Ms.	
27635	131772	p217130	931402be87fef92c660e68d12cc108d9	Ms.	

	school_state	Date	project_grade_category	\
33973	LA	2016-12-07 15:03:46	Grades 3-5	
27635	TX	2016-10-16 21:20:21	Grades PreK-2	

	project_title	\
33973	Silence is Golden!	
27635	Robotics in the classroom. Bring on the iPads!	

	project_essay_1	\
33973	My school is an at risk school located in the ...	
27635	As a teacher in a low-income/high poverty scho...	

	project_essay_2	...	price	\
33973	Silence is golden! My students just received	134.90	
27635	My students' lives will be changed, because th...	...	1226.45	

	quantity	title_wc	essay_wc	neg	neu	pos	compound	cat_1	\
33973	3	2	118	0.076	0.579	0.345	0.9917	0.327586	
27635	3	4	140	0.031	0.660	0.309	0.9911	0.382998	

	cat_0	
33973	0.327586	

27635 0.382998

[2 rows x 30 columns]

```
In [40]: #appending prob values to test data in a new column. Incase the class is not par
for idx,j in clean_subcat_count.iterrows():
    for idx,i in df_test.iterrows():
        if idx == df_test.at[idx, 'clean_subcategories']:
            df_test.at[idx, 'subcat_1'] = clean_subcat_count.at[idx, '1_prob']
            df_test.at[idx, 'subcat_0'] = clean_subcat_count.at[idx, '0_prob']

df_test['subcat_1']=df_test['subcat_1'].replace(np.nan,0.5)
df_test['subcat_0']=df_test['subcat_0'].replace(np.nan,0.5)

print(df_test.head(2))
```

	Unnamed: 0	id	teacher_id	teacher_prefix	
33973	112656	p208177	3fe3dd300b49b1bcb20c5ce7c770c9c3	Ms.	
27635	131772	p217130	931402be87fef92c660e68d12cc108d9	Ms.	

	school_state	Date	project_grade_category	
33973	LA	2016-12-07 15:03:46	Grades 3-5	
27635	TX	2016-10-16 21:20:21	Grades PreK-2	

	project_title	
33973	Silence is Golden!	
27635	Robotics in the classroom. Bring on the iPads!	

	project_essay_1	
33973	My school is an at risk school located in the ...	
27635	As a teacher in a low-income/high poverty scho...	

	project_essay_2	... title_wc	
33973	Silence is golden! My students just received ...	2	
27635	My students' lives will be changed, because th...	4	

	essay_wc	neg	neu	pos	compound	cat_1	cat_0	subcat_1	
33973	118	0.076	0.579	0.345	0.9917	0.327586	0.327586	0.673077	
27635	140	0.031	0.660	0.309	0.9911	0.382998	0.382998	0.604038	

	subcat_0	
33973	0.326923	
27635	0.395962	

[2 rows x 32 columns]

```
In [41]: df_train['subcat_1']=df_train['subcat_1'].replace(np.nan,0.5)
df_train['subcat_0']=df_train['subcat_0'].replace(np.nan,0.5)
```

2.2.1.3 Feature encoding for state

In [42]: [#https://stackoverflow.com/questions/3839729/count-unique-values-with-pandas-per](https://stackoverflow.com/questions/3839729/count-unique-values-with-pandas-per)

```
# Fetching unique value counts for each class
state_count = pd.DataFrame()
state_count[1] = df_train['school_state'].where(df_train['project_is_approved']!=0)
state_count[0] = df_train['school_state'].where(df_train['project_is_approved']==0)

#Replacing nan value counts with zeros
state_count[1]=state_count[1].replace(np.nan,0)
state_count[0]=state_count[0].replace(np.nan,0)

#print(state_count)

#Calculating probs for each class
for i in state_count.iterrows():
    state_count['1_prob'] = state_count[1]/(state_count[1]+state_count[0])
    state_count['0_prob'] = state_count[0]/(state_count[1]+state_count[0])

#print(state_count)

#appending prob values to train data in a new column

for idx,j in state_count.iterrows():
    for indx,i in df_train.iterrows():
        if idx == df_train.at[indx, 'school_state']:
            df_train.at[indx, 'state_1'] = state_count.at[idx, '1_prob']
            df_train.at[indx, 'state_0'] = state_count.at[idx, '0_prob']

print(df_test.head(2))
```

	Unnamed: 0	id	teacher_id	teacher_prefix
\				
33973	112656	p208177	3fe3dd300b49b1bcb20c5ce7c770c9c3	Ms.
27635	131772	p217130	931402be87fef92c660e68d12cc108d9	Ms.

	school_state	Date	project_grade_category	\
33973	LA	2016-12-07 15:03:46	Grades 3-5	
27635	TX	2016-10-16 21:20:21	Grades PreK-2	

	project_title	\
33973	Silence is Golden!	
27635	Robotics in the classroom. Bring on the iPads!	

	project_essay_1	\
33973	My school is an at risk school located in the ...	
27635	As a teacher in a low-income/high poverty scho...	

	project_essay_2	... title_wc	\
33973	Silence is golden! My students just received ...	2	
27635	My students' lives will be changed, because th...	4	

	essay_wc	neg	neu	pos	compound	cat_1	cat_0	subcat_1
\								
33973	118	0.076	0.579	0.345	0.9917	0.327586	0.327586	0.673077
27635	140	0.031	0.660	0.309	0.9911	0.382998	0.382998	0.604038

```

subcat_0
33973  0.326923
27635  0.395962

```

[2 rows x 32 columns]

```

In [43]: #appending prob values to test data in a new column. Incase the class is not par
for idx,j in state_count.iterrows():
    for indx,i in df_test.iterrows():
        if idx == df_test.at[indx, 'school_state']:
            df_test.at[indx, 'state_1'] = state_count.at[idx, '1_prob']
            df_test.at[indx, 'state_0'] = state_count.at[idx, '0_prob']

df_test['state_1']=df_test['state_1'].replace(np.nan,0.5)
df_test['state_0']=df_test['state_0'].replace(np.nan,0.5)

print(df_test.head(2))

```

```

      Unnamed: 0      id      teacher_id teacher_prefix \
33973      112656 p208177  3fe3dd300b49b1bcb20c5ce7c770c9c3      Ms.
27635      131772 p217130  931402be87fef92c660e68d12cc108d9      Ms.

      school_state      Date project_grade_category \
33973      LA 2016-12-07 15:03:46      Grades 3-5
27635      TX 2016-10-16 21:20:21      Grades PreK-2

      project_title \
33973      Silence is Golden!
27635  Robotics in the classroom. Bring on the iPads!

      project_essay_1 \
33973  My school is an at risk school located in the ...
27635  As a teacher in a low-income/high poverty scho...

      project_essay_2 ...      neg      neu \
33973  Silence is golden! My students just received ... ... 0.076 0.579
27635  My students' lives will be changed, because th... ... 0.031 0.660

      pos      compound      cat_1      cat_0      subcat_1      subcat_0      state_1 \
33973  0.345      0.9917  0.327586  0.327586  0.673077  0.326923  0.638217
27635  0.309      0.9911  0.382998  0.382998  0.604038  0.395962  0.615479

      state_0
33973  0.361783
27635  0.384521

```

[2 rows x 34 columns]

2.2.1.4 Feature encoding for teacher_prefix

```
In [44]: #https://stackoverflow.com/questions/3839729/count-unique-values-with-pandas-per

# Fetching unique value counts for each class
teacherprefix_count = pd.DataFrame()
teacherprefix_count[1] = df_train['teacher_prefix'].where(df_train['project_is_approved'], 1)
teacherprefix_count[0] = df_train['teacher_prefix'].where(df_train['project_is_approved'] == 0, 0)

#Replacing nan value counts with zeros
teacherprefix_count[1]=teacherprefix_count[1].replace(np.nan,0)
teacherprefix_count[0]=teacherprefix_count[0].replace(np.nan,0)

#print(teacherprefix_count)

#Calculating probs for each class
for i in teacherprefix_count.iterrows():
    teacherprefix_count['1_prob'] = teacherprefix_count[1]/(teacherprefix_count[1]+teacherprefix_count[0])
    teacherprefix_count['0_prob'] = teacherprefix_count[0]/(teacherprefix_count[1]+teacherprefix_count[0])

#print(teacherprefix_count)

#appending prob values to train data in a new column

for idx,j in teacherprefix_count.iterrows():
    for idx,i in df_train.iterrows():
        if idx == df_train.at[idx, 'teacher_prefix']:
            df_train.at[idx, 'teacherprefix_1'] = teacherprefix_count.at[idx, '1_prob']
            df_train.at[idx, 'teacherprefix_0'] = teacherprefix_count.at[idx, '0_prob']
```

```
In [45]: print(df_train['teacherprefix_0'].head(2))
```

```
21523    0.32013
4975     0.32013
Name: teacherprefix_0, dtype: float64
```

```
In [46]: #appending prob values to test data in a new column. In case the class is not present in the test data
for idx,j in teacherprefix_count.iterrows():
    for idx,i in df_test.iterrows():
        if idx == df_test.at[idx, 'teacher_prefix']:
            df_test.at[idx, 'teacherprefix_1'] = teacherprefix_count.at[idx, '1_prob']
            df_test.at[idx, 'teacherprefix_0'] = teacherprefix_count.at[idx, '0_prob']

df_test['teacherprefix_1']=df_test['teacherprefix_1'].replace(np.nan,0.5)
df_test['teacherprefix_0']=df_test['teacherprefix_0'].replace(np.nan,0.5)

print(df_test['teacherprefix_0'].head(2))
```

```
33973    0.336594
27635    0.336594
Name: teacherprefix_0, dtype: float64
```

```
In [47]: df_train['teacherprefix_1']=df_train['teacherprefix_1'].replace(np.nan,0.5)
df_train['teacherprefix_0']=df_train['teacherprefix_0'].replace(np.nan,0.5)
```

2.2.1.5 Feature encoding for project_grade_category

```
In [48]: #https://stackoverflow.com/questions/3839729/count-unique-values-with-pandas-per

# Fetching unique value counts for each class
project_grade_category_count = pd.DataFrame()
project_grade_category_count[1] = df_train['project_grade_category'].where(df_train['project_grade_category'] != '0')
project_grade_category_count[0] = df_train['project_grade_category'].where(df_train['project_grade_category'] == '0')

#Replacing nan value counts with zeros
project_grade_category_count[1]=project_grade_category_count[1].replace(np.nan,0)
project_grade_category_count[0]=project_grade_category_count[0].replace(np.nan,0)

#print(project_grade_category_count)

#Calculating probs for each class
for i in project_grade_category_count.iterrows():
    project_grade_category_count['1_prob'] = project_grade_category_count[1]/(project_grade_category_count[1].sum())
    project_grade_category_count['0_prob'] = project_grade_category_count[0]/(project_grade_category_count[0].sum())

#print(project_grade_category_count)

#appending prob values to train data in a new column

for idx,j in project_grade_category_count.iterrows():
    for idx,i in df_train.iterrows():
        if idx == df_train.at[idx, 'project_grade_category']:
            df_train.at[idx, 'project_grade_category_1'] = project_grade_category_count[j]['1_prob']
            df_train.at[idx, 'project_grade_category_0'] = project_grade_category_count[j]['0_prob']

print(df_train.head(2))
```

	project_title \
21523	Wiggle While You Work!
4975	Working in Small Groups Is a Big Deal

	project_essay_1 \
21523	As the librarian, I am privileged to work with...
4975	My students come from diverse backgrounds, and...

	project_essay_2 ...	cat_1 \
21523	My Project\r\nMy K-5 students are active, enth...	0.656489
4975	My students will use this carpet each day duri...	0.651769

	cat_0	subcat_1	subcat_0	state_1	state_0	teacherprefix_1 \
21523	0.343511	0.664634	0.335366	0.704492	0.295508	0.67987
4975	0.348231	0.593939	0.406061	0.651772	0.348228	0.67987

	teacherprefix_0	project_grade_category_1	project_grade_category_0
21523	0.32013	0.668596	0.331404
4975	0.32013	0.680718	0.319282

```
In [49]: #appending prob values to test data in a new column. Incase the class is not part
for idx,j in project_grade_category_count.iterrows():
    for indx,i in df_test.iterrows():
        if idx == df_test.at[indx, 'project_grade_category']:
            df_test.at[indx, 'project_grade_category_1'] = project_grade_category_count
            df_test.at[indx, 'project_grade_category_0'] = project_grade_category_count

df_test['project_grade_category_1']=df_test['project_grade_category_1'].replace(
df_test['project_grade_category_0']=df_test['project_grade_category_0'].replace(

print(df_test.head(2))
```

```

      Unnamed: 0      id      teacher_id teacher_prefix \
33973      112656 p208177 3fe3dd300b49b1bcb20c5ce7c770c9c3      Ms.
27635      131772 p217130 931402be87fef92c660e68d12cc108d9      Ms.

      school_state      Date project_grade_category \
33973      LA 2016-12-07 15:03:46      Grades 3-5
27635      TX 2016-10-16 21:20:21      Grades PreK-2

      project_title \
33973      Silence is Golden!
27635  Robotics in the classroom. Bring on the iPads!

      project_essay_1 \
33973  My school is an at risk school located in the ...
27635  As a teacher in a low-income/high poverty scho...

      project_essay_2 ...      cat_1 \
33973  Silence is golden! My students just received ... ... 0.327586
27635  My students' lives will be changed, because th... ... 0.382998

      cat_0 subcat_1 subcat_0 state_1 state_0 teacherprefix_1 \
33973  0.327586 0.673077 0.326923 0.638217 0.361783 0.663406
27635  0.382998 0.604038 0.395962 0.615479 0.384521 0.663406

      teacherprefix_0 project_grade_category_1 project_grade_category_0
33973      0.336594      0.680718      0.319282
27635      0.336594      0.668596      0.331404

[2 rows x 38 columns]
```

```
In [50]: print(len(df_train.columns), len(df_test.columns))
```

```
38 38
```

2.2.2 Vectorizing Numerical features

2.2.2.1 Vectorizing price


```
In [51]: # check this one: https://www.youtube.com/watch?v=0H0qOcLn3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/skle
from sklearn.preprocessing import StandardScaler

# Reshape your data either using array.reshape(-1, 1)
print(df_train.columns)
price_scaler = StandardScaler()
price_scaler.fit(df_train['price'].values.reshape(-1,1)) # finding the mean and s
print(f"Mean : {price_scaler.mean_[0]}, Standard deviation : {np.sqrt(price_scaler

# Now standardize the data with above maen and variance.
price_train_standardized = price_scaler.transform(df_train['price'].values.reshape
price_test_standardized = price_scaler.transform(df_test['price'].values.reshape
```

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
      'Date', 'project_grade_category', 'project_title', 'project_essay_1',
      'project_essay_2', 'project_essay_3', 'project_essay_4',
      'project_resource_summary',
      'teacher_number_of_previously_posted_projects', 'project_is_approved',
      'clean_categories', 'clean_subcategories', 'essay', 'clean_essay',
      'clean_title', 'price', 'quantity', 'title_wc', 'essay_wc', 'neg',
      'neu', 'pos', 'compound', 'cat_1', 'cat_0', 'subcat_1', 'subcat_0',
      'state_1', 'state_0', 'teacherprefix_1', 'teacherprefix_0',
      'project_grade_category_1', 'project_grade_category_0'],
      dtype='object')
Mean : 312.1284854285714, Standard deviation : 377.19732859848654
```

2.2.2.2 Vectorizing no. of previously posted projects

```
In [52]: # check this one: https://www.youtube.com/watch?v=0H0qOcLn3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/skle
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings("ignore")

prev_proj_scaler = StandardScaler()
prev_proj_scaler.fit(df_train['teacher_number_of_previously_posted_projects'].va
print(f"Mean : {prev_proj_scaler.mean_[0]}, Standard deviation : {np.sqrt(prev_p

# Now standardize the data with above mean and variance.
prev_proj_train_standardized = prev_proj_scaler.transform(df_train['teacher_numbe
prev_proj_test_standardized = prev_proj_scaler.transform(df_test['teacher_number
```

```
Mean : 10.376914285714285, Standard deviation : 26.557037133978444
```

2.2.2.3 Vectorizing word counts of project title

```
In [53]: # check this one: https://www.youtube.com/watch?v=0H0q0cLn3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/skle
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings("ignore")

wc_title_scalar = StandardScaler()
wc_title_scalar.fit(df_train['title_wc'].values.reshape(-1,1)) # finding the mean
print(f"Mean : {wc_title_scalar.mean_[0]}, Standard deviation : {np.sqrt(wc_title

# Now standardize the data with above mean and variance.
wc_title_train_standardized = wc_title_scalar.transform(df_train['title_wc'].value
wc_title_test_standardized = wc_title_scalar.transform(df_test['title_wc'].value
```

Mean : 3.665542857142857, Standard deviation : 1.542806762140393

2.2.2.4 Vectorizing word counts of essay text

```
In [54]: # check this one: https://www.youtube.com/watch?v=0H0q0cLn3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/skle
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings("ignore")

wc_essay_scalar = StandardScaler()
wc_essay_scalar.fit(df_train['essay_wc'].values.reshape(-1,1)) # finding the mean
print(f"Mean : {wc_essay_scalar.mean_[0]}, Standard deviation : {np.sqrt(wc_essay

# Now standardize the data with above mean and variance.
wc_essay_train_standardized = wc_essay_scalar.transform(df_train['essay_wc'].value
wc_essay_test_standardized = wc_essay_scalar.transform(df_test['essay_wc'].value
```

Mean : 136.5252, Standard deviation : 35.570028223451594

2.2.2.5 Vectorizing sentimental scores of project essays

'neg', 'neu', 'pos', 'compound'

```
In [55]: # check this one: https://www.youtube.com/watch?v=0H0q0cLn3Z4&t=530s
# standardization sklearn: https://scikit-Learn.org/stable/modules/generated/skle
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings("ignore")

neg_score_scalar = StandardScaler()
neg_score_scalar.fit(df_train['neg'].values.reshape(-1,1)) # finding the mean and
print(f"Mean : {neg_score_scalar.mean_[0]}, Standard deviation : {np.sqrt(neg_sco

# Now standardize the data with above mean and variance.
neg_score_train_standardized = neg_score_scalar.transform(df_train['neg'].values
neg_score_test_standardized = neg_score_scalar.transform(df_test['neg'].values.r
```

Mean : 0.04848257142857143, Standard deviation : 0.036422845016583216

```
In [56]: # check this one: https://www.youtube.com/watch?v=0H0q0cLn3Z4&t=530s
# standardization sklearn: https://scikit-Learn.org/stable/modules/generated/skle
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings("ignore")

neu_score_scalar = StandardScaler()
neu_score_scalar.fit(df_train['neu'].values.reshape(-1,1)) # finding the mean and
print(f"Mean : {neu_score_scalar.mean_[0]}, Standard deviation : {np.sqrt(neu_sco

# Now standardize the data with above mean and variance.
neu_score_train_standardized = neu_score_scalar.transform(df_train['neu'].values
neu_score_test_standardized = neu_score_scalar.transform(df_test['neu'].values.r
```

Mean : 0.6680472857142857, Standard deviation : 0.07614768934532848

```
In [57]: # check this one: https://www.youtube.com/watch?v=0H0q0cLn3Z4&t=530s
# standardization sklearn: https://scikit-Learn.org/stable/modules/generated/skle
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings("ignore")

compound_score_scalar = StandardScaler()
compound_score_scalar.fit(df_train['compound'].values.reshape(-1,1)) # finding the
print(f"Mean : {compound_score_scalar.mean_[0]}, Standard deviation : {np.sqrt(c

# Now standardize the data with above mean and variance.
compound_score_train_standardized = compound_score_scalar.transform(df_train['co
compound_score_test_standardized = compound_score_scalar.transform(df_test['comp
```

Mean : 0.957751222857143, Standard deviation : 0.15834582202325045

```
In [58]: from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings("ignore")

pos_score_scalar = StandardScaler()
pos_score_scalar.fit(df_train['pos'].values.reshape(-1,1)) # finding the mean and
print(f"Mean : {pos_score_scalar.mean_[0]}, Standard deviation : {np.sqrt(pos_score_scalar.var_[0])}")

# Now standardize the data with above mean and variance.
pos_score_train_standardized = pos_score_scalar.transform(df_train['pos'].values.reshape(-1,1))
pos_score_test_standardized = pos_score_scalar.transform(df_test['pos'].values.reshape(-1,1))
```

Mean : 0.2834720857142857, Standard deviation : 0.07832149800255407

2.2.2.6 Vectorizing Quantity

```
In [59]: # check this one: https://www.youtube.com/watch?v=0H0qOcLn3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings("ignore")

qty_scalar = StandardScaler()
qty_scalar.fit(df_train['quantity'].values.reshape(-1,1)) # finding the mean and
print(f"Mean : {qty_scalar.mean_[0]}, Standard deviation : {np.sqrt(qty_scalar.var_[0])}")

# Now standardize the data with above mean and variance.
qty_train_standardized = qty_scalar.transform(df_train['quantity'].values.reshape(-1,1))
qty_test_standardized = qty_scalar.transform(df_test['quantity'].values.reshape(-1,1))
```

Mean : 17.594514285714286, Standard deviation : 27.136284273876797

2.3 Make Data Model Ready: encoding eassay, and project_title

2.3.1 Vectorizing Text data

2.3.1.2 TFIDF vectorizer for essay text

```
In [60]: from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)

text_train_tfidf = vectorizer.fit_transform(df_train['clean_essay'])
text_test_tfidf = vectorizer.transform(df_test['clean_essay'])
print("Shape of matrix after one hot encoding ",text_train_tfidf.shape, text_test_tfidf.shape)
```

Shape of matrix after one hot encoding (35000, 10463) (15000, 10463)

2.3.1.2 Using Pretrained Models: TFIDF weighted W2V for essay text

```
In [61]: from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)

title_train_tfidf = vectorizer.fit_transform(df_train['clean_title'])
title_test_tfidf = vectorizer.transform(df_test['clean_title'])

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

Shape of matrix after one hot encoding (35000, 1586) (15000, 1586)

```
In [62]: # 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
```

```
In [63]: model = loadGloveModel('glove.42B.300d.txt')
```

Loading Glove Model

```

In [64]: words = []
         for i in preprocessed_essays:
             words.extend(i.split(' '))

         for i in preprocessed_title:
             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-

         import pickle
         with open('glove.42B.300d.txt', 'wb') as f:
             pickle.dump(words_courpus, f)

```

```

all the words in the coupus 7015309
the unique words in the coupus 43531
The number of words that are present in both glove vectors and our coupus 39363
( 90.425 %)
word 2 vec length 39363

```

```

In [65]: # storing variables into pickle files python: http://www.jessicayung.com/how-to-
         # make sure you have the glove_vectors file

         with open('glove.42B.300d.txt', 'rb') as f:
             model = pickle.load(f)
             glove_words = set(model.keys())

```

```

In [66]: # glove_words

```

```
In [67]: # average Word2Vec
# compute average word2vec for each review.

avg_w2v_train_text_vectors = []; # the avg-w2v for each sentence/review is stored
for sentence in tqdm(df_train['clean_essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length. 50 is the size of
    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_train_text_vectors.append(vector)

print(len(avg_w2v_train_text_vectors))
print(len(avg_w2v_train_text_vectors[0]))
```

```
0%|
| 0/35000 [00:00<?, ?it/s]
0%|
| 1/35000 [00:00<3:15:35, 2.98it/s]
0%|
| 153/35000 [00:00<2:16:26, 4.26it/s]
1%|
| 446/35000 [00:00<1:34:45, 6.08it/s]
2%|
| 799/35000 [00:00<1:05:42, 8.68it/s]
3%|
| 1021/35000 [00:00<45:46, 12.37it/s]
4%|
| 1268/35000 [00:00<31:52, 17.63it/s]
4%|
| 1475/35000 [00:00<22:16, 25.08it/s]
5%|
| 1738/35000 [00:01<15:32, 35.68it/s]
6%|
```

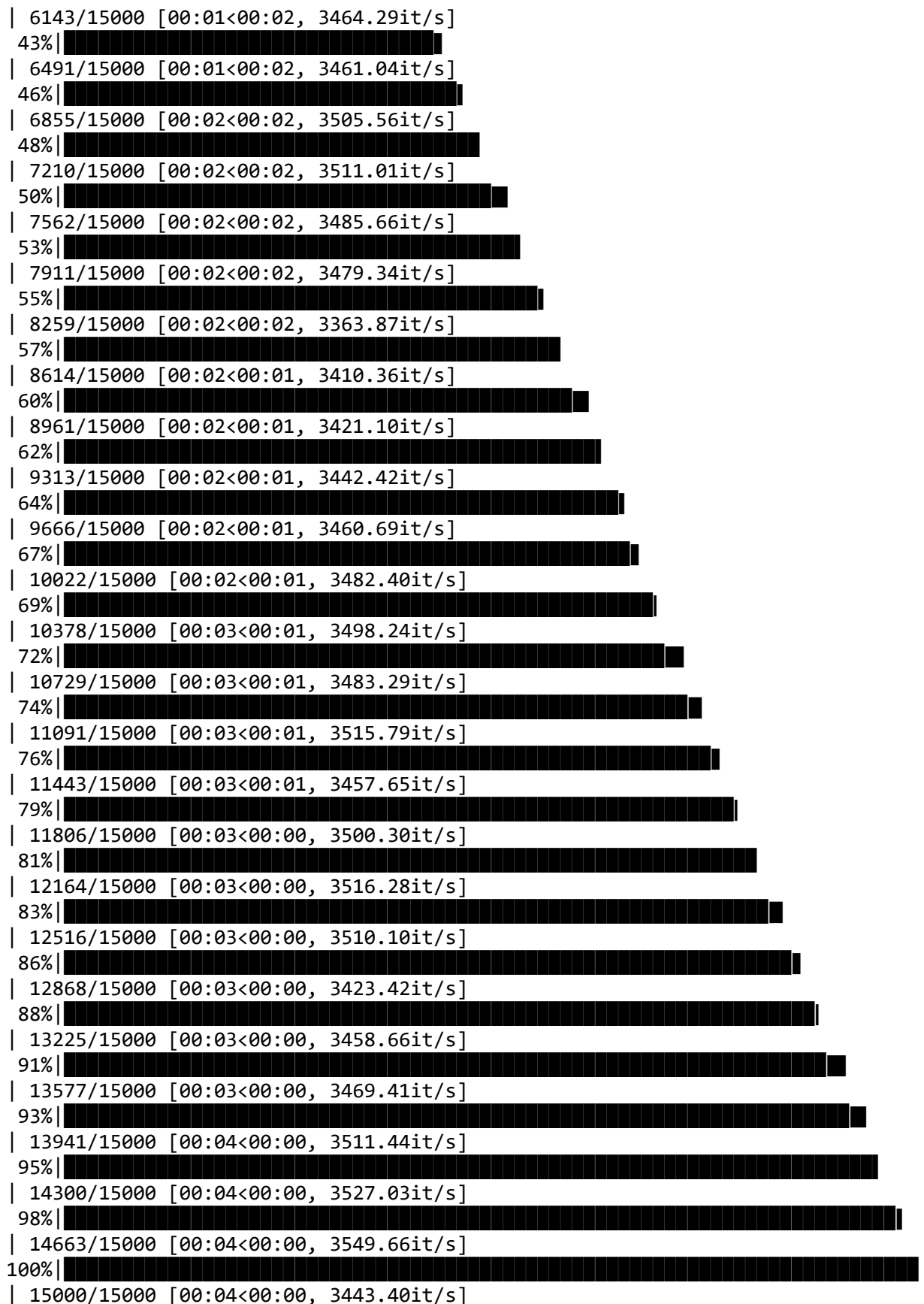


```
In [68]: # average Word2Vec
# compute average word2vec for each review.

avg_w2v_test_text_vectors = []; # the avg-w2v for each sentence/review is stored
for sentence in tqdm(df_test['clean_essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length. 50 is the size of
    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_test_text_vectors.append(vector)

print(len(avg_w2v_test_text_vectors))
print(len(avg_w2v_test_text_vectors[0]))
```

```
0%|
| 0/15000 [00:00<?, ?it/s]
2%|██
| 327/15000 [00:00<00:04, 3246.01it/s]
4%|████
| 656/15000 [00:00<00:04, 3252.05it/s]
7%|██████
| 1011/15000 [00:00<00:04, 3329.50it/s]
8%|████████
| 1274/15000 [00:00<00:04, 3074.18it/s]
11%|██████████
| 1607/15000 [00:00<00:04, 3140.27it/s]
13%|███████████
| 1927/15000 [00:00<00:04, 3151.43it/s]
15%|████████████
| 2298/15000 [00:00<00:03, 3294.08it/s]
18%|█████████████
| 2643/15000 [00:00<00:03, 3332.28it/s]
20%|██████████████
| 3003/15000 [00:00<00:03, 3401.26it/s]
22%|███████████████
| 3342/15000 [00:01<00:03, 3390.10it/s]
25%|████████████████
| 3697/15000 [00:01<00:03, 3429.11it/s]
27%|█████████████████
| 4040/15000 [00:01<00:03, 3422.36it/s]
29%|██████████████████
| 4391/15000 [00:01<00:03, 3440.42it/s]
32%|███████████████████
| 4733/15000 [00:01<00:03, 3416.85it/s]
34%|████████████████████
| 5094/15000 [00:01<00:02, 3465.28it/s]
36%|█████████████████████
| 5440/15000 [00:01<00:02, 3435.58it/s]
39%|██████████████████████
| 5790/15000 [00:01<00:02, 3446.70it/s]
41%|███████████████████████
```



15000

300


```
In [72]: # average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_train_text_vectors = []; # the avg-w2v for each sentence/review is stored here
for sentence in tqdm(df_train['clean_essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length. 50 is the size of the embedding
    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))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_train_text_vectors.append(vector)

print(len(tfidf_w2v_train_text_vectors))
print(len(tfidf_w2v_train_text_vectors[0]))
```

```
0%|
| 0/35000 [00:00<?, ?it/s]
0%|
| 53/35000 [00:00<01:07, 515.76it/s]
0%||
| 86/35000 [00:00<01:19, 440.60it/s]
0%||
| 135/35000 [00:00<01:16, 453.41it/s]
1%||
| 188/35000 [00:00<01:13, 473.03it/s]
1%||
| 240/35000 [00:00<01:11, 483.86it/s]
1%||
| 295/35000 [00:00<01:09, 500.97it/s]
1%||
| 346/35000 [00:00<01:08, 502.56it/s]
1%||
| 402/35000 [00:00<01:07, 516.02it/s]
1%||
```

```
In [73]: # average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_test_text_vectors = []; # the avg-w2v for each sentence/review is stored
for sentence in tqdm(df_test['clean_essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length. 50 is the size of the embedding
    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
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_test_text_vectors.append(vector)

print(len(tfidf_w2v_test_text_vectors))
print(len(tfidf_w2v_test_text_vectors[0]))
```

```
| 13572/15000 [00:28<00:03, 463.39it/s]
```

```
91%|
```

```
| 13624/15000 [00:29<00:02, 476.76it/s]
```

```
91%|
```

```
| 13673/15000 [00:29<00:02, 478.70it/s]
```

```
91%|
```

```
| 13723/15000 [00:29<00:02, 483.82it/s]
```

```
92%|
```

```
| 13772/15000 [00:29<00:02, 459.71it/s]
```

```
92%|
```

```
| 13825/15000 [00:29<00:02, 478.45it/s]
```

```
93%|
```

```
| 13881/15000 [00:29<00:02, 499.33it/s]
```

```
93%|
```

```
| 13933/15000 [00:29<00:02, 502.82it/s]
```

```
93%|
```

```
| 13989/15000 [00:29<00:01, 516.23it/s]
```

```
94%|
```

```
| 14042/15000 [00:29<00:01, 510.89it/s]
```

```
94%|
```

2.3.1.4 Using Pretrained Models: TFIDF weighted W2V for title

```
In [74]: # Similarly you can vectorize for title also

tfidf_model = TfidfVectorizer()
tfidf_model.fit_transform(df_train['clean_title'])
# 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 [75]: # average Word2Vec
# compute average word2vec for each project title.
tfidf_w2v_train_title_vectors = []; # the avg-w2v for each sentence/review is stored
for sentence in tqdm(df_train['clean_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length. 50 is the size of the embedding
    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
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_train_title_vectors.append(vector)

print(len(tfidf_w2v_train_title_vectors))
print(len(tfidf_w2v_train_title_vectors[0]))
```

```
0%|
| 0/35000 [00:00<?, ?it/s]
10%|██████████
| 3478/35000 [00:00<00:00, 34527.88it/s]
19%|██████████
| 6794/35000 [00:00<00:00, 34025.05it/s]
29%|██████████
| 10119/35000 [00:00<00:00, 33713.45it/s]
38%|██████████
| 13244/35000 [00:00<00:00, 32858.92it/s]
47%|██████████
| 16396/35000 [00:00<00:00, 32372.67it/s]
57%|██████████
| 19776/35000 [00:00<00:00, 32721.96it/s]
66%|██████████
| 23240/35000 [00:00<00:00, 33201.30it/s]
76%|██████████
| 26730/35000 [00:00<00:00, 33621.95it/s]
86%|██████████
| 30067/35000 [00:00<00:00, 33476.01it/s]
95%|██████████
| 33336/35000 [00:01<00:00, 33162.41it/s]
100%|██████████
35000/35000 [00:01<00:00, 33137.43it/s]
```

```
35000
300
```

```
In [76]: # average Word2Vec
# compute average word2vec for each project title.
tfidf_w2v_test_title_vectors = []; # the avg-w2v for each sentence/review is stored
for sentence in tqdm(df_test['clean_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length. 50 is the size of glove
    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
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_test_title_vectors.append(vector)

print(len(tfidf_w2v_test_title_vectors))
print(len(tfidf_w2v_test_title_vectors[0]))
```

```
0%|
| 0/15000 [00:00<?, ?it/s]
21%|██████████
| 3205/15000 [00:00<00:00, 31806.00it/s]
43%|██████████
| 6460/15000 [00:00<00:00, 31956.17it/s]
66%|██████████
| 9882/15000 [00:00<00:00, 32535.37it/s]
89%|██████████
| 13304/15000 [00:00<00:00, 32953.90it/s]
100%|██████████
15000/15000 [00:00<00:00, 32908.36it/s]

15000
300
```

2.4 Applying Decision Tree Classifier on different kinds of featurizations as mentioned in the instructions

GBDT (xgboost/lightgbm) Applying Models on different kind of featurization as mentioned in the instructions

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

Apply GBDT on these feature sets

Set 1: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)+sentiment Score of eassay(check the bellow example, include all 4 values as 4 features).

Set 2: categorical (instead of one hot encoding, try response coding: use probability values), numerical features + project_title (TFIDF W2V) + preprocessed_eassay (TFIDF W2V) The hyper parameter tuning (Consider any two hyper parameters) Find the best hyper parameter which will give the maximum AUC value find the best hyper parameter 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

Hyper parameter tuning method: GridSearch

Applying GBDT Classifier TFIDF, **SET 1** (GridSearch)

Set 1: categorical (instead of one hot encoding, try response coding: use probability values), numerical features + project_title (TFIDF) + preprocessed_eassay (TFIDF) + sentiment Score of eassay (check the below example, include all 4 values as 4 features).

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

Hyper parameter tuning method: GridSearch

In [77]: [#https://www.digitalocean.com/community/tutorials/how-to-plot-data-in-python-3-us](https://www.digitalocean.com/community/tutorials/how-to-plot-data-in-python-3-us)
[#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score)
[#https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter](https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter)

```
from scipy.sparse import hstack
from sklearn.model_selection import RandomizedSearchCV

import matplotlib.patches as mpatches
from sklearn.metrics import roc_auc_score

x_train = hstack((df_train['cat_1'].values.reshape(-1,1), df_train['cat_0'].values.reshape(-1,1),
                  df_train['subcat_0'].values.reshape(-1,1), df_train['state_1'].values.reshape(-1,1),
                  df_train['teacherprefix_1'].values.reshape(-1,1), df_train['teacher_satisfaction'].values.reshape(-1,1),
                  df_train['project_grade_category_1'].values.reshape(-1,1), df_train['project_grade_category_2'].values.reshape(-1,1),
                  price_train_standardized, prev_proj_train_standardized, wc_title_train_standardized, wc_title_test_standardized,
                  pos_score_train_standardized, neu_score_train_standardized, neg_score_train_standardized,
                  qty_train_standardized, text_train_tfidf, title_train_tfidf))
y_train = df_train['project_is_approved']

x_test = hstack((df_test['cat_1'].values.reshape(-1,1), df_test['cat_0'].values.reshape(-1,1),
                 df_test['subcat_0'].values.reshape(-1,1), df_test['state_1'].values.reshape(-1,1),
                 df_test['teacherprefix_1'].values.reshape(-1,1), df_test['teacher_satisfaction'].values.reshape(-1,1),
                 df_test['project_grade_category_1'].values.reshape(-1,1), df_test['project_grade_category_2'].values.reshape(-1,1),
                 prev_proj_test_standardized, wc_title_test_standardized, wc_title_train_standardized, wc_title_test_standardized,
                 pos_score_test_standardized, neu_score_test_standardized, neg_score_test_standardized,
                 qty_test_standardized, text_test_tfidf, title_test_tfidf))
y_test = df_test['project_is_approved']

print(x_train.shape, type(x_train), y_train.shape, type(y_train))
print(x_test.shape, type(x_test), y_test.shape, type(y_test))
```

```
(35000, 12068) <class 'scipy.sparse.coo.coo_matrix'> (35000,) <class 'pandas.core.series.Series'>
(15000, 12068) <class 'scipy.sparse.coo.coo_matrix'> (15000,) <class 'pandas.core.series.Series'>
```

```
In [78]: #https://stackabuse.com/cross-validation-and-grid-search-for-model-selection-in-p
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSe
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBClassifier

gbdt = XGBClassifier()

#Brute force approach for finding best K value
parameters = {'n_estimators': [10, 50, 100, 200, 500,1000], 'max_depth':[2, 3, 4,

#Training the model on train data
gs = RandomizedSearchCV(gbdt,parameters ,cv=3, scoring='roc_auc',n_jobs=-1,return
gs.fit(x_train, y_train)
```

```
Out[78]: RandomizedSearchCV(cv=3, error_score='raise-deprecating',
        estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_b
ylevel=1,
        colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
        max_delta_step=0, max_depth=3, min_child_weight=1, missing=None,
        n_estimators=100, n_jobs=1, nthread=None,
        objective='binary:logistic', random_state=0, reg_alpha=0,
        reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
        subsample=1, verbosity=1),
        fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
        param_distributions={'n_estimators': [10, 50, 100, 200, 500, 1000],
'max_depth': [2, 3, 4, 5]},
        pre_dispatch='2*n_jobs', random_state=None, refit=True,
        return_train_score=True, scoring='roc_auc', verbose=0)
```

```
In [79]: print('Best score: ',gs.best_score_)
print('k value with best score: ',gs.best_params_)
print(' ')
print(' ')
print('Train AUC scores')
print(gs.cv_results_['mean_train_score'])
print('CV AUC scores')
print(gs.cv_results_['mean_test_score'])
```

Best score: 0.7493643154433881

k value with best score: {'n_estimators': 500, 'max_depth': 2}

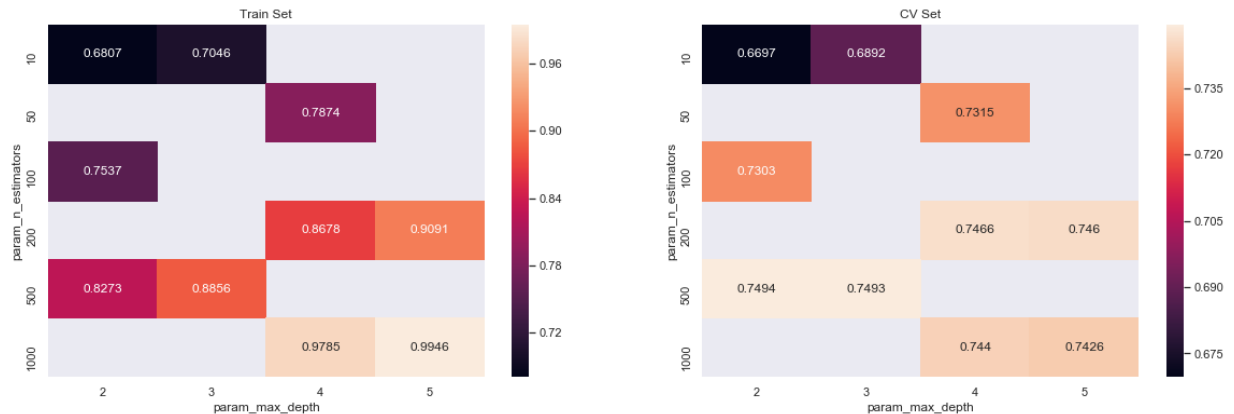
Train AUC scores

[0.78744269 0.75366858 0.88559228 0.68067226 0.90905401 0.70459099
0.99455287 0.86782992 0.97850563 0.82725446]

CV AUC scores

[0.7314984 0.73027368 0.74927721 0.66968437 0.74596511 0.68923037
0.74255738 0.74663995 0.74395347 0.74936432]

```
In [80]: import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(gs.cv_results_).groupby(['param_n_estimators', 'param_
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



```
In [81]: max_d = gs.best_params_['max_depth']
n_est = gs.best_params_['n_estimators']
print(gs.best_params_)
```

```
{'n_estimators': 500, 'max_depth': 2}
```

```
In [82]: def pred_prob(clf, data):
y_pred = []
y_pred = clf.predict_proba(data)[: ,1]
return y_pred
```

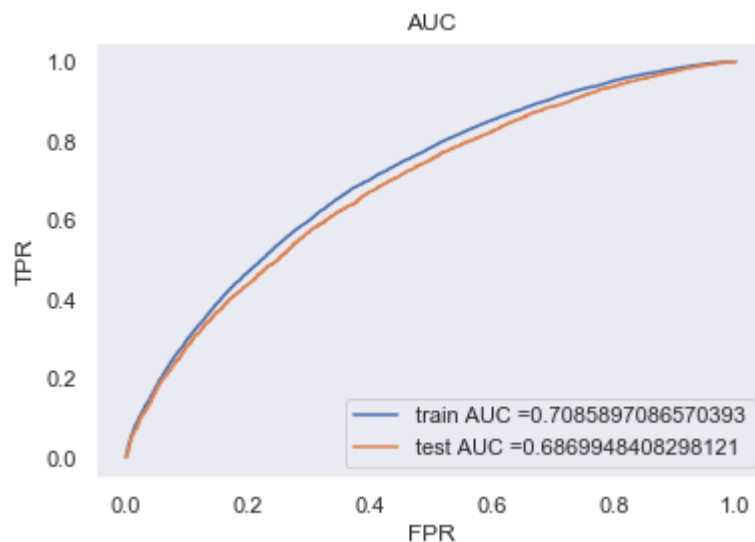
```
In [83]: # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html
from sklearn.metrics import roc_curve, auc
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
model = RandomForestClassifier(max_depth = max_d, n_estimators = n_est)

model.fit(x_train,y_train)

y_train_pred = pred_prob(model,x_train)
y_test_pred = pred_prob(model,x_test)

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.close
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("FPR")
plt.ylabel("TPR")
plt.title("AUC")
plt.grid()
plt.show()
```



```
In [84]: # we are writing our own function for predict, with defined threshold  
# we will pick a threshold that will give the least fpr  
def find_best_threshold(threshold, fpr, tpr):  
    t = threshold[np.argmax(tpr*(1-fpr))]  
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high  
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold",  
          return t  
  
def predict_with_best_t(proba, threshold):  
    predictions = []  
    for i in proba:  
        if i>=threshold:  
            predictions.append(1)  
        else:  
            predictions.append(0)  
    return predictions
```

```
In [85]: #our objective here is to make auc the maximum  
#so we find the best threshold that will give the least fpr  
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)  
print("Train confusion matrix")  
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 0.4275978017458856 for threshold 0.669  
Train confusion matrix  
[[ 7604  3975]  
 [ 8171 15250]]
```

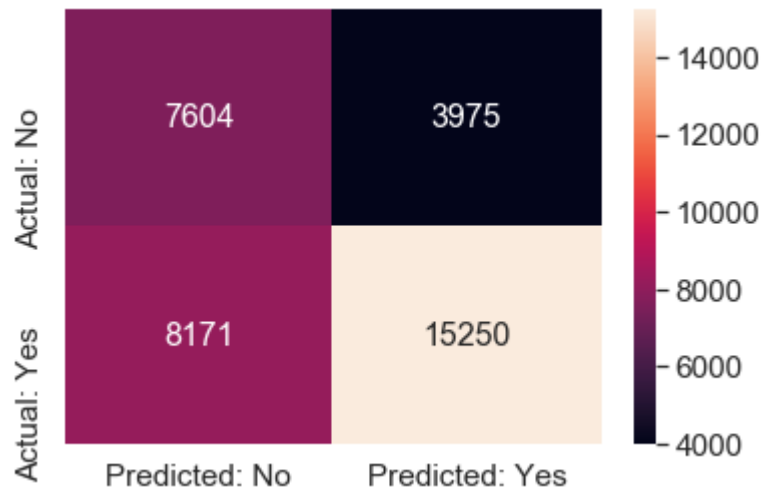
```
In [86]: # https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

print("Train data confusion matrix")

confusion_matrix_df_train = pd.DataFrame(confusion_matrix(y_train, predict_with_l
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_train, annot=True,annot_kws={"size": 16}, fmt='g
```

Train data confusion matrix

Out[86]: <matplotlib.axes._subplots.AxesSubplot at 0x22cc4c89be0>



```
In [87]: print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

Test confusion matrix

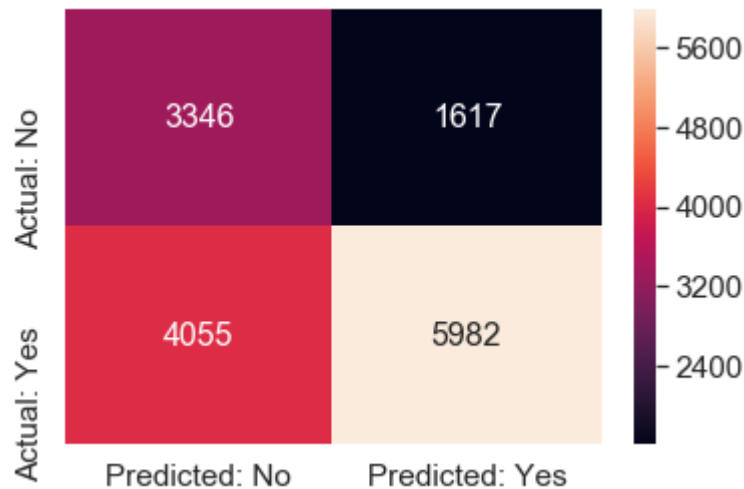
```
[[3346 1617]
 [4055 5982]]
```

```
In [88]: print("Test data confusion matrix")

confusion_matrix_df_test = pd.DataFrame(confusion_matrix(y_test, predict_with_be
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_test, annot=True,annot_kws={"size": 16}, fmt='g'
```

Test data confusion matrix

Out[88]: <matplotlib.axes._subplots.AxesSubplot at 0x22dff41cbe0>



Applying GBDT TFIDF W2V, SET 2

Set 2: categorical (instead of one hot encoding, try response coding: use probability values), numerical features + project_title (TFIDF W2V) + preprocessed_eassay (TFIDF W2V). The hyperparameter tuning (Consider any two hyper parameters) Find the best hyper parameter which will give the maximum AUC value find the best hyper parameter 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

Hyper parameter tuning method: GridSearch


```
In [89]: #https://www.digitalocean.com/community/tutorials/how-to-plot-data-in-python-3-us
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score
#https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter

from scipy.sparse import hstack
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
import matplotlib.patches as mpatches
from sklearn.metrics import roc_auc_score

y_train_tfidf_w2v = df_train['project_is_approved']

y_test_tfidf_w2v = df_test['project_is_approved']

print( y_train_tfidf_w2v.shape, type(y_train_tfidf_w2v))
print( y_test_tfidf_w2v.shape, type(y_test_tfidf_w2v))

(35000,) <class 'pandas.core.series.Series'>
(15000,) <class 'pandas.core.series.Series'>
```

```
In [90]: #https://blog.csdn.net/w55100/article/details/90369779
# if you use hstack without converting it into to a sparse matrix first,
#it shows an error: blocks must be 2-D

from scipy.sparse import coo_matrix, hstack
tr1 = coo_matrix(df_train['cat_1'].values.reshape(-1,1))
tr2 = coo_matrix(df_train['cat_0'].values.reshape(-1,1))
tr3 = coo_matrix(df_train['subcat_1'].values.reshape(-1,1))
tr4 = coo_matrix(df_train['subcat_0'].values.reshape(-1,1))
tr5 = coo_matrix(df_train['state_1'].values.reshape(-1,1))
tr6 = coo_matrix(df_train['state_0'].values.reshape(-1,1))
tr7 = coo_matrix(df_train['teacherprefix_1'].values.reshape(-1,1))
tr8 = coo_matrix(df_train['teacherprefix_0'].values.reshape(-1,1))
tr9 = coo_matrix(df_train['project_grade_category_1'].values.reshape(-1,1))
tr10 = coo_matrix(df_train['project_grade_category_0'].values.reshape(-1,1))
tr11 = coo_matrix(price_train_standardized)
tr12 = coo_matrix(prev_proj_train_standardized)
tr13 = coo_matrix(wc_title_train_standardized)
tr14 = coo_matrix(wc_essay_train_standardized)
tr15 = coo_matrix(qty_train_standardized)
tr16 = coo_matrix(tfidf_w2v_train_text_vectors)
tr17 = coo_matrix(tfidf_w2v_train_title_vectors)
```

```
In [91]: X_train = hstack([tr1,tr2,tr3,tr4,tr5,tr6,tr7,tr8,tr9,tr10,tr11,tr12,tr13,tr14,ti
```

```
In [92]: from scipy.sparse import coo_matrix, hstack
te1 = coo_matrix(df_test['cat_1'].values.reshape(-1,1))
te2 = coo_matrix(df_test['cat_0'].values.reshape(-1,1))
te3 = coo_matrix(df_test['subcat_1'].values.reshape(-1,1))
te4 = coo_matrix(df_test['subcat_0'].values.reshape(-1,1))
te5 = coo_matrix(df_test['state_1'].values.reshape(-1,1))
te6 = coo_matrix(df_test['state_0'].values.reshape(-1,1))
te7 = coo_matrix(df_test['teacherprefix_1'].values.reshape(-1,1))
te8 = coo_matrix(df_test['teacherprefix_0'].values.reshape(-1,1))
te9 = coo_matrix(df_test['project_grade_category_1'].values.reshape(-1,1))
te10 = coo_matrix(df_test['project_grade_category_0'].values.reshape(-1,1))
te11 = coo_matrix(price_test_standardized)
te12 = coo_matrix(prev_proj_test_standardized)
te13 = coo_matrix(wc_title_test_standardized)
te14 = coo_matrix(wc_essay_test_standardized)
te15 = coo_matrix(qty_test_standardized)
te16 = coo_matrix(tfidf_w2v_test_text_vectors)
te17 = coo_matrix(tfidf_w2v_test_title_vectors)
```

```
In [93]: X_test = hstack([te1,te2,te3,te4,te5,te6,te7,te8,te9,te10,te11,te12,te13,te14,te15,te16,te17])
```

```
In [94]: from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBClassifier

gbdt = XGBClassifier()

parameters = {'n_estimators': [10, 50, 100, 200, 500, 1000], 'max_depth': [2, 3, 4, 5],
              'min_child_weight': [1, 2, 5, 10, 20, 50, 100, 200, 500, 1000],
              'learning_rate': [0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
              'gamma': [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
              'colsample_bytree': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
              'colsample_bynode': [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288, 1048576, 2097152, 4194304, 8388608, 16777216, 33554432, 67108864, 134217728, 268435456, 536870912, 1073741824, 2147483648, 4294967296, 8589934592, 17179869184, 34359738368, 68719476736, 137438953472, 274877906944, 549755813888, 1099511627776, 2199023255552, 4398046511104, 8796093022208, 17592186044416, 35184372088832, 70368744177664, 140737488355328, 281474976710656, 562949953421312, 1125899906842624, 2251799813685248, 4503599627370496, 9007199254740992, 18014398509481984, 36028797018963968, 72057594037927936, 144115188075855872, 288230376151711744, 576460752303423488, 1152921504606846976, 2305843009213693952, 4611686018427387904, 9223372036854775808, 18446744073709551616, 36893488147419103232, 73786976294838206464, 147573952589676412928, 295147905179352825856, 590295810358705651712, 1180591620717411303424, 2361183241434822606848, 4722366482869645213696, 9444732965739290427392, 18889465931478580854784, 37778931862957161709568, 75557863725914323419136, 151115727451828646838272, 302231454903657293676544, 604462909807314587353088, 1208925819614629174706176, 2417851639229258349412352, 4835703278458516698824704, 9671406556917033397649408, 19342813113834066795298816, 38685626227668133590597632, 77371252455336267181195264, 154742504910672534362390528, 309485009821345068724781056, 618970019642690137449562112, 1237940039285380274899124224, 2475880078570760549798248448, 4951760157141521099596496896, 9903520314283042199192993792, 19807040628566084398385987584, 39614081257132168796771975168, 79228162514264337593543950336, 158456325028528675187087900672, 316912650057057350374175801344, 633825300114114700748351602688, 1267650600228229401496703205376, 2535301200456458802993406410752, 5070602400912917605986812821504, 10141204801825835211973625643008, 20282409603651670423947251286016, 40564819207303340847894502572032, 81129638414606681695789005144064, 162259276829213363391578010288128, 324518553658426726783156020576256, 649037107316853453566312041152512, 1298074214633706907132624082305024, 2596148429267413814265248164610048, 5192296858534827628530496329220096, 10384593717069655257060992658440192, 20769187434139310514121985316880384, 41538374868278621028243970633760768, 83076749736557242056487941267521536, 166153499473114484112975882535043072, 332306998946228968225951765070086144, 664613997892457936451903530140172288, 1329227995784915872903807060280344576, 2658455991569831745807614120560689152, 5316911983139663491615228241121378304, 10633823966279326983230456482242756608, 21267647932558653966460912964485513216, 42535295865117307932921825928971026432, 85070591730234615865843651857942052864, 170141183460469231731687303715884105728, 340282366920938463463374607431768211456, 680564733841876926926749214863536422912, 1361129467683753853853498429727072845824, 2722258935367507707706996859454145691648, 5444517870735015415413993718908291383296, 10889035741470030830827987437816582766592, 21778071482940061661655974875633165533184, 43556142965880123323311949751266331066368, 87112285931760246646623899502532662132736, 174224571863520493293247799005065324265472, 348449143727040986586495598010130648530944, 696898287454081973172991196020261297061888, 1393796574908163946345982392040522594123776, 2787593149816327892691964784081045188247552, 5575186299632655785383929568162090376495104, 11150372599265311570767859136324180752990208, 22300745198530623141535718272648361505980416, 44601490397061246283071436545296723011960832, 89202980794122492566142873090593446023921664, 178405961588244985132285746181186892047843328, 356811923176489970264571492362373784095686656, 713623846352979940529142984724747568191373312, 1427247692705959881058285969449495136382746624, 2854495385411919762116571938898990272765493248, 5708990770823839524233143877797980545530986496, 11417981541647679048466287755595961091061972992, 22835963083295358096932575511191922182123945984, 45671926166590716193865151022383844364247891968, 91343852333181432387730302044767688728495783936, 182687704666362864775460604089535377456991567872, 365375409332725729550921208179070754913983135744, 730750818665451459101842416358141509827966271488, 1461501637330902918203684832716283019655932542976, 2923003274661805836407369665432566039311865085952, 5846006549323611672814739330865132078623730171904, 11692013098647223345629478661730264157247460343808, 23384026197294446691258957323460528314494920687616, 46768052394588893382517914646921056628989841375232, 93536104789177786765035829293842113257979682750464, 187072209578355573530071658587684226515959365500928, 374144419156711147060143317175368453031918731001856, 748288838313422294120286634350736906063837462003712, 1496577676626844588240573268701473812127674924007424, 2993155353253689176481146537402947624255349848014848, 5986310706507378352962293074805895248510699696029696, 11972621413014756705924586149611790497021399392059392, 23945242826029513411849172299223580994042798784118784, 47890485652059026823698344598447161988085597568237568, 95780971304118053647396689196894323976171195136475136, 191561942608236107294793378393788647952342390272950272, 383123885216472214589586756787577295904684780545900544, 766247770432944429179173513575154591809369561091801088, 1532495540865888858358347027150309183618739122183602176, 3064991081731777716716694054300618367237478244367204352, 6129982163463555433433388108601236734474956488734408704, 12259964326927110866866776217202473468949912977468817408, 24519928653854221733733552434404946937899825954937634816, 49039857307708443467467104868809893875799651909875269632, 98079714615416886934934209737619787751599303819750539264, 196159429230833773869868419475239575503198607639501078528, 392318858461667547739736838950479151006397215279002157056, 784637716923335095479473677900958302012794430558004314112, 1569275433846670190958947355801916604025588861116008628224, 3138550867693340381917894711603833208051177722232017256448, 6277101735386680763835789423207666416102355444464034512896, 12554203470773361527671578846415332832204710888928069025792, 25108406941546723055343157692830665664409421777856138051584, 50216813883093446110686315385661331328818843555712276103168, 100433627766186892221372630771322662657637687111424552206336, 200867255532373784442745261542645325315275374222849104412672, 401734511064747568885490523085290650630550748445698208825344, 803469022129495137770981046170581301261101496891396417650688, 1606938044258990275541962092341162602522202993782792835301376, 3213876088517980551083924184682325205044405987565585670602752, 6427752177035961102167848369364650410088811975131171341205504, 12855504354071922204335696738729300820177623950262342682411008, 25711008708143844408671393477458601640355247900524685364822016, 51422017416287688817342786954917203280710495801049370729644032, 102844034832575377634685573909834406561420991602098741459288064, 205688069665150755269371147819668813122841983204197482918576128, 411376139330301510538742295639337626245683966408394965837152256, 822752278660603021077484591278675252491367932816789931674304512, 1645504557321206042154969182557350504982735865633579863348609024, 3291009114642412084309938365114701009965471731267159726697218048, 6582018229284824168619876730229402019930943462534319453394436096, 13164036458569648337239753460458804039861886925068638906788872192, 26328072917139296674479506920917608079723773850137277813577744384, 52656145834278593348959013841835216159447547700274555627155488768, 105312291668557186697918027683670432318895095400549111254310977536, 210624583337114373395836055367340864637790190801098222508621955072, 421249166674228746791672110734681729275580381602196445017243910144, 842498333348457493583344221469363458551160763204392890034487820288, 1684996666696914987166688442938726917102321526408785780068975640576, 3369993333393829974333376885877453834204643052817571560137951281152, 6739986666787659948666753771754907668409286105635143120275902562304, 13479973333575319897333507543509815336818572211270286240551805124608, 26959946667150639794667015087019630673637144422540572481103610249216, 53919893334301279589334030174039261347274288845081144962207220498432, 107839786668602559178668060348078522694548577690162289924414440996864, 215679573337205118357336120696157045389097155380324579848828881993728, 431359146674410236714672241392314090778194310760649159697657763987456, 862718293348820473429344482784628181556388621521298319395315527974912, 1725436586697640946858688965569256363112777243042596638790631055949824, 3450873173395281893717377931138512726225554486085193277581262111899648, 6901746346790563787434755862277025452451108972170386555162524223799296, 13803492693581127574869511724554050904902217944340773110325048447598592, 27606985387162255149739023449108101809804435888681546220650096895197184, 55213970774324510299478046898216203619608871777363092441300193790394368, 110427941548649020598956093796432407239217743554726184882600387580788736, 220855883097298041197912187592864814478435487109452369765200775161577472, 441711766194596082395824375185729628956870974218904739530401550323154944, 883423532389192164791648750371459257913741948437809479060803100646309888, 1766847064778384329583297500742918515827483896875618958121606201292619776, 3533694129556768659166595001485837031654967793751237916243212402585239552, 7067388259113537318333190002971674063309935587502475832486424805170479104, 14134776518227074636666380005943348126619871175004951664972849610340958208, 28269553036454149273332760011886696253239742350009903329945699220681916416, 56539106072908298546665520023773392506479484700019806659891398441363832832, 113078212145816597093331040047546785012958969400039613319782796882727665664, 226156424291633194186662080095093570025917938800079226639565593765455331328, 452312848583266388373324160190187140051835877600158453279131187530910662656, 904625697166532776746648320380374280103671755200316906558262375061821325312, 1809251394333065553493296640760748560207343510400633813116524750123642650624, 3618502788666131106986593281521497120414687020801267626233049500247285301248, 7237005577332262213973186563042994240829374041602535252466099000494570602496, 14474011154664524427946373126085988481658748083205070504932198000989141204992, 28948022309329048855892746252171976963317496166410141009864396001978282409984, 57896044618658097711785492504343953926634992332820282019728792003956564819968, 115792089237316195423570985008687907853269984665640564039457584007913129639936, 231584178474632390847141970017375815706539969331281128078915168015826259279872, 463168356949264781694283940034751631413079938662562256157830336031652518559744, 926336713898529563388567880069503262826159877325124512315660672063305037119488, 1852673427797059126777135760139006525652319754650249024631321344126610074238976, 3705346855594118253554271520278013051304639509300498049262642688253220148477952, 7410693711188236507108543040556026102609279018600996098525285376506440296955904, 14821387422376473014217086081112052205218558037201992197050570753012880593911808, 29642774844752946028434172162224104410437116074403984394101141506025761187823616, 59285549689505892056868344324448208820874232148807968788202283012051522375647232, 118571099379011784113736688648896417641748464297615937576404566024103044751294464, 237142198758023568227473377297792835283496928595231875152809132048206089502588928, 4742843975160471364549467545
```

```
In [95]: print('Best score: ',rs.best_score_)
print('k value with best score: ',rs.best_params_)
print(' ')
print('Train AUC scores')
print(rs.cv_results_['mean_train_score'])
print('CV AUC scores')
print(rs.cv_results_['mean_test_score'])
```

Best score: 0.7417198418225839

k value with best score: {'n_estimators': 200, 'max_depth': 4}

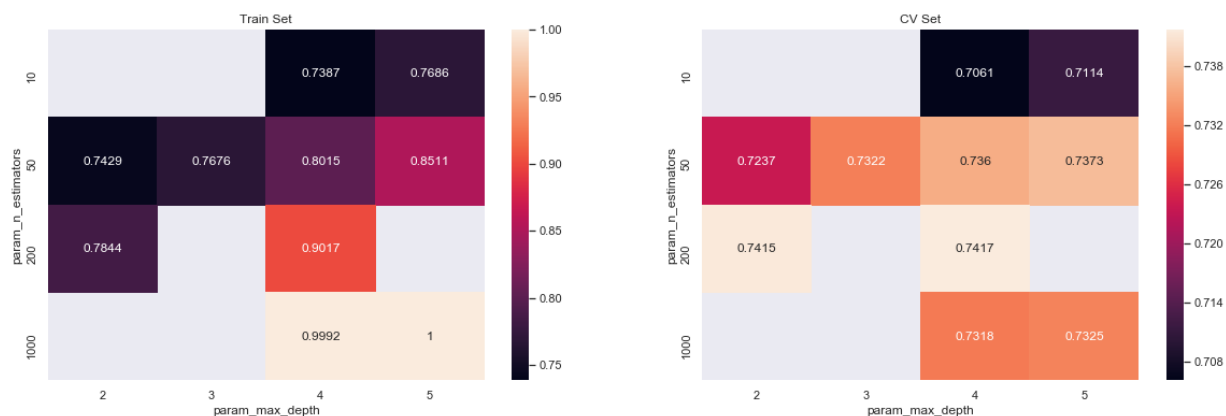
Train AUC scores

```
[0.85107069 0.80154248 0.99921985 0.73867242 0.74290908 0.76855389
 0.78435855 0.99999999 0.76757806 0.9017058 ]
```

CV AUC scores

```
[0.73731885 0.73595594 0.73182936 0.70610419 0.72370543 0.71144459
 0.74148531 0.7325167 0.73221765 0.74171984]
```

```
In [96]: import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(rs.cv_results_).groupby(['param_n_estimators', 'param_
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



```
In [97]: max_d = rs.best_params_['max_depth']
n_est = rs.best_params_['n_estimators']
print(rs.best_params_)
```

```
{'n_estimators': 200, 'max_depth': 4}
```

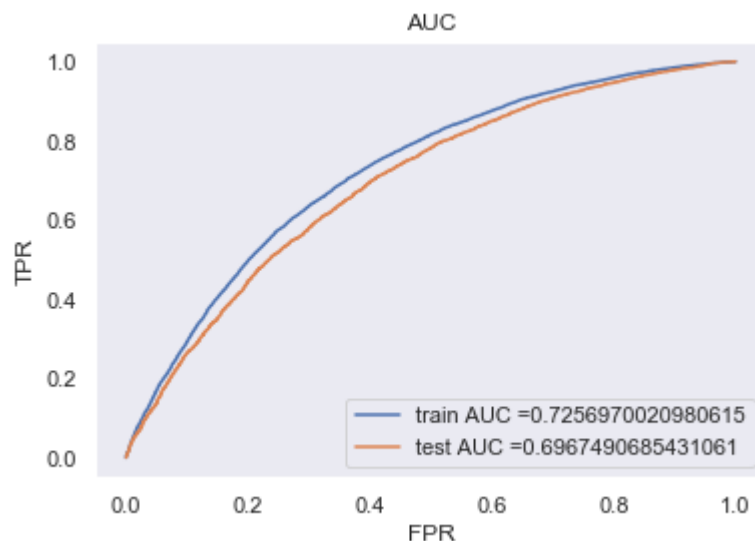
```
In [98]: # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html
from sklearn.metrics import roc_curve, auc
model = RandomForestClassifier(max_depth = max_d, n_estimators = n_est)

model.fit(X_train,y_train_tfidf_w2v)

y_train_pred = pred_prob(model,X_train)
y_test_pred = pred_prob(model,X_test)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train_tfidf_w2v, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test_tfidf_w2v, y_test_pred)

plt.close
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("FPR")
plt.ylabel("TPR")
plt.title("AUC")
plt.grid()
plt.show()
```



```
In [99]: #our objective here is to make auc the maximum
#so we find the best threshold that will give the least fpr
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train_tfidf_w2v, predict_with_best_t(y_train_pred, best_t
```

the maximum value of $tpr \cdot (1 - fpr)$ 0.4484977841822988 for threshold 0.659

Train confusion matrix

```
[[ 7589  3990]
 [ 7394 16027]]
```

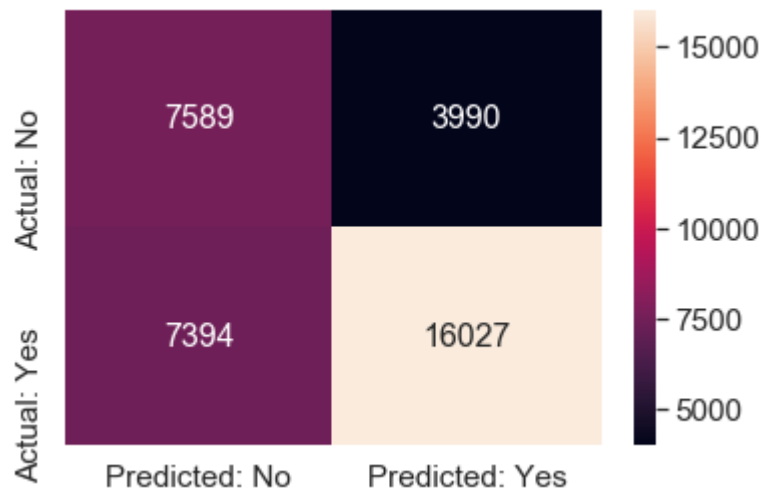
```
In [100]: #plotting confusion matrix using seaborn's heatmap
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

print("Train data confusion matrix")
```

```
confusion_matrix_df_train = pd.DataFrame(confusion_matrix(y_train_tfidf_w2v, pred
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_train, annot=True,annot_kws={"size": 16}, fmt='g
```

Train data confusion matrix

Out[100]: <matplotlib.axes._subplots.AxesSubplot at 0x22cc389c6a0>



```
In [101]: print("Test confusion matrix")
print(confusion_matrix(y_test_tfidf_w2v, predict_with_best_t(y_test_pred, best_t
```

Test confusion matrix

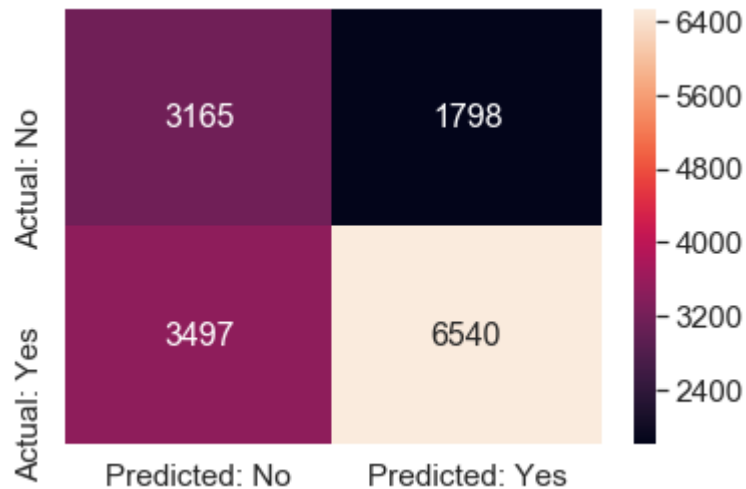
```
[[3165 1798]
 [3497 6540]]
```

```
In [102]: print("Test data confusion matrix")

confusion_matrix_df_test = pd.DataFrame(confusion_matrix(y_test_tfidf_w2v, predic
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

Test data confusion matrix

Out[102]: <matplotlib.axes._subplots.AxesSubplot at 0x22cc45176d8>



3. Conclusions

3.1 GBDT Results

```
In [107]: x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Hyperparameter", "AUC(Train Data)", "AUC(
Test Data)"]

x.add_row(["TFIDF", "XGB00ST", "{ 'n_estimators': 500, 'max_depth': 2}", '0.70', '0.68'])
x.add_row(["TFIDF AVG W2V", "XGB00ST", "{ 'n_estimators': 200, 'max_depth': 4}", '0.72', '0.69'])

print(x)
```

```
+-----+-----+-----+-----+
| Vectorizer | Model | Hyperparameter | AUC(Train Data) | AUC(Test Data) |
+-----+-----+-----+-----+
| TFIDF      | XGB00ST | { 'n_estimators': 500, 'max_depth': 2} | 0.70 | 0.68 |
| TFIDF AVG W2V | XGB00ST | { 'n_estimators': 200, 'max_depth': 4} | 0.72 | 0.69 |
+-----+-----+-----+-----+
```

In []: