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

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

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

```
D:\installed\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWarning: detected Win
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

1.1 Reading Data

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

```
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 (109248, 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']
```

```
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']
```

	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`

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

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

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

from collections import Counter
my_counter = Counter()
```

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

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

1.3 preprocessing of `project_subject_subcategories`

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

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

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

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

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

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

1.3 Text preprocessing

```
# 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)
```

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

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	pr
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	

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

```
# printing some random reviews
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
print("="*50)
print(project_data['essay'].values[20000])
print("="*50)
print(project_data['essay'].values[99999])
print("="*50)
```

```
My students are English learners that are working on English as their second or third la
==================================================
The 51 fifth grade students that will cycle through my classroom this year all love lear
==================================================
How do you remember your days of school? Was it in a sterile environment with plain wall
==================================================
My kindergarten students have varied disabilities ranging from speech and language delay
==================================================
The mediocre teacher tells. The good teacher explains. The superior teacher demonstrates
==================================================
```

```
# 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


sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

```
My kindergarten students have varied disabilities ranging from speech and language delay
==================================================
```

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

```
My kindergarten students have varied disabilities ranging from speech and language delay
```

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

```
My kindergarten students have varied disabilities ranging from speech and language delay
```

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "yo
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they',
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll"
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'h
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'unt
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'dur
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', '
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'bo
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'ver
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'does
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
            'won', "won't", 'wouldn', "wouldn't"]
```

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

```
100%|████████████████████████████████████████████████████████████████████| 109248/10
```

```
# after preprocesing
preprocessed_essays[20000]
```

```
'my kindergarten students varied disabilities ranging speech language delays cognitive d
```

1.4 Preprocessing of `project_title`

```
# similarly you can preprocess the titles also
```

1.5 Preparing data for models

```
project_data.columns
```

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', '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'],
      dtype='object')
```

we are going to consider

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

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

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

1.5.1 Vectorizing Categorical data

- https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorica

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary
categories_one_hot = vectorizer.fit_transform(project_data['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",categories_one_hot.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNee
Shape of matrix after one hot encodig  (109248, 9)
```

```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, bi
sub_categories_one_hot = vectorizer.fit_transform(project_data['clean_subcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",sub_categories_one_hot.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurric
Shape of matrix after one hot encodig  (109248, 30)
```

```
# you can do the similar thing with state, teacher_prefix and project_grade_category also
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

```
# We are considering only the words which appeared in at least 10 documents(rows or projects)
vectorizer = CountVectorizer(min_df=10)
text_bow = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_bow.shape)
```

```
Shape of matrix after one hot encodig  (109248, 16623)
```

```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
```

1.5.2.2 TFIDF vectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
text_tfidf = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (109248, 16623)
```

1.5.2.3 Using Pretrained Models: Avg W2V

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

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

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

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

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

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

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


# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-

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


'''
```

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef
```

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())


# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors.append(vector)

print(len(avg_w2v_vectors))
print(len(avg_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████████| 109248/10
109248
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

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


# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors.append(vector)

print(len(tfidf_w2v_vectors))
print(len(tfidf_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████████| 109248/1
109248
300
```

```
# Similarly you can vectorize for title also
```

1.5.3 Vectorizing Numerical features

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


# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preproce
from sklearn.preprocessing import StandardScaler

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.
# Reshape your data either using array.reshape(-1, 1)

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

# Now standardize the data with above maen and variance.
price_standardized = price_scalar.transform(project_data['price'].values.reshape(-1, 1))


price_standardized
```

```
array([[0.00098843, 0.00191166, 0.00330448, ..., 0.00153418, 0.00046704,
        0.00070265]])
```

1.5.4 Merging all the above features

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

```
print(categories_one_hot.shape)
print(sub_categories_one_hot.shape)
print(text_bow.shape)
print(price_standardized.shape)
```

```
(109248, 9)
(109248, 30)
(109248, 16623)
(109248, 1)
```

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X = hstack((categories_one_hot, sub_categories_one_hot, text_bow, price_standardized))
X.shape
```

```
(109248, 16663)
```

```
# please write all the code with proper documentation, and proper titles for each subsection
# when you plot any graph make sure you use
```

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

__ Computing Sentiment Scores__

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

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

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest stude
for learning my students learn in many different ways using all of our senses and multiple in
of techniques to help all my students succeed students in my class come from a variety of dif
for wonderful sharing of experiences and cultures including native americans our school is a
learners which can be seen through collaborative student project based learning in and out of
in my class love to work with hands on materials and have many different opportunities to pra
mastered having the social skills to work cooperatively with friends is a crucial aspect of t
montana is the perfect place to learn about agriculture and nutrition my students love to rol
in the early childhood classroom i have had several kids ask me can we try cooking with real
and create common core cooking lessons where we learn important math and writing concepts whi
food for snack time my students will have a grounded appreciation for the work that went into
of where the ingredients came from as well as how it is healthy for their bodies this project
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade
and mix up healthy plants from our classroom garden in the spring we will also create our own
shared with families students will gain math and literature skills as well as a life long enj
nannan'
ss = sid.polarity_scores(for_sentiment)

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

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

```
D:\installed\Anaconda3\lib\site-packages\nltk\twitter\__init__.py:20: UserWarning:

The twython library has not been installed. Some functionality from the twitter package

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

Assignment 11: TruncatedSVD

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

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

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

```
import sys
import math

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

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

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

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

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

    def predict_proba(self, X):
        dtest = xgb.DMatrix(X)
        return self.clf.predict(dtest)

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

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

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


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

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

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

# print(clf.grid_scores_)
best_parameters, score, _ = max(clf.grid_scores_, key=lambda x: x[1])
print('score:', score)
for param_name in sorted(best_parameters.keys()):
    print("%s: %r" % (param_name, best_parameters[param_name]))
```

```
score: 0.8333333333333334
colsample_bytree: 0.9
eta: 0.05
max_depth: 6
num_boost_round: 100
subsample: 0.9
```

2. TruncatedSVD

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

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

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import normalize

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_notebook as tqdm1
from tqdm import tqdm
import time
import os

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

from sklearn.model_selection import train_test_split
```

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

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

Text preprocessing(1)

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

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


project_data['clean_categories'] = cat_list
project_data.drop(['project_subject_categories'], axis=1, inplace=True)
project_data.head(5)
```

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


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



# ind = np.arange(len(sorted_cat_dict))
# plt.figure(figsize=(20,5))
# p1 = plt.bar(ind, list(sorted_cat_dict.values()))

# plt.ylabel('Projects')
# plt.title('% of projects aproved category wise')
# plt.xticks(ind, list(sorted_cat_dict.keys()))
# plt.show()
# print(sorted_cat_dict)


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

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

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


project_data['clean_subcategories'] = sub_cat_list
project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)
project_data.head(2)


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

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


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



# ind = np.arange(len(sorted_sub_cat_dict))
# plt.figure(figsize=(20,5))
# p1 = plt.bar(ind, list(sorted_sub_cat_dict.values()))

# plt.ylabel('Projects')
# plt.title('% of projects aproved state wise')
# plt.xticks(ind, list(sorted_sub_cat_dict.keys()))
# plt.show()


# 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)


# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes-for-all-grou
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
price_data.head(2)


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


#presence of the numerical digits in a strings with numeric : https://stackoverflow.com/a/198
def hasNumbers(inputString):
    return any(i.isdigit() for i in inputString)
p1 = project_data[['id','project_resource_summary']]
p1 = pd.DataFrame(data=p1)
p1.columns = ['id','digits_in_summary']
p1['digits_in_summary'] =  p1['digits_in_summary'].map(hasNumbers)
# https://stackoverflow.com/a/17383325/8089731
p1['digits_in_summary'] = p1['digits_in_summary'].astype(int)
project_data = pd.merge(project_data, p1, on='id', how='left')
project_data.head(5)
```

Text preprocessing(2)

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
```

```
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


# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "yo
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they',
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll"
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'h
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'unt
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'dur
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', '
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'bo
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'ver
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'does
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
            'won', "won't", 'wouldn', "wouldn't"]


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


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

```
preprocessed_titles[1000]
```

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

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

project_grade_cat_list = []
for i in tqdm1(project_grade_catogories):
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Ca
        if 'The' in j.split(): # this will split each of the catogory based on space "Math &
            j=j.replace('The','') # if we have the words "The" we are going to replace it wit
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    project_grade_cat_list.append(temp.strip())
```

```
project_data['clean_project_grade_category'] = project_grade_cat_list
project_data.drop(['project_grade_category'], axis=1, inplace=True)
project_data.head(2)
```

```
project_data.drop(['project_essay_1','project_essay_2','project_essay_3','project_essay_4'],
project_data.head(2)
```

```
#Replacing Nan's with maximum occured value: https://stackoverflow.com/a/51053916/8089731
project_data['teacher_prefix'].value_counts().argmax()
project_data.fillna(value=project_data['teacher_prefix'].value_counts().argmax(),axis=1,inpla
```

```
project_data['preprocessed_essays'] = preprocessed_essays
project_data['preprocessed_titles'] = preprocessed_titles
```

```
project_data.columns
```

2.2 Make Data Model Ready: encoding numerical, categorical features

```
X_train, X_test, y_train, y_test = train_test_split(project_data,project_data['project_is_app
# X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=

X_train.drop(['project_is_approved'], axis=1, inplace=True)
X_test.drop(['project_is_approved'], axis=1, inplace=True)
# X_cv.drop(['project_is_approved'], axis=1, inplace=True)
print(X_train.shape)
print(X_test.shape)
```

1.4.1 Vectorizing Categorical data

```
# we use count vectorizer to convert the values into one hot encoded features
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_cat = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, bi
vectorizer_cat.fit(X_train['clean_categories'].values)
print(vectorizer_cat.get_feature_names())


categories_one_hot_train = vectorizer_cat.transform(X_train['clean_categories'].values)
# categories_one_hot_cv = vectorizer_cat.transform(X_cv['clean_categories'].values)
categories_one_hot_test = vectorizer_cat.transform(X_test['clean_categories'].values)
print("Shape of matrix after one hot encodig_train ",categories_one_hot_train.shape)
# print("Shape of matrix after one hot encodig_cv ",categories_one_hot_cv.shape)
print("Shape of matrix after one hot encodig_test ",categories_one_hot_test.shape)
```

```
# we use count vectorizer to convert the values into one hot encoded features
vectorizer_sub_cat = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=F
vectorizer_sub_cat.fit(X_train['clean_subcategories'].values)
print(vectorizer_sub_cat.get_feature_names())


sub_categories_one_hot_train = vectorizer_sub_cat.transform(X_train['clean_subcategories'].va
# sub_categories_one_hot_cv = vectorizer_sub_cat.transform(X_cv['clean_subcategories'].values
sub_categories_one_hot_test = vectorizer_sub_cat.transform(X_test['clean_subcategories'].valu
print("Shape of matrix after one hot encodig_train ",sub_categories_one_hot_train.shape)
# print("Shape of matrix after one hot encodig_cv ",sub_categories_one_hot_cv.shape)
print("Shape of matrix after one hot encodig_test ",sub_categories_one_hot_test.shape)
```

```
# we use count vectorizer to convert the values into one hot encoded features
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_state = CountVectorizer( lowercase=False, binary=True)
vectorizer_state.fit(X_train['school_state'].values)
print(vectorizer_state.get_feature_names())
```

```
school_state_one_hot_train = vectorizer_state.transform(X_train['school_state'].values)
# school_state_one_hot_cv = vectorizer_state.transform(X_cv['school_state'].values)
school_state_one_hot_test = vectorizer_state.transform(X_test['school_state'].values)
print("Shape of matrix after one hot encodig_train ",school_state_one_hot_train.shape)
# print("Shape of matrix after one hot encodig_cv ",school_state_one_hot_cv.shape)
print("Shape of matrix after one hot encodig_test ",school_state_one_hot_test.shape)
```

```
# we use count vectorizer to convert the values into one hot encoded features
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_teacherprefix = CountVectorizer( lowercase=False, binary=True)
vectorizer_teacherprefix.fit(X_train['teacher_prefix'].values.astype('U'))
print(vectorizer_teacherprefix.get_feature_names())

#https://stackoverflow.com/a/39308809/8089731
teacher_prefix_one_hot_train = vectorizer_teacherprefix.transform(X_train['teacher_prefix'].v
# teacher_prefix_one_hot_cv = vectorizer_teacherprefix.transform(X_cv['teacher_prefix'].value
teacher_prefix_one_hot_test = vectorizer_teacherprefix.transform(X_test['teacher_prefix'].val
print("Shape of matrix after one hot encodig_train ",teacher_prefix_one_hot_train.shape)
# print("Shape of matrix after one hot encodig_cv ",teacher_prefix_one_hot_cv.shape)
print("Shape of matrix after one hot encodig_test ",teacher_prefix_one_hot_test[:5,:])
# print(X_train['teacher_prefix'].value_counts())
```

```
print(project_data['clean_project_grade_category'].unique())# we use count vectorizer to conv
from sklearn.feature_extraction.text import CountVectorizer
# https://stackoverflow.com/a/38161028/8089731
pattern = "(?u)\\b[\\w-]+\\b"
vectorizer_projectgrade = CountVectorizer(token_pattern=pattern, lowercase=False, binary=True
vectorizer_projectgrade.fit(X_train['clean_project_grade_category'].values)
print(vectorizer_projectgrade.get_feature_names())

#https://stackoverflow.com/a/39308809/8089731
project_grade_category_one_hot_train = vectorizer_projectgrade.transform(X_train['clean_proje
# project_grade_category_one_hot_cv = vectorizer_projectgrade.transform(X_cv['clean_project_g
project_grade_category_one_hot_test = vectorizer_projectgrade.transform(X_test['clean_project
print("Shape of matrix after one hot encodig_train ",project_grade_category_one_hot_train.sha
# print("Shape of matrix after one hot encodig_cv ",project_grade_category_one_hot_cv.shape)
print("Shape of matrix after one hot encodig_test ",project_grade_category_one_hot_test[:5,:]
```

Vectorizing Numerical features

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preproce
# from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.
# Reshape your data either using array.reshape(-1, 1)

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

# train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)
# Now standardize the data with above maen and variance.
price_standardized_train = normalize(X_train['price'].values.reshape(-1, 1),axis=0)
# price_standardized_cv = price_scalar.transform(X_cv['price'].values.reshape(-1, 1))
price_standardized_test = normalize(X_test['price'].values.reshape(-1, 1),axis=0)
print(price_standardized_train.shape)
# print(price_standardized_cv.shape)
print(price_standardized_test.shape)


# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preproce
# from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.
# Reshape your data either using array.reshape(-1, 1)

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

# Now standardize the data with above maen and variance.
quantity_standardized_train = normalize(X_train['quantity'].values.reshape(-1, 1),axis=0)
# quantity_standardized_cv = quantity_scalar.transform(X_cv['quantity'].values.reshape(-1, 1)
quantity_standardized_test = normalize(X_test['quantity'].values.reshape(-1, 1),axis=0)
print(quantity_standardized_train.shape)
# print(quantity_standardized_cv.shape)
print(quantity_standardized_test.shape)


# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preproce
# from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.
# Reshape your data either using array.reshape(-1, 1)

# teacher_number_of_previously_posted_projects_scalar = StandardScaler()
# teacher_number_of_previously_posted_projects_scalar.fit(X_train['teacher_number_of_previous
# print(f"Mean : {teacher_number_of_previously_posted_projects_scalar.mean_[0]}, Standard dev
# Now standardize the data with above maen and variance.
teacher_number_of_previously_posted_projects_standardized_train = normalize(X_train['teacher_
# teacher_number_of_previously_posted_projects_standardized_cv = teacher_number_of_previously
teacher_number_of_previously_posted_projects_standardized_test = normalize(X_test['teacher_nu
print(teacher_number_of_previously_posted_projects_standardized_train.shape)
# print(teacher_number_of_previously_posted_projects_standardized_cv.shape)
print(teacher_number_of_previously_posted_projects_standardized_test.shape)
```

2.3 Make Data Model Ready: encoding eassay, and project_title

```
X_train.head(2)
```

TFIDF Vectorizer on `project_TEXT/ESSAYS` (Train,Cv,Test)

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf_essays = TfidfVectorizer(min_df=10,max_features=5000,ngram_range=(1,2))
vectorizer_tfidf_essays.fit(X_train['preprocessed_essays'])

text_tfidf_train = vectorizer_tfidf_essays.transform(X_train['preprocessed_essays'])
# text_tfidf_cv = vectorizer_tfidf_essays.transform(X_cv['preprocessed_essays'])
text_tfidf_test = vectorizer_tfidf_essays.transform(X_test['preprocessed_essays'])
print("Shape of matrix after tfidf_text_train ",text_tfidf_train.shape)
# print("Shape of matrix after tfidf_text_cv ",text_tfidf_cv.shape)
print("Shape of matrix after tfidf_text_test ",text_tfidf_test.shape)
```

TFIDF Vectorizer on `project_title` (Train,Cv,Test)

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf_title = TfidfVectorizer(min_df=10)
vectorizer_tfidf_title.fit(X_train['preprocessed_titles'])

title_tfidf_train = vectorizer_tfidf_title.transform(X_train['preprocessed_titles'])
# title_tfidf_cv = vectorizer_tfidf_title.transform(X_cv['preprocessed_titles'])
title_tfidf_test = vectorizer_tfidf_title.transform(X_test['preprocessed_titles'])
print("Shape of matrix after tfidf_title_train ",title_tfidf_train.shape)
# print("Shape of matrix after tfidf_title_cv ",title_tfidf_cv.shape)
print("Shape of matrix after tfidf_title_test ",title_tfidf_test.shape)
```

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

import dill
# dill.dump_session('notebook_env.db')
```

```
dill.load_session('notebook_env.db')
```

```
C:\Users\LENOVO\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWarning: detected
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

```
concat_essays_titles= (list(X_train['preprocessed_essays'])+list(X_test['preprocessed_titles'
```

```
concat_essays_titles[109220]
```

```
'chromebook research'
```

```
len(concat_essays_titles)
```

```
109248
```

```
tf_idf_vectorizer = TfidfVectorizer()
tf_idf_vectorizer.fit_transform(concat_essays_titles)
```

```
<109248x48868 sparse matrix of type '<class 'numpy.float64'>'
	with 7993090 stored elements in Compressed Sparse Row format>
```

```
idf_score = tf_idf_vectorizer.idf_
feature_names  = tf_idf_vectorizer.get_feature_names()
```

```
idf_score_features=[]
for i in range(len(idf_score)):
    idf_score_features.append([idf_score[i],feature_names[i]])
```

```
idf_score_features.sort(reverse=True)
idf_score_features=idf_score_features[:2000]
```

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

2.2 Computing Co-occurance matrix

```
coo_matrix=np.zeros((2000,2000))
window=5
```

```
final_2000_features=[]
for i in range(2000):
    final_2000_features.append(idf_score_features[i][1])
```

```
for sentance in concat_essays_titles:
    word_sen=sentance.split()
    for idss,word in enumerate(word_sen):
        if word in final_2000_features:
            for i in range(max(0,idss-window),min(idss+window,len(word_sen))):
                if word_sen[i] in final_2000_features:
                    coo_matrix[final_2000_features.index(word_sen[i]),final_2000_features.ind
```

```
type(coo_matrix)
```

```
# with open('coo_matrix.pkl','wb') as f:
#     pickle.dump(coo_matrix, f)
```

```
with open('coo_matrix.pkl','rb') as f:
    coo_matri = pickle.load(f)
    print(type(coo_matri))
```

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

```
coo_matrix = np.array(coo_matri)
```

```
(coo_matrix)
```

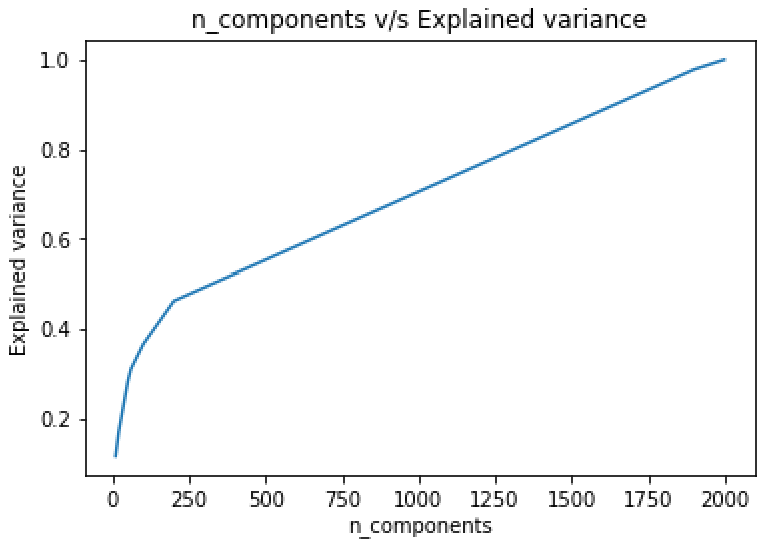
```
array([[1., 0., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 0., 2.]])
```

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

```
# finding optimal value of n_componenets(n) using truncated svd
from sklearn.decomposition import TruncatedSVD
n_components=[10,20,50,60,100,200,300,400,500,1000,1200,1500,1600,1700,1800,1900,1999]
explained_variance=[]
for n in n_components:
    svd=TruncatedSVD(n_components=n,random_state=42)
    svd.fit(coo_matrix)
    exvar=svd.explained_variance_ratio_.sum()
    explained_variance.append(exvar)

    print('n_components=',n,'variance=',exvar)
```

```
n_components= 10 variance= 0.1159590813580738
n_components= 20 variance= 0.16923911371858813
n_components= 50 variance= 0.28315252103920313
n_components= 60 variance= 0.31047651235692136
n_components= 100 variance= 0.3656284135288057
n_components= 200 variance= 0.46206687050301337
n_components= 300 variance= 0.49242876027040444
n_components= 400 variance= 0.5227869453600746
n_components= 500 variance= 0.5531439823006257
n_components= 1000 variance= 0.7049379035686646
n_components= 1200 variance= 0.765651384750285
n_components= 1500 variance= 0.8567388230867455
n_components= 1600 variance= 0.8870934317555152
n_components= 1700 variance= 0.9174524165591598
n_components= 1800 variance= 0.9478132776464756
n_components= 1900 variance= 0.9781688846378389
n_components= 1999 variance= 0.9999999999999809
```

```
#plotting curve between n_components and explained variance
plt.plot(n_components, explained_variance)
plt.xlabel('n_components')
plt.ylabel("Explained variance")
plt.title("n_components v/s Explained variance")
plt.show()
```



```
from sklearn.decomposition import TruncatedSVD

tsvd=TruncatedSVD(n_components=1800,random_state=42)
final_coo_matrix=tsvd.fit_transform(coo_matrix)
```

```
final_coo_matrix.shape
```

```
(2000, 1800)
```

```
final_coo_matrix
```

```
array([[ 1.32840022e-12,  2.68385593e-12,  3.24824011e-12, ...,
        -1.39555451e-02,  4.56617844e-02,  3.25500740e-02],
       [ 1.34762544e-12, -7.07120759e-12, -9.48902320e-12, ...,
         1.45315793e-02, -8.68778904e-03,  3.49422371e-03],
       [ 8.32975281e-13,  2.55641967e-12, -1.38707945e-11, ...,
         5.35582481e-02, -2.89401347e-02,  2.52105460e-02],
       ...,
       [ 2.99595591e-12,  1.49595751e-12, -1.40947788e-11, ...,
         1.82719563e-02, -2.54953153e-02,  1.33273143e-02],
       [-1.21354773e-12, -1.99786334e-12,  2.71971481e-11, ...,
        -1.58616926e-02, -2.59020772e-02, -2.16188073e-02],
       [-1.47697393e-16,  1.51848033e-16,  8.46502386e-15, ...,
        -2.06532044e-17, -3.66672093e-17,  8.64100661e-18]])
```

```
final_coo_matrix[0]
```

```
array([ 1.32840022e-12,  2.68385593e-12,  3.24824011e-12, ...,
       -1.39555451e-02,  4.56617844e-02,  3.25500740e-02])
```

```
model = {}
for i in range(len(final_2000_features)):
    model[final_2000_features[i]] = final_coo_matrix[i]
```

```
# model = final_2000_features
glove_words =  set(model.keys())
```

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_titles_vectors_train = []; # the avg-w2v for each sentence/review is stored in this l
for sentence in tqdm1(X_train['preprocessed_titles']): # for each review/sentence
    vector = np.zeros(1800) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_titles_vectors_train.append(vector)
```

```
HBox(children=(IntProgress(value=0, max=73196), HTML(value='')))
```

```
avg_w2v_essays_vectors_train = []; # the avg-w2v for each sentence/review is stored in this l
for sentence in tqdm1(X_train['preprocessed_essays']): # for each review/sentence
```

```
    vector = np.zeros(1800) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essays_vectors_train.append(vector)
```

```
HBox(children=(IntProgress(value=0, max=73196), HTML(value='')))
```

```
avg_w2v_titles_vectors_test = []; # the avg-w2v for each sentence/review is stored in this li
for sentence in tqdm1(X_test['preprocessed_titles']): # for each review/sentence
    vector = np.zeros(1800) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_titles_vectors_test.append(vector)
```

```
HBox(children=(IntProgress(value=0, max=36052), HTML(value='')))
```

```
avg_w2v_essays_vectors_test = []; # the avg-w2v for each sentence/review is stored in this li
for sentence in tqdm1(X_test['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(1800) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essays_vectors_test.append(vector)
```

```
HBox(children=(IntProgress(value=0, max=36052), HTML(value='')))
```

2.4 Merge the features from step 3 and step 4

Word counts(TITLES)

```
title_wordcount_train = []
```

```
title_train = list(X_train['preprocessed_titles'])
for i in tqdm1(title_train):
    b = len(str(i).split())
    title_wordcount_train.append(b)
title_wordcount_train = np.array(title_wordcount_train)

title_wordcount_test = []
title_test = list(X_test['preprocessed_titles'])
for i in tqdm1(title_test):
    b = len(str(i).split())
    title_wordcount_test.append(b)
title_wordcount_test = np.array(title_wordcount_test)

print(title_wordcount_train.shape)
print(title_wordcount_test.shape)
```

```
HBox(children=(IntProgress(value=0, max=73196), HTML(value='')))

HBox(children=(IntProgress(value=0, max=36052), HTML(value='')))

(73196,)
(36052,)
```

Standardizing Word counts(TITLES)

```
from sklearn.preprocessing import StandardScaler

title_wordcount_scalar = StandardScaler()
title_wordcount_scalar.fit(title_wordcount_train.reshape(-1,1))

title_wordcount_standardized_train = title_wordcount_scalar.transform(title_wordcount_train.r
title_wordcount_standardized_test = title_wordcount_scalar.transform(title_wordcount_test.res

print(title_wordcount_standardized_train.shape)
print(title_wordcount_standardized_test.shape)
```

```
(73196, 1)
(36052, 1)
```

Word counts(ESSAYS)

```
essay_wordcount_train = []
essay_train = list(X_train['preprocessed_essays'])
for i in tqdm1(essay_train):
    b = len(str(i).split())
    essay_wordcount_train.append(b)
essay_wordcount_train = np.array(essay_wordcount_train)
```

```
essay_wordcount_test = []
essay_test = list(X_test['preprocessed_titles'])
for i in tqdm1(essay_test):
    b = len(str(i).split())
    essay_wordcount_test.append(b)
essay_wordcount_test = np.array(essay_wordcount_test)

print(essay_wordcount_train.shape)
print(essay_wordcount_test.shape)
```

```
HBox(children=(IntProgress(value=0, max=73196), HTML(value='')))

HBox(children=(IntProgress(value=0, max=36052), HTML(value='')))

(73196,)
(36052,)
```

Standardizing Word counts(ESSAYS)

```
from sklearn.preprocessing import StandardScaler

essay_wordcount_scalar = StandardScaler()
essay_wordcount_scalar.fit(essay_wordcount_train.reshape(-1,1))

essay_wordcount_standardized_train = essay_wordcount_scalar.transform(essay_wordcount_train.r
essay_wordcount_standardized_test = essay_wordcount_scalar.transform(essay_wordcount_test.res

print(essay_wordcount_standardized_train.shape)
print(essay_wordcount_standardized_test.shape)
```

```
(73196, 1)
(36052, 1)
```

Sentiment scores for each essay

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

import nltk
nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()

essay_sentscore_train = []
essay_train = list(X_train['preprocessed_essays'])
for i in tqdm1(essay_train):
    ss = sid.polarity_scores(str(i))
    essay_sentscore_train.append(ss)
essay_sentscore_train = np.array(essay_sentscore_train)

essay_negscore_train = []
essay_neuscore_train = []
essay_posscore_train = []
essay_compoundscore_train = []
for it in essay_sentscore_train:
    a = it['neg']
    essay_negscore_train.append(a)
    b = it['neu']
    essay_neuscore_train.append(b)
    c = it['pos']
    essay_posscore_train.append(c)
    d = it['compound']
    essay_compoundscore_train.append(d)

essay_negscore_train = np.array(essay_negscore_train).reshape(-1,1)
essay_neuscore_train = np.array(essay_neuscore_train).reshape(-1,1)
essay_posscore_train = np.array(essay_posscore_train).reshape(-1,1)
essay_compoundscore_train = np.array(essay_compoundscore_train).reshape(-1,1)

print((essay_negscore_train.shape))
print((essay_neuscore_train.shape))
print((essay_posscore_train.shape))
print((essay_compoundscore_train.shape))


##############################################################

essay_sentscore_test = []
essay_test = list(X_test['preprocessed_essays'])
for i in tqdm1(essay_test):
    ss = sid.polarity_scores(str(i))
    essay_sentscore_test.append(ss)
essay_sentscore_test = np.array(essay_sentscore_test)

essay_negscore_test = []
essay_neuscore_test = []
essay_posscore_test = []
essay_compoundscore_test = []
for it in essay_sentscore_test:
    a = it['neg']
    essay_negscore_test.append(a)
    b = it['neu']
    essay_neuscore_test.append(b)
    c = it['pos']
    essay_posscore_test.append(c)
    d = it['compound']
    essay_compoundscore_test.append(d)
```

```
essay_negscore_test = np.array(essay_negscore_test).reshape(-1,1)
essay_neuscore_test = np.array(essay_neuscore_test).reshape(-1,1)
essay_posscore_test = np.array(essay_posscore_test).reshape(-1,1)
essay_compoundscore_test = np.array(essay_compoundscore_test).reshape(-1,1)

print((essay_negscore_test.shape))
print((essay_neuscore_test.shape))
print((essay_posscore_test.shape))
print((essay_compoundscore_test.shape))
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /home/dileep_teja3/nltk_data...
HBox(children=(IntProgress(value=0, max=73196), HTML(value='')))

(73196, 1)
(73196, 1)
(73196, 1)
(73196, 1)
HBox(children=(IntProgress(value=0, max=36052), HTML(value='')))

(36052, 1)
(36052, 1)
(36052, 1)
(36052, 1)
```

```
final_coo_matrix.shape
```

```
(2000, 1800)
```

```
categories_one_hot_train.shape
```

```
(73196, 9)
```

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

X_tr = hstack((categories_one_hot_train,sub_categories_one_hot_train,school_state_one_hot_tra
               ,project_grade_category_one_hot_train,price_standardized_train,quantity_standa
               ,teacher_number_of_previously_posted_projects_standardized_train,essay_negscor
               ,essay_compoundscore_train,title_wordcount_standardized_train
               ,essay_wordcount_standardized_train,avg_w2v_titles_vectors_train,avg_w2v_essay

X_te = hstack((categories_one_hot_test,sub_categories_one_hot_test,school_state_one_hot_test,
               ,project_grade_category_one_hot_test,price_standardized_test,quantity_standard
               ,teacher_number_of_previously_posted_projects_standardized_test,essay_negscore
               ,essay_compoundscore_test,title_wordcount_standardized_test
               ,essay_wordcount_standardized_test,avg_w2v_titles_vectors_test,avg_w2v_essays_

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

```
(73196, 3708) (73196,)
(36052, 3708) (36052,)
========================================================================================
```

2.5 Apply XGBoost on the Final Features from the above section

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

```
from sklearn.model_selection import GridSearchCV
import xgboost as xgb
import time

start_time = time.time()
gbdt = xgb.XGBClassifier(n_jobs=-1,class_weight='balanced')
parameters = {'n_estimators': [10, 100, 500], 'max_depth':[10, 50, 100, 500]}
clf = GridSearchCV(gbdt, parameters, cv= 3, scoring='roc_auc',return_train_score=True)
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
print("Execution time: " + str((time.time() - start_time)) + ' ms')
```

```
Execution time: 1624.1976990699768 ms
```

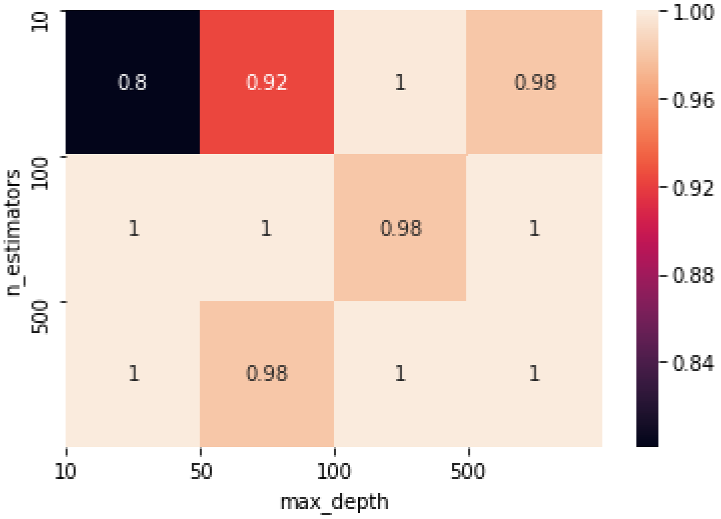
```
import dill
# dill.dump_session('notebook_env11.db')
dill.load_session('notebook_env11.db')
```

```
train_auc = train_auc.reshape(3,4)
cv_auc = cv_auc.reshape(3,4)
train_auc
cv_auc
```

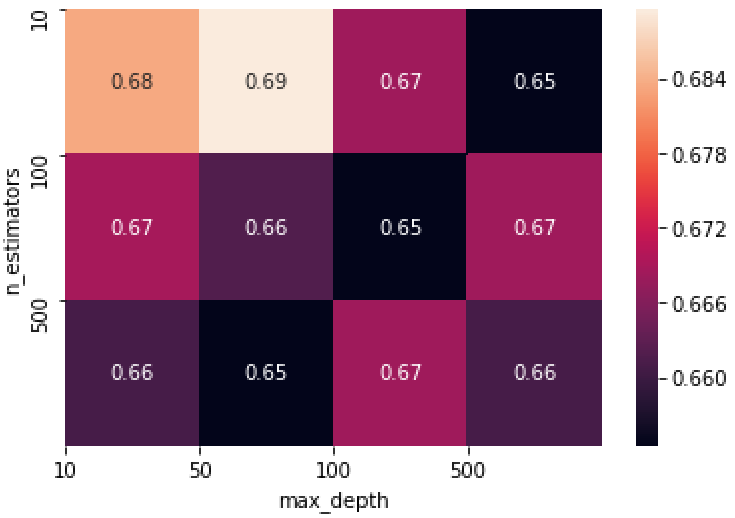
```
array([[0.68351461, 0.68952022, 0.66843937, 0.65448579],
       [0.66898794, 0.66197499, 0.65448579, 0.66840205],
       [0.66095329, 0.65448579, 0.66840205, 0.66095329]])
```

```
import matplotlib.pyplot as plt
# plt.show()
```

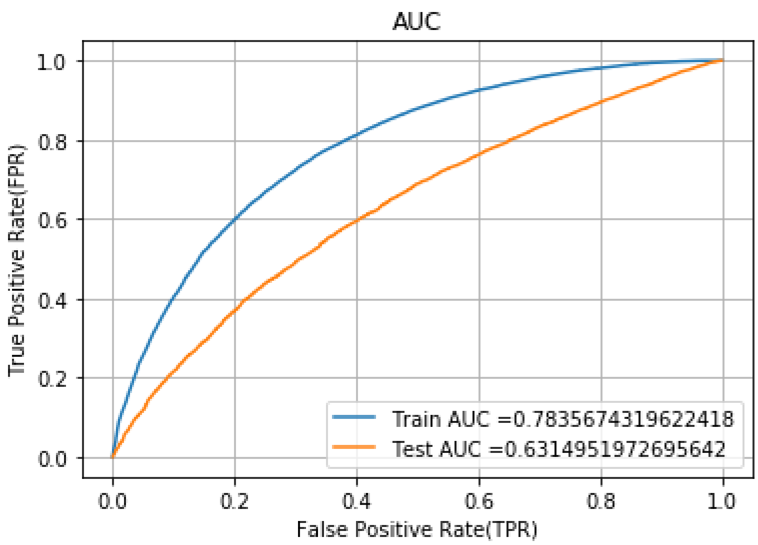
```
import numpy as np; np.random.seed(0)
import seaborn as sns


sns.heatmap(train_auc,annot=True)

plt.yticks(np.arange(3), [10, 100, 500])
plt.xticks(np.arange(4), [10, 50, 100, 500])

plt.xlabel('max_depth')
plt.ylabel('n_estimators')


plt.show()
```



```
import matplotlib.pyplot as plt
# plt.show()

import numpy as np; np.random.seed(0)
import seaborn as sns


sns.heatmap(cv_auc,annot=True)

plt.yticks(np.arange(3), [10, 100, 500])
plt.xticks(np.arange(4), [10, 50, 100, 500])

plt.xlabel('max_depth')
plt.ylabel('n_estimators')


plt.show()
```



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

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


from sklearn.metrics import roc_curve, auc

gbdt = xgb.XGBClassifier(max_depth = 10, n_estimators = 10,n_jobs=-1,class_weight='balanced')
gbdt.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the pos
# not the predicted outputs

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

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```

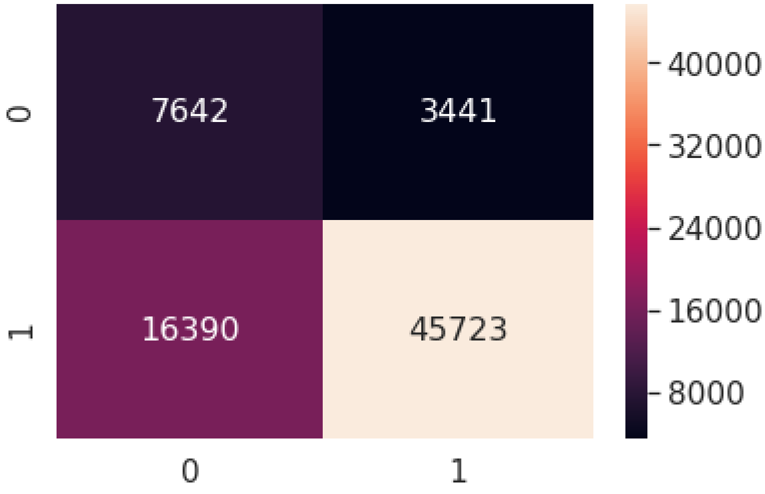


```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

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

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

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


print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train[:], predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)
print("Test confusion matrix")
print(confusion_matrix(y_test[:], predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))
```

```
==========================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.5075769738266626 for threshold 0.713
[[ 7642  3441]
 [16390 45723]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.35821232052725976 for threshold 0.741
[[ 5316   143]
 [28497  2096]]
```

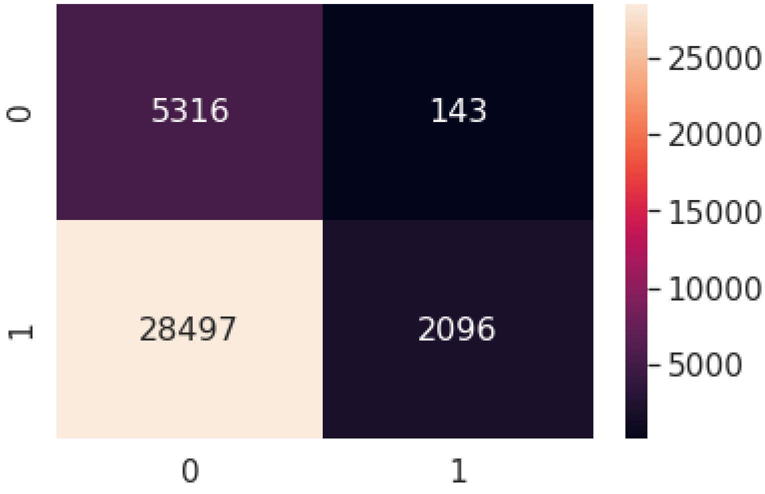
```
conf_matr_df_train =  pd.DataFrame(confusion_matrix(y_train[:], predict(y_train_pred, tr_thre
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.5075769738266626 for threshold 0.713
<matplotlib.axes._subplots.AxesSubplot at 0x7f48da7fffd0>
```



```
conf_matr_df_test =  pd.DataFrame(confusion_matrix(y_test[:], predict(y_test_pred, tr_thresho
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.35821232052725976 for threshold 0.741
<matplotlib.axes._subplots.AxesSubplot at 0x7f48da769c88>
```



3. Conclusion

```
from prettytable import PrettyTable
x = PrettyTable()
x.field names = ["Vectorizer", "Model", "Hyper Parameter", "AUC"]
```

```
x.add_row(["AVG W2V", "XGBoost", "Max Depth:10 , n_estimators:10", 0.63])
print(x)
```

```
+------------+---------+--------------------------------+------+
| Vectorizer |  Model  |         Hyper Parameter        | AUC  |
+------------+---------+--------------------------------+------+
|  AVG W2V   | XGBoost | Max Depth:10 , n_estimators:10 | 0.63 |
+------------+---------+--------------------------------+------+
```