DonorsChoose

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of funding. Right now, a large number of volunteers is needed to manually screen each submission before it's approved to be posted on the DonorsChoose.org website.

Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, there are three main problems they need to solve:

- How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as quickly and as efficiently as possible
- How to increase the consistency of project vetting across different volunteers to improve the experience for teachers
- How to focus volunteer time on the applications that need the most assistance

The goal of the competition is to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved, using the text of project descriptions as well as additional metadata about the project, teacher, and school. DonorsChoose.org can then use this information to identify projects most likely to need further review before approval.

About the DonorsChoose Data Set

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

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located ([Two-letter U.S. postal code](https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_codes)). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy` • `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy materials to manage sensory needs!`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*

Feature	Description
`project_essay_4`	Fourth application essay
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teacher. **Example:** `2`

[*] See the section **Notes on the Essay Data** for more details about these features.

Additionally, the `resources.csv` data set provides more data about the resources required for each project. Each line in this file represents a resource required by a project:

Feature	Description
`id`	A `project_id` value from the `train.csv` file. **Example:** `p036502`
`description`	Desciption of the resource. **Example:** `Tenor Saxophone Reeds, Box of 25`
`quantity`	Quantity of the resource required. **Example:** `3`
`price`	Price of the resource required. **Example:** `9.95`

Note: Many projects require multiple resources. The `id` value corresponds to a `project_id` in train.csv, so you use it as a key to retrieve all resources needed for a project:

The data set contains the following label (the value you will attempt to predict):

Label	Description
`project_is_approved`	A binary flag indicating whether DonorsChoose approved the project. A value of `0` indicates the project was not approved, and a value of `1` indicates the project was approved.

Notes on the Essay Data

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

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

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

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

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

In [1]:

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

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

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

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

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

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

from tqdm import tqdm
import os


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

1.1 Reading Data

In [2]:

```
project_data = pd.read_csv('train_data.csv',nrows=50000)
resource_data = pd.read_csv('resources.csv')
```

In [3]:

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

```
Number of data points in train data (50000, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]:

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

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

Out[4]:

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

1.2 preprocessing of `project_subject_categories`

In [5]:

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

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

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [6]:

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

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

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

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

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

```
    my_counter.update(word.split())

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

In [7]:

```
# We need to get rid of The spaces between the text and the hyphens because they're special charac
ters.
#Rmoving multiple characters from a string in Python
#https://stackoverflow.com/questions/3411771/multiple-character-replace-with-python

project_grade_category = []

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

In [8]:

```
project_data.drop(['project_grade_category'], axis = 1, inplace = True)
project_data["project_grade_category"] = project_grade_category
print("After removing the special characters ,Column values:  ")
np.unique(project_data["project_grade_category"].values)
```

```
After removing the special characters ,Column values:
```

Out[8]:

```
array(['Grades_3_5', 'Grades_6_8', 'Grades_9_12', 'Grades_PreK_2'],
      dtype=object)
```

In [9]:

```
#NaN values in techer prefix will create a problem while encoding,so we replace NaN values with th
e mode of that particular column
#removing dot(.) since it is a special character
mode_of_teacher_prefix = project_data['teacher_prefix'].value_counts().index[0]


project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna(mode_of_teacher_prefix)
```

In [10]:

```
prefixes = []

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

In [11]:

```
project_data.drop(['teacher_prefix'], axis = 1, inplace = True)
project_data["teacher_prefix"] = prefixes
print("After removing the special characters ,Column values:  ")
np.unique(project_data["teacher_prefix"].values)
```

```
After removing the special characters ,Column values:
```

Out[11]:

```
array(['Dr', 'Mr', 'Mrs', 'Ms', 'Teacher'], dtype=object)
```

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]:

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

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [14]:

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

```
My kindergarten students have varied disabilities ranging from speech and language delays, cogniti
ve delays, gross/fine motor delays, to autism. They are eager beavers and always strive to work th
eir hardest working past their limitations. \r\n\r\nThe materials we have are the ones I seek out
for my students. I teach in a Title I school where most of the students receive free or reduced pr
ice lunch.  Despite their disabilities and limitations, my students love coming to school and come
eager to learn and explore.Have you ever felt like you had ants in your pants and you needed to gr
oove and move as you were in a meeting? This is how my kids feel all the time. The want to be able
to move as they learn or so they say.Wobble chairs are the answer and I love then because they dev
elop their core, which enhances gross motor and in Turn fine motor skills. \r\nThey also want to l
earn through games, my kids do not want to sit and do worksheets. They want to learn to count by j
umping and playing. Physical engagement is the key to our success. The number toss and color and s
hape mats can make that happen. My students will forget they are doing work and just have the fun
a 6 year old deserves.nannan
==================================================
```

In [15]:

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

```
My kindergarten students have varied disabilities ranging from speech and language delays, cogniti
ve delays, gross/fine motor delays, to autism. They are eager beavers and always strive to work th
eir hardest working past their limitations.     The materials we have are the ones I seek out for
my students. I teach in a Title I school where most of the students receive free or reduced price
lunch.  Despite their disabilities and limitations, my students love coming to school and come eag
er to learn and explore.Have you ever felt like you had ants in your pants and you needed to groov
e and move as you were in a meeting? This is how my kids feel all the time. The want to be able to
move as they learn or so they say.Wobble chairs are the answer and I love then because they develo
p their core, which enhances gross motor and in Turn fine motor skills.   They also want to learn t
hrough games, my kids do not want to sit and do worksheets. They want to learn to count by jumping
and playing. Physical engagement is the key to our success. The number toss and color and shape ma
ts can make that happen. My students will forget they are doing work and just have the fun a 6 yea
r old deserves.nannan
```

In [16]:

```
#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 delays cognitiv
e delays gross fine motor delays to autism They are eager beavers and always strive to work their
hardest working past their limitations The materials we have are the ones I seek out for my studen
ts I teach in a Title I school where most of the students receive free or reduced price lunch
Despite their disabilities and limitations my students love coming to school and come eager to lea
rn and explore Have you ever felt like you had ants in your pants and you needed to groove and mov
e as you were in a meeting This is how my kids feel all the time The want to be able to move as th
ey learn or so they say Wobble chairs are the answer and I love then because they develop their co
re which enhances gross motor and in Turn fine motor skills They also want to learn through games
my kids do not want to sit and do worksheets They want to learn to count by jumping and playing Ph
ysical engagement is the key to our success The number toss and color and shape mats can make that
happen My students will forget they are doing work and just have the fun a 6 year old deserves nan
nan
```

In [17]:

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

In [18]:

```
#convert all the words to lower case first and then remove the stopwords
for i in range(len(project_data['essay'].values)):
    project_data['essay'].values[i] = project_data['essay'].values[i].lower()
```

In [19]:

```
# 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 = sent.replace('nan',' ')
    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())
```

```
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████| 50000/50000 [00:20<00:00, 2479.35it/s]
```

In [20]:

```
#creating a new column with the preprocessed essays and replacing it with the original columns
project_data['preprocessed_essays'] = preprocessed_essays
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
```

In [21]:

```
essay_word_count=[]
for i in range(len(project_data['preprocessed_essays'])):
    essay_word_count.append(len(project_data['preprocessed_essays'][i].split()))
```

In [22]:

```
project_data['essay_word_count'] = essay_word_count
```

1.4 Preprocessing of `project_title`

In [23]:

```
#convert all the words to lower case first and then remove the stopwords
for i in range(len(project_data['project_title'].values)):
    project_data['project_title'].values[i] = project_data['project_title'].values[i].lower()
```

In [24]:

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

```
100%|██████████| 50000/50000 [00:00<00:00, 57706.82it/s]
```

In [25]:

```
#creating a new column with the preprocessed titles,useful for analysis
project_data['preprocessed_titles'] = preprocessed_titles
```

In [26]:

```
title_word_count=[]
for i in range(len(project_data['preprocessed_titles'])):
    title_word_count.append(len(project_data['preprocessed_titles'][i].split()))
```

In [27]:

```
project_data['title_word_count'] = title_word_count
```

In [28]:

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

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

```
100%|██████████| 50000/50000 [01:00<00:00, 832.18it/s]
```

In [29]:

```
#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
```

Splitting data into Train and test: Stratified Sampling

In [30]:

```
# train test split

from sklearn.model_selection import train_test_split

project_data_train, project_data_test, y_train, y_test = train_test_split(project_data, project_dat
a['project_is_approved'], test_size=0.33, stratify = project_data['project_is_approved'])
```

In [31]:

```
print("Split ratio")
print('-'*50)
print('Train dataset:',len(project_data_train)/len(project_data)*100,'%\n','size:',len(project_data
_train))
print('Test dataset:',len(project_data_test)/len(project_data)*100,'%\n','size:',len(project_data_t
est))
```

```
Split ratio
--------------------------------------------------
Train dataset: 67.0 %
 size: 33500
Test dataset: 33.0 %
 size: 16500
```

In [32]:

```
#Features
project_data_train.drop(['project_is_approved'], axis=1, inplace=True)

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

1.5 Preparing data for models

In [33]:

```
project_data.columns
```

Out[33]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'school_state',
       'project_submitted_datetime', 'project_title',
       'project_resource_summary',
```

```
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'project_grade_category',
       'teacher_prefix', 'essay', 'preprocessed_essays', 'essay_word_count',
       'preprocessed_titles', 'title_word_count', '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
```

Make Data Model Ready: vectorizing numerical, categorical features (with response coding)

Make Data Model Ready: encoding eassay, and project_title

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [34]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer_bow_essay = CountVectorizer(min_df=10)
vectorizer_bow_essay.fit(project_data_train['preprocessed_essays'].values)   #Fitting has to be on
Train data


train_essay_bow = vectorizer_bow_essay.transform(project_data_train['essay'].values)

test_essay_bow = vectorizer_bow_essay.transform(project_data_test['essay'].values)


print("Shape of train data matrix after one hot encoding ",train_essay_bow.shape)

print("Shape of test data matrix after one hot encoding ",test_essay_bow.shape)
```

```
Shape of train data matrix after one hot encoding  (33500, 10329)
Shape of test data matrix after one hot encoding  (16500, 10329)
```

In [35]:

```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
vectorizer_bow_title = CountVectorizer(min_df=10)
vectorizer_bow_title.fit_transform(project_data_train['preprocessed_titles'].values)     #Fitting ha
s to be on Train data


train_title_bow = vectorizer_bow_title.transform(project_data_train['preprocessed_titles'].values)

test_title_bow = vectorizer_bow_title.transform(project_data_test['preprocessed_titles'].values)
```

```
print("Shape of train data matrix after one hot encoding ",train_title_bow.shape)

print("Shape of test data matrix after one hot encoding ",test_title_bow.shape)
```

```
Shape of train data matrix after one hot encoding  (33500, 1538)
Shape of test data matrix after one hot encoding  (16500, 1538)
```

1.5.2.2 TFIDF vectorizer

In [36]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf_essay = TfidfVectorizer(min_df=10)
vectorizer_tfidf_essay.fit(project_data_train['preprocessed_essays'])      #Fitting has to be on
Train data

train_essay_tfidf = vectorizer_tfidf_essay.transform(project_data_train['preprocessed_essays'].val
ues)

test_essay_tfidf =
vectorizer_tfidf_essay.transform(project_data_test['preprocessed_essays'].values)

print("Shape of train data matrix after one hot encoding ",train_essay_tfidf.shape)

print("Shape of test data matrix after one hot encoding ",test_essay_tfidf.shape)
```

```
Shape of train data matrix after one hot encoding  (33500, 10329)
Shape of test data matrix after one hot encoding  (16500, 10329)
```

In [37]:

```
vectorizer_tfidf_title = TfidfVectorizer(min_df=10)
vectorizer_tfidf_title.fit(project_data_train['preprocessed_titles'])      #Fitting has to be on
Train data

train_title_tfidf = vectorizer_tfidf_title.transform(project_data_train['preprocessed_titles'].val
ues)

test_title_tfidf =
vectorizer_tfidf_title.transform(project_data_test['preprocessed_titles'].values)

print("Shape of train data matrix after one hot encoding ",train_title_tfidf.shape)

print("Shape of test data matrix after one hot encoding ",test_title_tfidf.shape)
```

```
Shape of train data matrix after one hot encoding  (33500, 1538)
Shape of test data matrix after one hot encoding  (16500, 1538)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [38]:

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

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

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

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

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

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

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

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


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

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


'''
```

Out[38]:

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

In [39]:

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

In [40]:

```
# average Word2Vec
# compute average word2vec for each review.
train_avg_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
```

```python
for sentence in tqdm(project_data_train['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
    train_avg_w2v_essays.append(vector)

print(len(train_avg_w2v_essays))
print(len(train_avg_w2v_essays[0]))
```

```
100%|██████████| 33500/33500 [00:06<00:00, 5499.96it/s]
```

```
33500
300
```

In [41]:

```python
# average Word2Vec
# compute average word2vec for each review.
test_avg_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(project_data_test['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
    test_avg_w2v_essays.append(vector)

print(len(test_avg_w2v_essays))
print(len(test_avg_w2v_essays[0]))
```

```
100%|██████████| 16500/16500 [00:03<00:00, 5414.87it/s]
```

```
16500
300
```

In [42]:

```python
# average Word2Vec
# compute average word2vec for each review.
train_avg_w2v_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(project_data_train['preprocessed_titles']): # 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
    train_avg_w2v_titles.append(vector)

print(len(train_avg_w2v_titles))
print(len(train_avg_w2v_titles[0]))
```

```
100%|██████████| 33500/33500 [00:00<00:00, 95552.04it/s]
```

```
33500
300
```

In [43]:

```
# average Word2Vec
# compute average word2vec for each review.
test_avg_w2v_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(project_data_test['preprocessed_titles']): # 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
    test_avg_w2v_titles.append(vector)

print(len(test_avg_w2v_titles))
print(len(test_avg_w2v_titles[0]))
```

```
100%|██████████| 16500/16500 [00:00<00:00, 97571.52it/s]
```

```
16500
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [44]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(project_data_train['preprocessed_essays'].values)
# 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 [45]:

```
# average Word2Vec
# compute average word2vec for each review.
train_tfidf_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(project_data_train['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.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf 
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_tfidf_w2v_essays.append(vector)

print(len(train_tfidf_w2v_essays))
print(len(train_tfidf_w2v_essays[0]))
```

```
100%|██████████| 33500/33500 [00:42<00:00, 788.50it/s]
```

```
33500
300
```

In [46]:

In [46]:

```
# average Word2Vec
# compute average word2vec for each review.
test_tfidf_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(project_data_test['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.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_tfidf_w2v_essays.append(vector)

print(len(test_tfidf_w2v_essays))
print(len(test_tfidf_w2v_essays[0]))
```

```
100%|██████████| 16500/16500 [00:20<00:00, 803.77it/s]
```

```
16500
300
```

In [47]:

```
# Similarly you can vectorize for title also
tfidf_model = TfidfVectorizer()
tfidf_model.fit(project_data_train['preprocessed_titles'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [48]:

```
# average Word2Vec
# compute average word2vec for each review.
train_tfidf_w2v_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(project_data_train['preprocessed_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_tfidf_w2v_titles.append(vector)

print(len(train_tfidf_w2v_titles))
print(len(train_tfidf_w2v_titles[0]))
```

```
100%|██████████| 33500/33500 [00:00<00:00, 47888.03it/s]
```

```
33500
300
```

In [49]:

```
# average Word2Vec
# compute average word2vec for each review.
test_tfidf_w2v_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(project_data_test['preprocessed_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_tfidf_w2v_titles.append(vector)

print(len(test_tfidf_w2v_titles))
print(len(test_tfidf_w2v_titles[0]))
```

```
100%|██████████| 16500/16500 [00:00<00:00, 46054.69it/s]
```

```
16500
300
```

1.5.3 Vectorizing Numerical features

In [50]:

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

In [51]:

```
project_data_train = pd.merge(project_data_train, price_data, on='id', how='left')

project_data_test = pd.merge(project_data_test, price_data, on='id', how='left')
```

In [52]:

```
from sklearn.preprocessing import Normalizer
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

normalizer = Normalizer()
normalizer.fit(project_data_train['price'].values.reshape(1,-1))


price_normalized_train = normalizer.transform(project_data_train['price'].values.reshape(1, -1))

price_normalized_test = normalizer.transform(project_data_test['price'].values.reshape(1, -1))
#reshaping again after normalization

price_normalized_train = price_normalized_train.reshape(-1, 1)
price_normalized_test =  price_normalized_test.reshape(-1, 1)

print('After normalization')
print(price_normalized_train.shape)

print(price_normalized_test.shape)
```

```
After normalization
(33500, 1)
(16500, 1)
```

In [53]:

```
normalizer = Normalizer()
normalizer.fit(project_data_train['quantity'].values.reshape(1,-1))


quantity_normalized_train = normalizer.transform(project_data_train['quantity'].values.reshape(1, -
1))

quantity_normalized_test = normalizer.transform(project_data_test['quantity'].values.reshape(1, -1)
)

#reshaping again after normalization

quantity_normalized_train = quantity_normalized_train.reshape(-1,1)
quantity_normalized_test = quantity_normalized_test.reshape(-1,1)

print('After normalization')
print(quantity_normalized_train.shape)

print(quantity_normalized_test.shape)
```

```
After normalization
(33500, 1)
(16500, 1)
```

In [54]:

```
normalizer = Normalizer()
normalizer.fit(project_data_train['teacher_number_of_previously_posted_projects'].values.reshape(1
,-1))


previously_posted_projects_normalized_train =
normalizer.transform(project_data_train['teacher_number_of_previously_posted_projects'].values.res
hape(1, -1))

previously_posted_projects_normalized_test =
normalizer.transform(project_data_test['teacher_number_of_previously_posted_projects'].values.resh
ape(1, -1))

#reshaping again after normalization

previously_posted_projects_normalized_train = previously_posted_projects_normalized_train.reshape(
-1,1)
previously_posted_projects_normalized_test = previously_posted_projects_normalized_test.reshape(-1
,1)

print('After normalization')
print(previously_posted_projects_normalized_train.shape)

print(previously_posted_projects_normalized_test.shape)
```

```
After normalization
(33500, 1)
(16500, 1)
```

In [55]:

```
normalizer = Normalizer()
normalizer.fit(project_data_train['essay_word_count'].values.reshape(-1,1))


essay_word_count_normalized_train = normalizer.transform(project_data_train['essay_word_count'].va
lues.reshape(1, -1))

essay_word_count_normalized_test = normalizer.transform(project_data_test['essay_word_count'].valu
es.reshape(1, -1))

#reshaping again after normalization

essay_word_count_normalized_train = essay_word_count_normalized_train.reshape(-1, 1)
essay_word_count_normalized_test = essay_word_count_normalized_test.reshape(-1, 1)
```

```
print('After normalization')
print(essay_word_count_normalized_train.shape)

print(essay_word_count_normalized_test.shape)
```

```
After normalization
(33500, 1)
(16500, 1)
```

In [56]:

```
normalizer = Normalizer()
normalizer.fit(project_data_train['title_word_count'].values.reshape(-1,1))


title_word_count_normalized_train = normalizer.transform(project_data_train['title_word_count'].va
lues.reshape(1, -1))

title_word_count_normalized_test = normalizer.transform(project_data_test['title_word_count'].valu
es.reshape(1, -1))

#reshaping again after normalization

title_word_count_normalized_train = title_word_count_normalized_train.reshape(-1, 1)
title_word_count_normalized_test = title_word_count_normalized_test.reshape(-1, 1)




print('After normalization')
print(title_word_count_normalized_train.shape)

print(title_word_count_normalized_test.shape)
```

```
After normalization
(33500, 1)
(16500, 1)
```

In [57]:

```
normalizer = Normalizer()
normalizer.fit(project_data_train['neg'].values.reshape(-1,1))


sent_neg_train = normalizer.transform(project_data_train['neg'].values.reshape(1, -1))

sent_neg_test = normalizer.transform(project_data_test['neg'].values.reshape(1, -1))

#reshaping again after normalization
sent_neg_train = sent_neg_train.reshape(-1,1)
sent_neg_test = sent_neg_test.reshape(-1,1)



print('After normalization')
print(sent_neg_train.shape)

print(sent_neg_test.shape)
```

```
After normalization
(33500, 1)
(16500, 1)
```

In [58]:

```
normalizer = Normalizer()
normalizer.fit(project_data_train['pos'].values.reshape(-1,1))


sent_pos_train = normalizer.transform(project_data_train['pos'].values.reshape(1, -1))

sent_pos_test = normalizer.transform(project_data_test['pos'].values.reshape(1, -1))
```

```
sent_pos_test = normalizer.transform(project_data_test['pos'].values.reshape(1, -1))

#reshaping again after normalization
sent_pos_train = sent_pos_train.reshape(-1,1)
sent_pos_test = sent_pos_test.reshape(-1,1)



print('After normalization')

print(sent_pos_train.shape)

print(sent_pos_test.shape)
```

```
After normalization
(33500, 1)
(16500, 1)
```

In [59]:

```
normalizer = Normalizer()
normalizer.fit(project_data_train['neu'].values.reshape(-1,1))


sent_neu_train = normalizer.transform(project_data_train['neu'].values.reshape(1, -1))

sent_neu_test = normalizer.transform(project_data_test['neu'].values.reshape(1, -1))

#reshaping again after normalization
sent_neu_train = sent_neu_train.reshape(-1,1)
sent_neu_test = sent_neu_test.reshape(-1,1)


print('After normalization')
print(sent_neu_train.shape)

print(sent_neu_test.shape)
```

```
After normalization
(33500, 1)
(16500, 1)
```

In [60]:

```
normalizer = Normalizer()
normalizer.fit(project_data_train['compound'].values.reshape(-1,1))


sent_compound_train = normalizer.transform(project_data_train['compound'].values.reshape(1, -1))

sent_compound_test = normalizer.transform(project_data_test['compound'].values.reshape(1, -1))


#reshaping again after normalization
sent_compound_train = sent_compound_train.reshape(-1,1)
sent_compound_test = sent_compound_test.reshape(-1,1)


print('After normalization')
print(sent_compound_train.shape)

print(sent_compound_test.shape)
```

```
After normalization
(33500, 1)
(16500, 1)
```

Response coding for Categorical Data

I wrote my own function to calculate the response values for categorical features (works for

both train and test data)

In [61]:

```python
#https://stackoverflow.com/questions/11869910/pandas-filter-rows-of-dataframe-with-operator-chaini
ng
def mask(df, key, value):
    return df[df[key] == value]

def get_response(data,data_label):
    cat_values = np.unique(data).tolist()
    df = pd.DataFrame({'feature':data.values.tolist(),'label':data_label.values.tolist()})
    pd.DataFrame.mask = mask

    accep = {};reject={};prob_neg = {};prob_pos={}
    for i in cat_values:
        count_0 = len(df.mask('feature', i).mask('label', 0))
        count_1 = len(df.mask('feature', i).mask('label', 1))
        total   =   count_0 + count_1
        prob_0 = count_0/total
        prob_1 = count_1/total
        accep[i] = count_1
        reject[i] = count_0
        prob_neg[i] = prob_0
        prob_pos[i] = prob_1

    return prob_neg,prob_pos
```

In [62]:

```python
cat_0_train = get_response(project_data_train['clean_categories'],y_train)[0]
cat_1_train = get_response(project_data_train['clean_categories'],y_train)[1]
```

In [63]:

```python
subcat_0_train = get_response(project_data_train['clean_subcategories'],y_train)[0]
subcat_1_train = get_response(project_data_train['clean_subcategories'],y_train)[1]
```

In [64]:

```python
state_0_train = get_response(project_data_train['school_state'],y_train)[0]
state_1_train = get_response(project_data_train['school_state'],y_train)[1]
```

In [65]:

```python
prefix_0_train = get_response(project_data_train['teacher_prefix'],y_train)[0]
prefix_1_train = get_response(project_data_train['teacher_prefix'],y_train)[1]
```

In [66]:

```python
grad_cat_0_train = get_response(project_data_train['project_grade_category'],y_train)[0]
grad_cat_1_train = get_response(project_data_train['project_grade_category'],y_train)[1]
```

In [67]:

```python
cat_0_test = get_response(project_data_test['clean_categories'],y_test)[0]
cat_1_test = get_response(project_data_test['clean_categories'],y_test)[1]
```

In [68]:

```python
subcat_0_test = get_response(project_data_test['clean_subcategories'],y_test)[0]
subcat_1_test = get_response(project_data_test['clean_subcategories'],y_test)[1]
```

In [69]:

```python
state_0_test = get_response(project_data_test['school_state'],y_test)[0]
state_1_test = get_response(project_data_test['school_state'],y_test)[1]
```

In [70]:

```
prefix_0_test = get_response(project_data_test['teacher_prefix'],y_test)[0]
prefix_1_test = get_response(project_data_test['teacher_prefix'],y_test)[1]
```

In [71]:

```
grad_cat_0_test = get_response(project_data_test['project_grade_category'],y_test)[0]
grad_cat_1_test = get_response(project_data_test['project_grade_category'],y_test)[1]
```

In [72]:

```
cat_0_train
```

Out[72]:

```
{'AppliedLearning': 0.19329896907216496,
 'AppliedLearning Health_Sports': 0.14893617021276595,
 'AppliedLearning History_Civics': 0.16363636363636364,
 'AppliedLearning Literacy_Language': 0.16400580551523947,
 'AppliedLearning Math_Science': 0.1836734693877551,
 'AppliedLearning Music_Arts': 0.18181818181818182,
 'AppliedLearning SpecialNeeds': 0.2018140589569161,
 'AppliedLearning Warmth Care_Hunger': 0.16666666666666666,
 'Health_Sports': 0.148708254568368,
 'Health_Sports AppliedLearning': 0.21875,
 'Health_Sports History_Civics': 0.06666666666666667,
 'Health_Sports Literacy_Language': 0.1910569105691057,
 'Health_Sports Math_Science': 0.24285714285714285,
 'Health_Sports Music_Arts': 0.25,
 'Health_Sports SpecialNeeds': 0.13959390862944163,
 'Health_Sports Warmth Care_Hunger': 0.1,
 'History_Civics': 0.18953068592057762,
 'History_Civics AppliedLearning': 0.2,
 'History_Civics Health_Sports': 0.0,
 'History_Civics Literacy_Language': 0.087248322147651,
 'History_Civics Math_Science': 0.15789473684210525,
 'History_Civics Music_Arts': 0.15555555555555556,
 'History_Civics SpecialNeeds': 0.2054794520547945,
 'Literacy_Language': 0.13683777106780126,
 'Literacy_Language AppliedLearning': 0.17989417989417988,
 'Literacy_Language Health_Sports': 0.18181818181818182,
 'Literacy_Language History_Civics': 0.13765182186234817,
 'Literacy_Language Math_Science': 0.1334674714956405,
 'Literacy_Language Music_Arts': 0.16165413533834586,
 'Literacy_Language SpecialNeeds': 0.15057283142389524,
 'Literacy_Language Warmth Care_Hunger': 0.0,
 'Math_Science': 0.1822745703803823,
 'Math_Science AppliedLearning': 0.14945652173913043,
 'Math_Science Health_Sports': 0.2204724409448819,
 'Math_Science History_Civics': 0.15025906735751296,
 'Math_Science Literacy_Language': 0.12640449438202248,
 'Math_Science Music_Arts': 0.17834394904458598,
 'Math_Science SpecialNeeds': 0.19185059422750425,
 'Math_Science Warmth Care_Hunger': 0.3333333333333333,
 'Music_Arts': 0.13349968808484092,
 'Music_Arts AppliedLearning': 0.25,
 'Music_Arts Health_Sports': 0.16666666666666666,
 'Music_Arts History_Civics': 0.6,
 'Music_Arts SpecialNeeds': 0.05555555555555555,
 'Music_Arts Warmth Care_Hunger': 1.0,
 'SpecialNeeds': 0.186401833460657,
 'SpecialNeeds Health_Sports': 0.1111111111111111,
 'SpecialNeeds Music_Arts': 0.18947368421052632,
 'SpecialNeeds Warmth Care_Hunger': 0.0,
 'Warmth Care_Hunger': 0.06295399515738499}
```

In [73]:

```
cat_1_train
```

Out[73]:

```
{'AppliedLearning': 0.8067010309278351,
 'AppliedLearning Health_Sports': 0.851063829787234,
 'AppliedLearning History_Civics': 0.8363636363636363,
 'AppliedLearning Literacy_Language': 0.8359941944847605,
 'AppliedLearning Math_Science': 0.8163265306122449,
 'AppliedLearning Music_Arts': 0.8181818181818182,
 'AppliedLearning SpecialNeeds': 0.7981859410430839,
 'AppliedLearning Warmth Care_Hunger': 0.8333333333333334,
 'Health_Sports': 0.851291745431632,
 'Health_Sports AppliedLearning': 0.78125,
 'Health_Sports History_Civics': 0.9333333333333333,
 'Health_Sports Literacy_Language': 0.8089430894308943,
 'Health_Sports Math_Science': 0.7571428571428571,
 'Health_Sports Music_Arts': 0.75,
 'Health_Sports SpecialNeeds': 0.8604060913705583,
 'Health_Sports Warmth Care_Hunger': 0.9,
 'History_Civics': 0.8104693140794224,
 'History_Civics AppliedLearning': 0.8,
 'History_Civics Health_Sports': 1.0,
 'History_Civics Literacy_Language': 0.912751677852349,
 'History_Civics Math_Science': 0.8421052631578947,
 'History_Civics Music_Arts': 0.8444444444444444,
 'History_Civics SpecialNeeds': 0.7945205479452054,
 'Literacy_Language': 0.8631622289321987,
 'Literacy_Language AppliedLearning': 0.8201058201058201,
 'Literacy_Language Health_Sports': 0.8181818181818182,
 'Literacy_Language History_Civics': 0.8623481781376519,
 'Literacy_Language Math_Science': 0.8665325285043595,
 'Literacy_Language Music_Arts': 0.8383458646616542,
 'Literacy_Language SpecialNeeds': 0.8494271685761048,
 'Literacy_Language Warmth Care_Hunger': 1.0,
 'Math_Science': 0.8177254296196177,
 'Math_Science AppliedLearning': 0.8505434782608695,
 'Math_Science Health_Sports': 0.7795275590551181,
 'Math_Science History_Civics': 0.8497409326424871,
 'Math_Science Literacy_Language': 0.8735955056179775,
 'Math_Science Music_Arts': 0.821656050955414,
 'Math_Science SpecialNeeds': 0.8081494057724957,
 'Math_Science Warmth Care_Hunger': 0.6666666666666666,
 'Music_Arts': 0.8665003119151591,
 'Music_Arts AppliedLearning': 0.75,
 'Music_Arts Health_Sports': 0.8333333333333334,
 'Music_Arts History_Civics': 0.4,
 'Music_Arts SpecialNeeds': 0.9444444444444444,
 'Music_Arts Warmth Care_Hunger': 0.0,
 'SpecialNeeds': 0.813598166539343,
 'SpecialNeeds Health_Sports': 0.8888888888888888,
 'SpecialNeeds Music_Arts': 0.8105263157894737,
 'SpecialNeeds Warmth Care_Hunger': 1.0,
 'Warmth Care_Hunger': 0.937046004842615}
```

In [74]:

```
cat_neg_train = []
cat_pos_train = []
for i in project_data_train['clean_categories']:
    cat_neg_train.append(cat_0_train[i])
    cat_pos_train.append(cat_1_train[i])
project_data_train['cat_0'] = cat_neg_train
project_data_train['cat_1'] = cat_pos_train
```

In [75]:

```
subcat_0_train
```

Out[75]:

```
{'AppliedSciences': 0.1970310391363023,
 'AppliedSciences CharacterEducation': 0.16666666666666666,
 'AppliedSciences Civics_Government': 0.2,
 'AppliedSciences College_CareerPrep': 0.12878787878787878,
 'AppliedSciences CommunityService': 0.2857142857142857,
 'AppliedSciences ESL': 0.125,
 'AppliedSciences EarlyDevelopment': 0.19607843137254902,
 'AppliedSciences EnvironmentalScience': 0.22727272727272727,
```

'AppliedSciences Extracurricular': 0.047619047619047616,
'AppliedSciences FinancialLiteracy': 1.0,
'AppliedSciences ForeignLanguages': 0.3333333333333333,
'AppliedSciences Gym_Fitness': 0.14285714285714285,
'AppliedSciences Health_LifeScience': 0.16049382716049382,
'AppliedSciences Health_Wellness': 0.23529411764705882,
'AppliedSciences History_Geography': 0.17857142857142858,
'AppliedSciences Literacy': 0.13559322033898305,
'AppliedSciences Literature_Writing': 0.12213740458015267,
'AppliedSciences Mathematics': 0.1727447216890595,
'AppliedSciences Music': 0.13333333333333333,
'AppliedSciences NutritionEducation': 1.0,
'AppliedSciences Other': 0.15789473684210525,
'AppliedSciences ParentInvolvement': 0.1111111111111111,
'AppliedSciences PerformingArts': 0.125,
'AppliedSciences SocialSciences': 0.11764705882352941,
'AppliedSciences SpecialNeeds': 0.14035087719298245,
'AppliedSciences TeamSports': 0.0,
'AppliedSciences VisualArts': 0.16939890710382513,
'AppliedSciences Warmth Care_Hunger': 0.0,
'CharacterEducation': 0.20909090909090908,
'CharacterEducation Civics_Government': 0.0,
'CharacterEducation College_CareerPrep': 0.23333333333333334,
'CharacterEducation CommunityService': 0.12,
'CharacterEducation ESL': 0.0,
'CharacterEducation EarlyDevelopment': 0.2777777777777778,
'CharacterEducation EnvironmentalScience': 0.125,
'CharacterEducation Extracurricular': 0.2222222222222222,
'CharacterEducation FinancialLiteracy': 0.0,
'CharacterEducation ForeignLanguages': 0.25,
'CharacterEducation Gym_Fitness': 0.3333333333333333,
'CharacterEducation Health_LifeScience': 0.0,
'CharacterEducation Health_Wellness': 0.14705882352941177,
'CharacterEducation History_Geography': 0.0,
'CharacterEducation Literacy': 0.12871287128712872,
'CharacterEducation Literature_Writing': 0.17307692307692307,
'CharacterEducation Mathematics': 0.18181818181818182,
'CharacterEducation Music': 0.25,
'CharacterEducation Other': 0.1891891891891892,
'CharacterEducation ParentInvolvement': 0.2857142857142857,
'CharacterEducation PerformingArts': 0.25,
'CharacterEducation SocialSciences': 0.2,
'CharacterEducation SpecialNeeds': 0.15384615384615385,
'CharacterEducation TeamSports': 0.3333333333333333,
'CharacterEducation VisualArts': 0.08695652173913043,
'CharacterEducation Warmth Care_Hunger': 1.0,
'Civics_Government': 0.18181818181818182,
'Civics_Government College_CareerPrep': 0.5,
'Civics_Government CommunityService': 0.2,
'Civics_Government ESL': 0.0,
'Civics_Government Economics': 0.4,
'Civics_Government EnvironmentalScience': 0.25,
'Civics_Government FinancialLiteracy': 0.0,
'Civics_Government Health_LifeScience': 0.14285714285714285,
'Civics_Government Health_Wellness': 0.0,
'Civics_Government History_Geography': 0.1791044776119403,
'Civics_Government Literacy': 0.06,
'Civics_Government Literature_Writing': 0.0967741935483871,
'Civics_Government Mathematics': 0.5,
'Civics_Government NutritionEducation': 0.0,
'Civics_Government PerformingArts': 1.0,
'Civics_Government SocialSciences': 0.12,
'Civics_Government SpecialNeeds': 0.25,
'Civics_Government TeamSports': 0.0,
'Civics_Government VisualArts': 0.16666666666666666,
'College_CareerPrep': 0.21428571428571427,
'College_CareerPrep CommunityService': 0.0,
'College_CareerPrep ESL': 0.0,
'College_CareerPrep EarlyDevelopment': 0.0,
'College_CareerPrep Economics': 0.0,
'College_CareerPrep EnvironmentalScience': 0.0,
'College_CareerPrep Extracurricular': 0.07692307692307693,
'College_CareerPrep FinancialLiteracy': 0.5,
'College_CareerPrep ForeignLanguages': 0.4,
'College_CareerPrep Gym_Fitness': 1.0,
'College_CareerPrep Health_LifeScience': 0.08333333333333333,
'College_CareerPrep Health_Wellness': 0.3333333333333333,

'College_CareerPrep History_Geography': 0.2,
'College_CareerPrep Literacy': 0.09333333333333334,
'College_CareerPrep Literature_Writing': 0.17,
'College_CareerPrep Mathematics': 0.14473684210526316,
'College_CareerPrep Music': 0.6666666666666666,
'College_CareerPrep NutritionEducation': 0.5,
'College_CareerPrep Other': 0.14814814814814814,
'College_CareerPrep ParentInvolvement': 0.23076923076923078,
'College_CareerPrep PerformingArts': 0.125,
'College_CareerPrep SocialSciences': 0.2222222222222222,
'College_CareerPrep SpecialNeeds': 0.2926829268292683,
'College_CareerPrep TeamSports': 0.0,
'College_CareerPrep VisualArts': 0.17307692307692307,
'College_CareerPrep Warmth Care_Hunger': 0.0,
'CommunityService': 0.25,
'CommunityService ESL': 0.5,
'CommunityService EarlyDevelopment': 0.0,
'CommunityService Economics': 0.0,
'CommunityService EnvironmentalScience': 0.16666666666666666,
'CommunityService Extracurricular': 0.2857142857142857,
'CommunityService FinancialLiteracy': 0.0,
'CommunityService Gym_Fitness': 0.0,
'CommunityService Health_LifeScience': 0.0,
'CommunityService Health_Wellness': 0.5,
'CommunityService History_Geography': 0.0,
'CommunityService Literacy': 0.0,
'CommunityService Literature_Writing': 0.25,
'CommunityService Mathematics': 0.2,
'CommunityService Music': 0.0,
'CommunityService Other': 0.0,
'CommunityService ParentInvolvement': 0.0,
'CommunityService PerformingArts': 0.0,
'CommunityService SocialSciences': 1.0,
'CommunityService SpecialNeeds': 0.375,
'CommunityService VisualArts': 0.35714285714285715,
'ESL': 0.15267175572519084,
'ESL EarlyDevelopment': 0.10526315789473684,
'ESL EnvironmentalScience': 0.3,
'ESL Extracurricular': 0.0,
'ESL ForeignLanguages': 0.14285714285714285,
'ESL Gym_Fitness': 0.5,
'ESL Health_LifeScience': 0.38461538461538464,
'ESL Health_Wellness': 0.0,
'ESL History_Geography': 0.16666666666666666,
'ESL Literacy': 0.140625,
'ESL Literature_Writing': 0.16736401673640167,
'ESL Mathematics': 0.1686746987951807,
'ESL Music': 0.0,
'ESL Other': 0.3333333333333333,
'ESL ParentInvolvement': 0.5,
'ESL PerformingArts': 0.0,
'ESL SocialSciences': 0.6,
'ESL SpecialNeeds': 0.18181818181818182,
'ESL VisualArts': 0.23529411764705882,
'EarlyDevelopment': 0.17518248175182483,
'EarlyDevelopment EnvironmentalScience': 0.3076923076923077,
'EarlyDevelopment Extracurricular': 0.4,
'EarlyDevelopment ForeignLanguages': 0.0,
'EarlyDevelopment Gym_Fitness': 0.1,
'EarlyDevelopment Health_LifeScience': 0.2727272727272727,
'EarlyDevelopment Health_Wellness': 0.09473684210526316,
'EarlyDevelopment History_Geography': 0.0,
'EarlyDevelopment Literacy': 0.16877637130801687,
'EarlyDevelopment Literature_Writing': 0.2,
'EarlyDevelopment Mathematics': 0.22826086956521738,
'EarlyDevelopment Music': 0.14285714285714285,
'EarlyDevelopment NutritionEducation': 0.0,
'EarlyDevelopment Other': 0.12,
'EarlyDevelopment ParentInvolvement': 0.15384615384615385,
'EarlyDevelopment PerformingArts': 0.0,
'EarlyDevelopment SocialSciences': 0.0,
'EarlyDevelopment SpecialNeeds': 0.1762114537444934,
'EarlyDevelopment TeamSports': 0.3333333333333333,
'EarlyDevelopment VisualArts': 0.2127659574468085,
'EarlyDevelopment Warmth Care_Hunger': 0.0,
'Economics': 0.16666666666666666,
'Economics EnvironmentalScience': 0.0,

```
'Economics EnvironmentalScience': 0.0,
'Economics FinancialLiteracy': 0.19047619047619047,
'Economics History_Geography': 0.0,
'Economics Literacy': 0.0,
'Economics Literature_Writing': 0.0,
'Economics Mathematics': 0.0,
'Economics Music': 0.0,
'Economics NutritionEducation': 0.0,
'Economics Other': 0.0,
'Economics SocialSciences': 0.0,
'EnvironmentalScience': 0.17415730337078653,
'EnvironmentalScience Extracurricular': 0.3333333333333333,
'EnvironmentalScience FinancialLiteracy': 0.0,
'EnvironmentalScience ForeignLanguages': 0.0,
'EnvironmentalScience Gym_Fitness': 0.5,
'EnvironmentalScience Health_LifeScience': 0.1864406779661017,
'EnvironmentalScience Health_Wellness': 0.17647058823529413,
'EnvironmentalScience History_Geography': 0.02127659574468085,
'EnvironmentalScience Literacy': 0.14173228346456693,
'EnvironmentalScience Literature_Writing': 0.1411764705882353,
'EnvironmentalScience Mathematics': 0.16600790513833993,
'EnvironmentalScience Music': 0.0,
'EnvironmentalScience NutritionEducation': 0.4444444444444444,
'EnvironmentalScience Other': 0.25,
'EnvironmentalScience ParentInvolvement': 0.0,
'EnvironmentalScience PerformingArts': 0.3333333333333333,
'EnvironmentalScience SocialSciences': 0.19047619047619047,
'EnvironmentalScience SpecialNeeds': 0.16666666666666666,
'EnvironmentalScience VisualArts': 0.19696969696969696,
'EnvironmentalScience Warmth Care_Hunger': 0.5,
'Extracurricular': 0.25,
'Extracurricular ForeignLanguages': 0.0,
'Extracurricular Gym_Fitness': 0.0,
'Extracurricular Health_LifeScience': 0.5,
'Extracurricular Health_Wellness': 0.25,
'Extracurricular History_Geography': 0.0,
'Extracurricular Literacy': 0.3333333333333333,
'Extracurricular Literature_Writing': 0.25,
'Extracurricular Mathematics': 0.17647058823529413,
'Extracurricular Music': 0.14285714285714285,
'Extracurricular NutritionEducation': 1.0,
'Extracurricular Other': 0.25,
'Extracurricular ParentInvolvement': 0.25,
'Extracurricular PerformingArts': 0.0,
'Extracurricular SocialSciences': 0.0,
'Extracurricular SpecialNeeds': 0.2857142857142857,
'Extracurricular TeamSports': 0.0,
'Extracurricular VisualArts': 0.20833333333333334,
'FinancialLiteracy': 0.1896551724137931,
'FinancialLiteracy ForeignLanguages': 0.0,
'FinancialLiteracy Health_LifeScience': 0.0,
'FinancialLiteracy Health_Wellness': 0.0,
'FinancialLiteracy History_Geography': 0.0,
'FinancialLiteracy Literacy': 0.14285714285714285,
'FinancialLiteracy Literature_Writing': 1.0,
'FinancialLiteracy Mathematics': 0.13043478260869565,
'FinancialLiteracy Other': 0.0,
'FinancialLiteracy ParentInvolvement': 0.0,
'FinancialLiteracy SocialSciences': 0.0,
'FinancialLiteracy SpecialNeeds': 0.1875,
'FinancialLiteracy VisualArts': 0.0,
'ForeignLanguages': 0.2268041237113402,
'ForeignLanguages Gym_Fitness': 0.0,
'ForeignLanguages Health_LifeScience': 0.0,
'ForeignLanguages Health_Wellness': 0.25,
'ForeignLanguages History_Geography': 0.0,
'ForeignLanguages Literacy': 0.18032786885245902,
'ForeignLanguages Literature_Writing': 0.21739130434782608,
'ForeignLanguages Mathematics': 0.0,
'ForeignLanguages Music': 0.0,
'ForeignLanguages Other': 0.0,
'ForeignLanguages PerformingArts': 0.0,
'ForeignLanguages SocialSciences': 0.5,
'ForeignLanguages SpecialNeeds': 0.0,
'ForeignLanguages VisualArts': 0.16666666666666666,
'Gym_Fitness': 0.18351063829787234,
'Gym_Fitness Health_LifeScience': 0.0,
'Gym_Fitness Health_Wellness': 0.10632183908045977,
```

'Gym_Fitness Health_Wellness': 0.16632103960045977,
'Gym_Fitness History_Geography': 0.0,
'Gym_Fitness Literacy': 0.3333333333333333,
'Gym_Fitness Literature_Writing': 0.3333333333333333,
'Gym_Fitness Mathematics': 0.16666666666666666,
'Gym_Fitness Music': 0.3333333333333333,
'Gym_Fitness NutritionEducation': 0.19047619047619047,
'Gym_Fitness Other': 0.3333333333333333,
'Gym_Fitness PerformingArts': 0.16666666666666666,
'Gym_Fitness SocialSciences': 0.0,
'Gym_Fitness SpecialNeeds': 0.1111111111111111,
'Gym_Fitness TeamSports': 0.20320855614973263,
'Gym_Fitness VisualArts': 0.0,
'Health_LifeScience': 0.15677966101694915,
'Health_LifeScience Health_Wellness': 0.1276595744680851,
'Health_LifeScience History_Geography': 0.1875,
'Health_LifeScience Literacy': 0.09183673469387756,
'Health_LifeScience Literature_Writing': 0.109375,
'Health_LifeScience Mathematics': 0.19254658385093168,
'Health_LifeScience Music': 0.0,
'Health_LifeScience NutritionEducation': 0.42857142857142855,
'Health_LifeScience Other': 0.3333333333333333,
'Health_LifeScience ParentInvolvement': 0.0,
'Health_LifeScience SocialSciences': 0.25,
'Health_LifeScience SpecialNeeds': 0.22448979591836735,
'Health_LifeScience TeamSports': 0.0,
'Health_LifeScience VisualArts': 0.15384615384615385,
'Health_LifeScience Warmth Care_Hunger': 0.0,
'Health_Wellness': 0.13487133984028393,
'Health_Wellness History_Geography': 0.0,
'Health_Wellness Literacy': 0.19745222929936307,
'Health_Wellness Literature_Writing': 0.1625,
'Health_Wellness Mathematics': 0.2727272727272727,
'Health_Wellness Music': 0.1,
'Health_Wellness NutritionEducation': 0.1646090534979424,
'Health_Wellness Other': 0.19642857142857142,
'Health_Wellness ParentInvolvement': 1.0,
'Health_Wellness PerformingArts': 0.16666666666666666,
'Health_Wellness SocialSciences': 0.125,
'Health_Wellness SpecialNeeds': 0.13855421686746988,
'Health_Wellness TeamSports': 0.1523809523809524,
'Health_Wellness VisualArts': 0.38461538461538464,
'Health_Wellness Warmth Care_Hunger': 0.1,
'History_Geography': 0.25925925925925924,
'History_Geography Literacy': 0.04878048780487805,
'History_Geography Literature_Writing': 0.12365591397849462,
'History_Geography Mathematics': 0.20512820512820512,
'History_Geography Music': 0.0,
'History_Geography Other': 0.3333333333333333,
'History_Geography ParentInvolvement': 0.0,
'History_Geography PerformingArts': 0.14285714285714285,
'History_Geography SocialSciences': 0.13402061855670103,
'History_Geography SpecialNeeds': 0.2571428571428571,
'History_Geography VisualArts': 0.20408163265306123,
'Literacy': 0.11084745762711865,
'Literacy Literature_Writing': 0.14745465184318315,
'Literacy Mathematics': 0.12807295796986518,
'Literacy Music': 0.06976744186046512,
'Literacy NutritionEducation': 0.5,
'Literacy Other': 0.1864406779661017,
'Literacy ParentInvolvement': 0.19148936170212766,
'Literacy PerformingArts': 0.15789473684210525,
'Literacy SocialSciences': 0.14634146341463414,
'Literacy SpecialNeeds': 0.12450851900393185,
'Literacy TeamSports': 0.0,
'Literacy VisualArts': 0.15822784810126583,
'Literacy Warmth Care_Hunger': 0.0,
'Literature_Writing': 0.16117216117216118,
'Literature_Writing Mathematics': 0.13747954173486088,
'Literature_Writing Music': 0.1111111111111111,
'Literature_Writing Other': 0.21052631578947367,
'Literature_Writing ParentInvolvement': 0.06666666666666667,
'Literature_Writing PerformingArts': 0.14634146341463414,
'Literature_Writing SocialSciences': 0.09900990099009901,
'Literature_Writing SpecialNeeds': 0.1975,
'Literature_Writing TeamSports': 0.2,
'Literature_Writing VisualArts': 0.19306930693069307,
'Literature_Writing Warmth Care_Hunger': 0.0,

```
 'Literature_Writing Warmth Care_Hunger': 0.0,
 'Mathematics': 0.18153846153846154,
 'Mathematics Music': 0.1111111111111111,
 'Mathematics NutritionEducation': 0.0,
 'Mathematics Other': 0.21428571428571427,
 'Mathematics ParentInvolvement': 0.19047619047619047,
 'Mathematics PerformingArts': 0.0,
 'Mathematics SocialSciences': 0.18181818181818182,
 'Mathematics SpecialNeeds': 0.20765027322404372,
 'Mathematics TeamSports': 0.5,
 'Mathematics VisualArts': 0.2112676056338028,
 'Mathematics Warmth Care_Hunger': 0.5,
 'Music': 0.11597374179431072,
 'Music Other': 0.5,
 'Music ParentInvolvement': 0.0,
 'Music PerformingArts': 0.09027777777777778,
 'Music SocialSciences': 0.3333333333333333,
 'Music SpecialNeeds': 0.03333333333333333,
 'Music TeamSports': 0.0,
 'Music VisualArts': 0.16666666666666666,
 'NutritionEducation': 0.21839080459770116,
 'NutritionEducation Other': 0.25,
 'NutritionEducation SpecialNeeds': 0.375,
 'NutritionEducation TeamSports': 0.0,
 'NutritionEducation VisualArts': 0.5,
 'Other': 0.18461538461538463,
 'Other ParentInvolvement': 0.2,
 'Other PerformingArts': 1.0,
 'Other SocialSciences': 0.0,
 'Other SpecialNeeds': 0.21782178217821782,
 'Other TeamSports': 0.0,
 'Other VisualArts': 0.1875,
 'Other Warmth Care_Hunger': 0.0,
 'ParentInvolvement': 0.45454545454545453,
 'ParentInvolvement PerformingArts': 0.0,
 'ParentInvolvement SocialSciences': 0.2,
 'ParentInvolvement SpecialNeeds': 0.4,
 'ParentInvolvement TeamSports': 0.0,
 'ParentInvolvement VisualArts': 0.08333333333333333,
 'ParentInvolvement Warmth Care_Hunger': 0.0,
 'PerformingArts': 0.10294117647058823,
 'PerformingArts SocialSciences': 1.0,
 'PerformingArts SpecialNeeds': 0.16666666666666666,
 'PerformingArts TeamSports': 0.3333333333333333,
 'PerformingArts VisualArts': 0.2413793103448276,
 'SocialSciences': 0.18181818181818182,
 'SocialSciences SpecialNeeds': 0.07142857142857142,
 'SocialSciences VisualArts': 0.05555555555555555,
 'SpecialNeeds': 0.186401833460657,
 'SpecialNeeds TeamSports': 0.1111111111111111,
 'SpecialNeeds VisualArts': 0.18947368421052632,
 'SpecialNeeds Warmth Care_Hunger': 0.0,
 'TeamSports': 0.18237082066869,
 'TeamSports VisualArts': 0.5,
 'VisualArts': 0.16444444444444445,
 'VisualArts Warmth Care_Hunger': 1.0,
 'Warmth Care_Hunger': 0.06295399515738499}
```

In [76]:

```
subcat_1_train
```

Out[76]:

```
{'AppliedSciences': 0.8029689608636977,
 'AppliedSciences CharacterEducation': 0.8333333333333334,
 'AppliedSciences Civics_Government': 0.8,
 'AppliedSciences College_CareerPrep': 0.8712121212121212,
 'AppliedSciences CommunityService': 0.7142857142857143,
 'AppliedSciences ESL': 0.875,
 'AppliedSciences EarlyDevelopment': 0.803921568627451,
 'AppliedSciences EnvironmentalScience': 0.7727272727272727,
 'AppliedSciences Extracurricular': 0.9523809523809523,
 'AppliedSciences FinancialLiteracy': 0.0,
 'AppliedSciences ForeignLanguages': 0.6666666666666666,
 'AppliedSciences Gym_Fitness': 0.8571428571428571,
```

```
'AppliedSciences Health_LifeScience': 0.8395061728395061,
'AppliedSciences Health_Wellness': 0.7647058823529411,
'AppliedSciences History_Geography': 0.8214285714285714,
'AppliedSciences Literacy': 0.864406779661017,
'AppliedSciences Literature_Writing': 0.8778625954198473,
'AppliedSciences Mathematics': 0.8272552783109405,
'AppliedSciences Music': 0.8666666666666667,
'AppliedSciences NutritionEducation': 0.0,
'AppliedSciences Other': 0.8421052631578947,
'AppliedSciences ParentInvolvement': 0.8888888888888888,
'AppliedSciences PerformingArts': 0.875,
'AppliedSciences SocialSciences': 0.8823529411764706,
'AppliedSciences SpecialNeeds': 0.8596491228070176,
'AppliedSciences TeamSports': 1.0,
'AppliedSciences VisualArts': 0.8306010928961749,
'AppliedSciences Warmth Care_Hunger': 1.0,
'CharacterEducation': 0.7909090909090909,
'CharacterEducation Civics_Government': 1.0,
'CharacterEducation College_CareerPrep': 0.7666666666666667,
'CharacterEducation CommunityService': 0.88,
'CharacterEducation ESL': 1.0,
'CharacterEducation EarlyDevelopment': 0.7222222222222222,
'CharacterEducation EnvironmentalScience': 0.875,
'CharacterEducation Extracurricular': 0.7777777777777778,
'CharacterEducation FinancialLiteracy': 1.0,
'CharacterEducation ForeignLanguages': 0.75,
'CharacterEducation Gym_Fitness': 0.6666666666666666,
'CharacterEducation Health_LifeScience': 1.0,
'CharacterEducation Health_Wellness': 0.8529411764705882,
'CharacterEducation History_Geography': 1.0,
'CharacterEducation Literacy': 0.8712871287128713,
'CharacterEducation Literature_Writing': 0.8269230769230769,
'CharacterEducation Mathematics': 0.8181818181818182,
'CharacterEducation Music': 0.75,
'CharacterEducation Other': 0.8108108108108109,
'CharacterEducation ParentInvolvement': 0.7142857142857143,
'CharacterEducation PerformingArts': 0.75,
'CharacterEducation SocialSciences': 0.8,
'CharacterEducation SpecialNeeds': 0.8461538461538461,
'CharacterEducation TeamSports': 0.6666666666666666,
'CharacterEducation VisualArts': 0.9130434782608695,
'CharacterEducation Warmth Care_Hunger': 0.0,
'Civics_Government': 0.8181818181818182,
'Civics_Government College_CareerPrep': 0.5,
'Civics_Government CommunityService': 0.8,
'Civics_Government ESL': 1.0,
'Civics_Government Economics': 0.6,
'Civics_Government EnvironmentalScience': 0.75,
'Civics_Government FinancialLiteracy': 1.0,
'Civics_Government Health_LifeScience': 0.8571428571428571,
'Civics_Government Health_Wellness': 1.0,
'Civics_Government History_Geography': 0.8208955223880597,
'Civics_Government Literacy': 0.94,
'Civics_Government Literature_Writing': 0.9032258064516129,
'Civics_Government Mathematics': 0.5,
'Civics_Government NutritionEducation': 1.0,
'Civics_Government PerformingArts': 0.0,
'Civics_Government SocialSciences': 0.88,
'Civics_Government SpecialNeeds': 0.75,
'Civics_Government TeamSports': 1.0,
'Civics_Government VisualArts': 0.8333333333333334,
'College_CareerPrep': 0.7857142857142857,
'College_CareerPrep CommunityService': 1.0,
'College_CareerPrep ESL': 1.0,
'College_CareerPrep EarlyDevelopment': 1.0,
'College_CareerPrep Economics': 1.0,
'College_CareerPrep EnvironmentalScience': 1.0,
'College_CareerPrep Extracurricular': 0.9230769230769231,
'College_CareerPrep FinancialLiteracy': 0.5,
'College_CareerPrep ForeignLanguages': 0.6,
'College_CareerPrep Gym_Fitness': 0.0,
'College_CareerPrep Health_LifeScience': 0.9166666666666666,
'College_CareerPrep Health_Wellness': 0.6666666666666666,
'College_CareerPrep History_Geography': 0.8,
'College_CareerPrep Literacy': 0.9066666666666666,
'College_CareerPrep Literature_Writing': 0.83,
'College_CareerPrep Mathematics': 0.8552631578947368,
```

```
'College_CareerPrep Music': 0.3333333333333333,
'College_CareerPrep NutritionEducation': 0.5,
'College_CareerPrep Other': 0.8518518518518519,
'College_CareerPrep ParentInvolvement': 0.7692307692307693,
'College_CareerPrep PerformingArts': 0.875,
'College_CareerPrep SocialSciences': 0.7777777777777778,
'College_CareerPrep SpecialNeeds': 0.7073170731707317,
'College_CareerPrep TeamSports': 1.0,
'College_CareerPrep VisualArts': 0.8269230769230769,
'College_CareerPrep Warmth Care_Hunger': 1.0,
'CommunityService': 0.75,
'CommunityService ESL': 0.5,
'CommunityService EarlyDevelopment': 1.0,
'CommunityService Economics': 1.0,
'CommunityService EnvironmentalScience': 0.8333333333333334,
'CommunityService Extracurricular': 0.7142857142857143,
'CommunityService FinancialLiteracy': 1.0,
'CommunityService Gym_Fitness': 1.0,
'CommunityService Health_LifeScience': 1.0,
'CommunityService Health_Wellness': 0.5,
'CommunityService History_Geography': 1.0,
'CommunityService Literacy': 1.0,
'CommunityService Literature_Writing': 0.75,
'CommunityService Mathematics': 0.8,
'CommunityService Music': 1.0,
'CommunityService Other': 1.0,
'CommunityService ParentInvolvement': 1.0,
'CommunityService PerformingArts': 1.0,
'CommunityService SocialSciences': 0.0,
'CommunityService SpecialNeeds': 0.625,
'CommunityService VisualArts': 0.6428571428571429,
'ESL': 0.8473282442748091,
'ESL EarlyDevelopment': 0.8947368421052632,
'ESL EnvironmentalScience': 0.7,
'ESL Extracurricular': 1.0,
'ESL ForeignLanguages': 0.8571428571428571,
'ESL Gym_Fitness': 0.5,
'ESL Health_LifeScience': 0.6153846153846154,
'ESL Health_Wellness': 1.0,
'ESL History_Geography': 0.8333333333333334,
'ESL Literacy': 0.859375,
'ESL Literature_Writing': 0.8326359832635983,
'ESL Mathematics': 0.8313253012048193,
'ESL Music': 1.0,
'ESL Other': 0.6666666666666666,
'ESL ParentInvolvement': 0.5,
'ESL PerformingArts': 1.0,
'ESL SocialSciences': 0.4,
'ESL SpecialNeeds': 0.8181818181818182,
'ESL VisualArts': 0.7647058823529411,
'EarlyDevelopment': 0.8248175182481752,
'EarlyDevelopment EnvironmentalScience': 0.6923076923076923,
'EarlyDevelopment Extracurricular': 0.6,
'EarlyDevelopment ForeignLanguages': 1.0,
'EarlyDevelopment Gym_Fitness': 0.9,
'EarlyDevelopment Health_LifeScience': 0.7272727272727273,
'EarlyDevelopment Health_Wellness': 0.9052631578947369,
'EarlyDevelopment History_Geography': 1.0,
'EarlyDevelopment Literacy': 0.8312236286919831,
'EarlyDevelopment Literature_Writing': 0.8,
'EarlyDevelopment Mathematics': 0.7717391304347826,
'EarlyDevelopment Music': 0.8571428571428571,
'EarlyDevelopment NutritionEducation': 1.0,
'EarlyDevelopment Other': 0.88,
'EarlyDevelopment ParentInvolvement': 0.8461538461538461,
'EarlyDevelopment PerformingArts': 1.0,
'EarlyDevelopment SocialSciences': 1.0,
'EarlyDevelopment SpecialNeeds': 0.8237885462555066,
'EarlyDevelopment TeamSports': 0.6666666666666666,
'EarlyDevelopment VisualArts': 0.7872340425531915,
'EarlyDevelopment Warmth Care_Hunger': 1.0,
'Economics': 0.8333333333333334,
'Economics EnvironmentalScience': 1.0,
'Economics FinancialLiteracy': 0.8095238095238095,
'Economics History_Geography': 1.0,
'Economics Literacy': 1.0,
'Economics Literature_Writing': 1.0,
```

```
'Economics Mathematics': 1.0,
'Economics Music': 1.0,
'Economics NutritionEducation': 1.0,
'Economics Other': 1.0,
'Economics SocialSciences': 1.0,
'EnvironmentalScience': 0.8258426966292135,
'EnvironmentalScience Extracurricular': 0.6666666666666666,
'EnvironmentalScience FinancialLiteracy': 1.0,
'EnvironmentalScience ForeignLanguages': 1.0,
'EnvironmentalScience Gym_Fitness': 0.5,
'EnvironmentalScience Health_LifeScience': 0.8135593220338984,
'EnvironmentalScience Health_Wellness': 0.8235294117647058,
'EnvironmentalScience History_Geography': 0.9787234042553191,
'EnvironmentalScience Literacy': 0.8582677165354331,
'EnvironmentalScience Literature_Writing': 0.8588235294117647,
'EnvironmentalScience Mathematics': 0.83399209486166,
'EnvironmentalScience Music': 1.0,
'EnvironmentalScience NutritionEducation': 0.5555555555555556,
'EnvironmentalScience Other': 0.75,
'EnvironmentalScience ParentInvolvement': 1.0,
'EnvironmentalScience PerformingArts': 0.6666666666666666,
'EnvironmentalScience SocialSciences': 0.8095238095238095,
'EnvironmentalScience SpecialNeeds': 0.8333333333333334,
'EnvironmentalScience VisualArts': 0.803030303030303,
'EnvironmentalScience Warmth Care_Hunger': 0.5,
'Extracurricular': 0.75,
'Extracurricular ForeignLanguages': 1.0,
'Extracurricular Gym_Fitness': 1.0,
'Extracurricular Health_LifeScience': 0.5,
'Extracurricular Health_Wellness': 0.75,
'Extracurricular History_Geography': 1.0,
'Extracurricular Literacy': 0.6666666666666666,
'Extracurricular Literature_Writing': 0.75,
'Extracurricular Mathematics': 0.8235294117647058,
'Extracurricular Music': 0.8571428571428571,
'Extracurricular NutritionEducation': 0.0,
'Extracurricular Other': 0.75,
'Extracurricular ParentInvolvement': 0.75,
'Extracurricular PerformingArts': 1.0,
'Extracurricular SocialSciences': 1.0,
'Extracurricular SpecialNeeds': 0.7142857142857143,
'Extracurricular TeamSports': 1.0,
'Extracurricular VisualArts': 0.7916666666666666,
'FinancialLiteracy': 0.8103448275862069,
'FinancialLiteracy ForeignLanguages': 1.0,
'FinancialLiteracy Health_LifeScience': 1.0,
'FinancialLiteracy Health_Wellness': 1.0,
'FinancialLiteracy History_Geography': 1.0,
'FinancialLiteracy Literacy': 0.8571428571428571,
'FinancialLiteracy Literature_Writing': 0.0,
'FinancialLiteracy Mathematics': 0.8695652173913043,
'FinancialLiteracy Other': 1.0,
'FinancialLiteracy ParentInvolvement': 1.0,
'FinancialLiteracy SocialSciences': 1.0,
'FinancialLiteracy SpecialNeeds': 0.8125,
'FinancialLiteracy VisualArts': 1.0,
'ForeignLanguages': 0.7731958762886598,
'ForeignLanguages Gym_Fitness': 1.0,
'ForeignLanguages Health_LifeScience': 1.0,
'ForeignLanguages Health_Wellness': 0.75,
'ForeignLanguages History_Geography': 1.0,
'ForeignLanguages Literacy': 0.819672131147541,
'ForeignLanguages Literature_Writing': 0.782608695652174,
'ForeignLanguages Mathematics': 1.0,
'ForeignLanguages Music': 1.0,
'ForeignLanguages Other': 1.0,
'ForeignLanguages PerformingArts': 1.0,
'ForeignLanguages SocialSciences': 0.5,
'ForeignLanguages SpecialNeeds': 1.0,
'ForeignLanguages VisualArts': 0.8333333333333334,
'Gym_Fitness': 0.8164893617021277,
'Gym_Fitness Health_LifeScience': 1.0,
'Gym_Fitness Health_Wellness': 0.8936781609195402,
'Gym_Fitness History_Geography': 1.0,
'Gym_Fitness Literacy': 0.6666666666666666,
'Gym_Fitness Literature_Writing': 0.6666666666666666,
'Gym_Fitness Mathematics': 0.8333333333333334,
```

```
 'Gym_Fitness Music': 0.6666666666666666,
 'Gym_Fitness NutritionEducation': 0.8095238095238095,
 'Gym_Fitness Other': 0.6666666666666666,
 'Gym_Fitness PerformingArts': 0.8333333333333334,
 'Gym_Fitness SocialSciences': 1.0,
 'Gym_Fitness SpecialNeeds': 0.8888888888888888,
 'Gym_Fitness TeamSports': 0.7967914438502673,
 'Gym_Fitness VisualArts': 1.0,
 'Health_LifeScience': 0.8432203389830508,
 'Health_LifeScience Health_Wellness': 0.8723404255319149,
 'Health_LifeScience History_Geography': 0.8125,
 'Health_LifeScience Literacy': 0.9081632653061225,
 'Health_LifeScience Literature_Writing': 0.890625,
 'Health_LifeScience Mathematics': 0.8074534161490683,
 'Health_LifeScience Music': 1.0,
 'Health_LifeScience NutritionEducation': 0.5714285714285714,
 'Health_LifeScience Other': 0.6666666666666666,
 'Health_LifeScience ParentInvolvement': 1.0,
 'Health_LifeScience SocialSciences': 0.75,
 'Health_LifeScience SpecialNeeds': 0.7755102040816326,
 'Health_LifeScience TeamSports': 1.0,
 'Health_LifeScience VisualArts': 0.8461538461538461,
 'Health_LifeScience Warmth Care_Hunger': 1.0,
 'Health_Wellness': 0.865128660159716,
 'Health_Wellness History_Geography': 1.0,
 'Health_Wellness Literacy': 0.802547770700637,
 'Health_Wellness Literature_Writing': 0.8375,
 'Health_Wellness Mathematics': 0.7272727272727273,
 'Health_Wellness Music': 0.9,
 'Health_Wellness NutritionEducation': 0.8353909465020576,
 'Health_Wellness Other': 0.8035714285714286,
 'Health_Wellness ParentInvolvement': 0.0,
 'Health_Wellness PerformingArts': 0.8333333333333334,
 'Health_Wellness SocialSciences': 0.875,
 'Health_Wellness SpecialNeeds': 0.8614457831325302,
 'Health_Wellness TeamSports': 0.8476190476190476,
 'Health_Wellness VisualArts': 0.6153846153846154,
 'Health_Wellness Warmth Care_Hunger': 0.9,
 'History_Geography': 0.7407407407407407,
 'History_Geography Literacy': 0.9512195121951219,
 'History_Geography Literature_Writing': 0.8763440860215054,
 'History_Geography Mathematics': 0.7948717948717948,
 'History_Geography Music': 1.0,
 'History_Geography Other': 0.6666666666666666,
 'History_Geography ParentInvolvement': 1.0,
 'History_Geography PerformingArts': 0.8571428571428571,
 'History_Geography SocialSciences': 0.865979381443299,
 'History_Geography SpecialNeeds': 0.7428571428571429,
 'History_Geography VisualArts': 0.7959183673469388,
 'Literacy': 0.8891525423728813,
 'Literacy Literature_Writing': 0.8525453481568168,
 'Literacy Mathematics': 0.8719270420301348,
 'Literacy Music': 0.9302325581395349,
 'Literacy NutritionEducation': 0.5,
 'Literacy Other': 0.8135593220338984,
 'Literacy ParentInvolvement': 0.8085106382978723,
 'Literacy PerformingArts': 0.8421052631578947,
 'Literacy SocialSciences': 0.8536585365853658,
 'Literacy SpecialNeeds': 0.8754914809960681,
 'Literacy TeamSports': 1.0,
 'Literacy VisualArts': 0.8417721518987342,
 'Literacy Warmth Care_Hunger': 1.0,
 'Literature_Writing': 0.8388278388278388,
 'Literature_Writing Mathematics': 0.8625204582651391,
 'Literature_Writing Music': 0.8888888888888888,
 'Literature_Writing Other': 0.7894736842105263,
 'Literature_Writing ParentInvolvement': 0.9333333333333333,
 'Literature_Writing PerformingArts': 0.8536585365853658,
 'Literature_Writing SocialSciences': 0.900990099009901,
 'Literature_Writing SpecialNeeds': 0.8025,
 'Literature_Writing TeamSports': 0.8,
 'Literature_Writing VisualArts': 0.806930693069307,
 'Literature_Writing Warmth Care_Hunger': 1.0,
 'Mathematics': 0.8184615384615385,
 'Mathematics Music': 0.8888888888888888,
 'Mathematics NutritionEducation': 1.0,
 'Mathematics Other': 0.7857142857142857,
```

```
 'Mathematics ParentInvolvement': 0.8095238095238095,
 'Mathematics PerformingArts': 1.0,
 'Mathematics SocialSciences': 0.8181818181818182,
 'Mathematics SpecialNeeds': 0.7923497267759563,
 'Mathematics TeamSports': 0.5,
 'Mathematics VisualArts': 0.7887323943661971,
 'Mathematics Warmth Care_Hunger': 0.5,
 'Music': 0.884026258205689З,
 'Music Other': 0.5,
 'Music ParentInvolvement': 1.0,
 'Music PerformingArts': 0.9097222222222222,
 'Music SocialSciences': 0.6666666666666666,
 'Music SpecialNeeds': 0.9666666666666667,
 'Music TeamSports': 1.0,
 'Music VisualArts': 0.8333333333333334,
 'NutritionEducation': 0.7816091954022989,
 'NutritionEducation Other': 0.75,
 'NutritionEducation SpecialNeeds': 0.625,
 'NutritionEducation TeamSports': 1.0,
 'NutritionEducation VisualArts': 0.5,
 'Other': 0.8153846153846154,
 'Other ParentInvolvement': 0.8,
 'Other PerformingArts': 0.0,
 'Other SocialSciences': 1.0,
 'Other SpecialNeeds': 0.7821782178217822,
 'Other TeamSports': 1.0,
 'Other VisualArts': 0.8125,
 'Other Warmth Care_Hunger': 1.0,
 'ParentInvolvement': 0.5454545454545454,
 'ParentInvolvement PerformingArts': 1.0,
 'ParentInvolvement SocialSciences': 0.8,
 'ParentInvolvement SpecialNeeds': 0.6,
 'ParentInvolvement TeamSports': 1.0,
 'ParentInvolvement VisualArts': 0.9166666666666666,
 'ParentInvolvement Warmth Care_Hunger': 1.0,
 'PerformingArts': 0.8970588235294118,
 'PerformingArts SocialSciences': 0.0,
 'PerformingArts SpecialNeeds': 0.8333333333333334,
 'PerformingArts TeamSports': 0.6666666666666666,
 'PerformingArts VisualArts': 0.7586206896551724,
 'SocialSciences': 0.8181818181818182,
 'SocialSciences SpecialNeeds': 0.9285714285714286,
 'SocialSciences VisualArts': 0.9444444444444444,
 'SpecialNeeds': 0.813598166539343,
 'SpecialNeeds TeamSports': 0.8888888888888888,
 'SpecialNeeds VisualArts': 0.8105263157894737,
 'SpecialNeeds Warmth Care_Hunger': 1.0,
 'TeamSports': 0.817629179331307,
 'TeamSports VisualArts': 0.5,
 'VisualArts': 0.8355555555555556,
 'VisualArts Warmth Care_Hunger': 0.0,
 'Warmth Care_Hunger': 0.937046004842615}
```

In [77]:

```
subcat_neg_train = []
subcat_pos_train = []
for i in project_data_train['clean_subcategories']:
    subcat_neg_train.append(subcat_0_train[i])
    subcat_pos_train.append(subcat_1_train[i])
project_data_train['subcat_0'] = subcat_neg_train
project_data_train['subcat_1'] = subcat_pos_train
```

In [78]:

```
state_0_train
```

Out[78]:

```
{'AK': 0.1523809523809524,
 'AL': 0.1547169811320754 7,
 'AR': 0.18627450980392157,
 'AZ': 0.15109034267912771,
 'CA': 0.14046610169491525,
 'CO': 0.162534435261708,
```

```
 'CT': 0.10477941176470588,
 'DC': 0.22093023255813954,
 'DE': 0.1320754716981132,
 'FL': 0.1797872340425532,
 'GA': 0.1630794701986755,
 'HI': 0.17088607594936708,
 'IA': 0.13658536585365855,
 'ID': 0.19170984455958548,
 'IL': 0.16310975609756098,
 'IN': 0.16046213093709885,
 'KS': 0.1092896174863388,
 'KY': 0.12781954887218044,
 'LA': 0.16759002770083103,
 'MA': 0.16204986149584488,
 'MD': 0.16997792494481237,
 'ME': 0.14965986394557823,
 'MI': 0.16666666666666666,
 'MN': 0.15873015873015872,
 'MO': 0.13745271122320302,
 'MS': 0.16470588235294117,
 'MT': 0.21739130434782608,
 'NC': 0.13966836734693877,
 'ND': 0.07317073170731707,
 'NE': 0.18478260869565216,
 'NH': 0.09782608695652174,
 'NJ': 0.17691154422788605,
 'NM': 0.14450867052023122,
 'NV': 0.134228187919463 1,
 'NY': 0.13895015438906044,
 'OH': 0.11475409836065574,
 'OK': 0.15988779803646563,
 'OR': 0.18087855297157623,
 'PA': 0.15817409766454352,
 'RI': 0.14942528735632185,
 'SC': 0.1527777777777778,
 'SD': 0.15306122448979592,
 'TN': 0.15503875968992248,
 'TX': 0.18664909969257795,
 'UT': 0.16827852998065765,
 'VA': 0.1430817610062893,
 'VT': 0.23809523809523808,
 'WA': 0.11869031377899045,
 'WI': 0.16304347826086957,
 'WV': 0.13548387096774195,
 'WY': 0.20588235294117646}
```

In [79]:

```
state_1_train
```

Out[79]:

```
{'AK': 0.8476190476190476,
 'AL': 0.8452830188679246,
 'AR': 0.8137254901960784,
 'AZ': 0.8489096573208723,
 'CA': 0.8595338983050848,
 'CO': 0.837465564738292,
 'CT': 0.8952205882352942,
 'DC': 0.7790697674418605,
 'DE': 0.8679245283018868,
 'FL': 0.8202127659574469,
 'GA': 0.8369205298013245,
 'HI': 0.8291139240506329,
 'IA': 0.863414634146341 5,
 'ID': 0.8082901554404145,
 'IL': 0.836890243902439,
 'IN': 0.8395378690629012,
 'KS': 0.8907103825136612,
 'KY': 0.8721804511278195,
 'LA': 0.832409972299169,
 'MA': 0.8379501385041551,
 'MD': 0.8300220750551877,
 'ME': 0.8503401360544217,
 'MI': 0.8333333333333334,
```

```
'MN': 0.8412698412698413,
 'MO': 0.862547288776797,
 'MS': 0.8352941176470589,
 'MT': 0.782608695652174,
 'NC': 0.8603316326530612,
 'ND': 0.926829268292683,
 'NE': 0.8152173913043478,
 'NH': 0.9021739130434783,
 'NJ': 0.823088455772114,
 'NM': 0.8554913294797688,
 'NV': 0.8657718120805369,
 'NY': 0.8610498456109396,
 'OH': 0.8852459016393442,
 'OK': 0.8401122019635343,
 'OR': 0.8191214470284238,
 'PA': 0.8418259023354565,
 'RI': 0.8505747126436781,
 'SC': 0.8472222222222222,
 'SD': 0.8469387755102041,
 'TN': 0.8449612403100775,
 'TX': 0.8133509003074221,
 'UT': 0.8317214700193424,
 'VA': 0.8569182389937107,
 'VT': 0.7619047619047619,
 'WA': 0.8813096862210096,
 'WI': 0.8369565217391305,
 'WV': 0.864516129032258,
 'WY': 0.7941176470588235}
```

In [80]:

```
state_neg_train = []
state_pos_train = []
for i in project_data_train['school_state']:
    state_neg_train.append(state_0_train[i])
    state_pos_train.append(state_1_train[i])
project_data_train['state_0'] = state_neg_train
project_data_train['state_1'] = state_pos_train
```

In [81]:

```
prefix_0_train
```

Out[81]:

```
{'Dr': 0.5,
 'Mr': 0.16078551702976374,
 'Mrs': 0.14822666590271014,
 'Ms': 0.15854364915184113,
 'Teacher': 0.19971469329529243}
```

In [82]:

```
prefix_1_train
```

Out[82]:

```
{'Dr': 0.5,
 'Mr': 0.8392144829702363,
 'Mrs': 0.8517733340972898,
 'Ms': 0.8414563508481588,
 'Teacher': 0.8002853067047075}
```

In [83]:

```
prefix_neg_train = []
prefix_pos_train = []
for i in project_data_train['teacher_prefix']:
    prefix_neg_train.append(prefix_0_train[i])
    prefix_pos_train.append(prefix_1_train[i])
project_data_train['prefix_0'] = prefix_neg_train
project_data_train['prefix_1'] = prefix_pos_train
```

In [84]:

```
grad_cat_0_train
```

Out[84]:

```
{'Grades_3_5': 0.14883234576507495,
 'Grades_6_8': 0.16423712342079688,
 'Grades_9_12': 0.17179951690821257,
 'Grades_PreK_2': 0.15080710547652393}
```

In [85]:

```
grad_cat_1_train
```

Out[85]:

```
{'Grades_3_5': 0.851167654234925,
 'Grades_6_8': 0.8357628765792031,
 'Grades_9_12': 0.8282004830917874,
 'Grades_PreK_2': 0.8491928945234761}
```

In [86]:

```
grade_neg_train = []
grade_pos_train = []
for i in project_data_train['project_grade_category']:
    grade_neg_train.append(grad_cat_0_train[i])
    grade_pos_train.append(grad_cat_1_train[i])
project_data_train['grade_0'] = grade_neg_train
project_data_train['grade_1'] = grade_pos_train
```

In [87]:

```
project_data_train.columns
```

Out[87]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'school_state',
       'project_submitted_datetime', 'project_title',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'clean_categories',
       'clean_subcategories', 'project_grade_category', 'teacher_prefix',
       'essay', 'preprocessed_essays', 'essay_word_count',
       'preprocessed_titles', 'title_word_count', 'neg', 'neu', 'pos',
       'compound', 'price', 'quantity', 'cat_0', 'cat_1', 'subcat_0',
       'subcat_1', 'state_0', 'state_1', 'prefix_0', 'prefix_1', 'grade_0',
       'grade_1'],
      dtype='object')
```

In [88]:

```
project_data_train.head()
```

Out[88]:

	Unnamed: 0	id	teacher_id	school_state	project_submitted_datetime	project_title	project_resource_su
0	150799	p143868	79035a795a20edf0792b390535afde8d	VA	2016-07-30 18:03:38	reading is fundamental	My students need tv and learn a
1	110848	p057203	13465062735441af726acd10d7ed50ac	CO	2016-12-23 15:23:35	character education through early morning bask...	My studer basketballs to c
2	52340	p037393	21f2243d51d566e486a8f429af75d7dc	WI	2016-09-13 15:29:02	bounce and learn	My students trampoline t bc

	Unnamed: 0	id	teacher_id	school_state	project_submitted_datetime	project_title	project_resource_summary
3	62498	p177980	17257472e5549a7bd0cba82635a40b8e	MA	2016-09-25 19:42:40	redecorate our dramatic play center!	My students ne furniture for the d
4	8722	p056227	d02a6876dee2ab709295b00bd3920859	CA	2016-08-13 03:05:03	encouraging healthy eating through dramatic play!	My students need a va examples c

5 rows × 33 columns

In [89]:

```
cat_0_test
```

Out[89]:

```
{'AppliedLearning': 0.1837837837837838,
 'AppliedLearning Health_Sports': 0.19736842105263158,
 'AppliedLearning History_Civics': 0.2608695652173913,
 'AppliedLearning Literacy_Language': 0.13069908814589665,
 'AppliedLearning Math_Science': 0.20218579234972678,
 'AppliedLearning Music_Arts': 0.22033898305084745,
 'AppliedLearning SpecialNeeds': 0.1774891774891775,
 'AppliedLearning Warmth Care_Hunger': 0.5,
 'Health_Sports': 0.1651376146788991,
 'Health_Sports AppliedLearning': 0.14285714285714285,
 'Health_Sports History_Civics': 0.1,
 'Health_Sports Literacy_Language': 0.15447154471544716,
 'Health_Sports Math_Science': 0.22916666666666666,
 'Health_Sports Music_Arts': 0.2777777777777778,
 'Health_Sports SpecialNeeds': 0.1297071129707113,
 'Health_Sports Warmth Care_Hunger': 0.0,
 'History_Civics': 0.15789473684210525,
 'History_Civics AppliedLearning': 0.3,
 'History_Civics Health_Sports': 0.0,
 'History_Civics Literacy_Language': 0.08333333333333333,
 'History_Civics Math_Science': 0.12280701754385964,
 'History_Civics Music_Arts': 0.17777777777777778,
 'History_Civics SpecialNeeds': 0.26666666666666666,
 'Literacy_Language': 0.1334798132381214,
 'Literacy_Language AppliedLearning': 0.12087912087912088,
 'Literacy_Language Health_Sports': 0.18181818181818182,
 'Literacy_Language History_Civics': 0.08620689655172414,
 'Literacy_Language Math_Science': 0.135752688172043,
 'Literacy_Language Music_Arts': 0.19083969465648856,
 'Literacy_Language SpecialNeeds': 0.13682432432432431,
 'Math_Science': 0.17647058823529413,
 'Math_Science AppliedLearning': 0.18274111675126903,
 'Math_Science Health_Sports': 0.23333333333333334,
 'Math_Science History_Civics': 0.16853932584269662,
 'Math_Science Literacy_Language': 0.1398176291793313,
 'Math_Science Music_Arts': 0.1619718309859155,
 'Math_Science SpecialNeeds': 0.1595744680851064,
 'Math_Science Warmth Care_Hunger': 0.0,
 'Music_Arts': 0.1629139072847682,
 'Music_Arts Health_Sports': 0.25,
 'Music_Arts History_Civics': 0.25,
 'Music_Arts SpecialNeeds': 0.12903225806451613,
 'SpecialNeeds': 0.18874172185430463,
 'SpecialNeeds Health_Sports': 0.6,
 'SpecialNeeds Music_Arts': 0.17777777777777778,
 'SpecialNeeds Warmth Care_Hunger': 0.25,
 'Warmth Care_Hunger': 0.09844559585492228}
```

In [90]:

```
cat_1_test
```

Out[90]:

```
{'AppliedLearning': 0.8162162162162162,
 'AppliedLearning Health_Sports': 0.8026315789473685,
 'AppliedLearning History_Civics': 0.7391304347826086,
```

```
 'AppliedLearning Literacy_Language': 0.8693009118541033,
 'AppliedLearning Math_Science': 0.7978142076502732,
 'AppliedLearning Music_Arts': 0.7796610169491526,
 'AppliedLearning SpecialNeeds': 0.8225108225108225,
 'AppliedLearning Warmth Care_Hunger': 0.5,
 'Health_Sports': 0.8348623853211009,
 'Health_Sports AppliedLearning': 0.8571428571428571,
 'Health_Sports History_Civics': 0.9,
 'Health_Sports Literacy_Language': 0.8455284552845529,
 'Health_Sports Math_Science': 0.7708333333333334,
 'Health_Sports Music_Arts': 0.7222222222222222,
 'Health_Sports SpecialNeeds': 0.8702928870292888,
 'Health_Sports Warmth Care_Hunger': 1.0,
 'History_Civics': 0.8421052631578947,
 'History_Civics AppliedLearning': 0.7,
 'History_Civics Health_Sports': 1.0,
 'History_Civics Literacy_Language': 0.9166666666666666,
 'History_Civics Math_Science': 0.8771929824561403,
 'History_Civics Music_Arts': 0.8222222222222222,
 'History_Civics SpecialNeeds': 0.7333333333333333,
 'Literacy_Language': 0.8665201867618786,
 'Literacy_Language AppliedLearning': 0.8791208791208791,
 'Literacy_Language Health_Sports': 0.8181818181818182,
 'Literacy_Language History_Civics': 0.9137931034482759,
 'Literacy_Language Math_Science': 0.864247311827957,
 'Literacy_Language Music_Arts': 0.8091603053435115,
 'Literacy_Language SpecialNeeds': 0.8631756756756757,
 'Math_Science': 0.8235294117647058,
 'Math_Science AppliedLearning': 0.817258883248731,
 'Math_Science Health_Sports': 0.7666666666666667,
 'Math_Science History_Civics': 0.8314606741573034,
 'Math_Science Literacy_Language': 0.8601823708206687,
 'Math_Science Music_Arts': 0.8380281690140845,
 'Math_Science SpecialNeeds': 0.8404255319148937,
 'Math_Science Warmth Care_Hunger': 1.0,
 'Music_Arts': 0.8370860927152318,
 'Music_Arts Health_Sports': 0.75,
 'Music_Arts History_Civics': 0.75,
 'Music_Arts SpecialNeeds': 0.8709677419354839,
 'SpecialNeeds': 0.8112582781456954,
 'SpecialNeeds Health_Sports': 0.4,
 'SpecialNeeds Music_Arts': 0.8222222222222222,
 'SpecialNeeds Warmth Care_Hunger': 0.75,
 'Warmth Care_Hunger': 0.9015544041450777}
```

In [91]:

```
cat_neg_test = []
cat_pos_test = []
for i in project_data_test['clean_categories']:
    cat_neg_test.append(cat_0_test[i])
    cat_pos_test.append(cat_1_test[i])
project_data_test['cat_0'] = cat_neg_test
project_data_test['cat_1'] = cat_pos_test
```

In [92]:

```
subcat_0_test
```

Out[92]:

```
{'AppliedSciences': 0.19154929577464788,
 'AppliedSciences CharacterEducation': 0.2222222222222222,
 'AppliedSciences Civics_Government': 0.0,
 'AppliedSciences College_CareerPrep': 0.21739130434782608,
 'AppliedSciences CommunityService': 0.0,
 'AppliedSciences ESL': 0.2,
 'AppliedSciences EarlyDevelopment': 0.21875,
 'AppliedSciences Economics': 1.0,
 'AppliedSciences EnvironmentalScience': 0.14173228346456693,
 'AppliedSciences Extracurricular': 0.09523809523809523,
 'AppliedSciences ForeignLanguages': 1.0,
 'AppliedSciences Gym_Fitness': 0.5,
 'AppliedSciences Health_LifeScience': 0.17346938775510204,
 'AppliedSciences Health_Wellness': 0.0,
```

```
'AppliedSciences History_Geography': 0.2222222222222222,
'AppliedSciences Literacy': 0.18681318681318682,
'AppliedSciences Literature_Writing': 0.12280701754385964,
'AppliedSciences Mathematics': 0.17659137577002054,
'AppliedSciences Music': 0.2,
'AppliedSciences NutritionEducation': 0.0,
'AppliedSciences Other': 0.15384615384615385,
'AppliedSciences ParentInvolvement': 0.2,
'AppliedSciences PerformingArts': 0.0,
'AppliedSciences SocialSciences': 0.0,
'AppliedSciences SpecialNeeds': 0.11864406779661017,
'AppliedSciences TeamSports': 0.0,
'AppliedSciences VisualArts': 0.17699115044247787,
'CharacterEducation': 0.24489795918367346,
'CharacterEducation College_CareerPrep': 0.058823529411764705,
'CharacterEducation CommunityService': 0.5714285714285714,
'CharacterEducation ESL': 1.0,
'CharacterEducation EarlyDevelopment': 0.20833333333333334,
'CharacterEducation Economics': 1.0,
'CharacterEducation EnvironmentalScience': 0.5,
'CharacterEducation Extracurricular': 0.2222222222222222,
'CharacterEducation Gym_Fitness': 0.5,
'CharacterEducation Health_LifeScience': 0.4,
'CharacterEducation Health_Wellness': 0.09090909090909091,
'CharacterEducation History_Geography': 0.0,
'CharacterEducation Literacy': 0.1111111111111111,
'CharacterEducation Literature_Writing': 0.20689655172413793,
'CharacterEducation Mathematics': 0.14285714285714285,
'CharacterEducation Music': 0.16666666666666666,
'CharacterEducation NutritionEducation': 0.0,
'CharacterEducation Other': 0.21052631578947367,
'CharacterEducation ParentInvolvement': 0.0,
'CharacterEducation SocialSciences': 0.3333333333333333,
'CharacterEducation SpecialNeeds': 0.2857142857142857,
'CharacterEducation TeamSports': 0.3333333333333333,
'CharacterEducation VisualArts': 0.3076923076923077,
'CharacterEducation Warmth Care_Hunger': 1.0,
'Civics_Government': 0.2727272727272727,
'Civics_Government College_CareerPrep': 0.0,
'Civics_Government CommunityService': 0.3333333333333333,
'Civics_Government Economics': 0.0,
'Civics_Government Extracurricular': 1.0,
'Civics_Government FinancialLiteracy': 1.0,
'Civics_Government Health_LifeScience': 0.0,
'Civics_Government Health_Wellness': 0.0,
'Civics_Government History_Geography': 0.11428571428571428,
'Civics_Government Literacy': 0.05555555555555555,
'Civics_Government Literature_Writing': 0.15384615384615385,
'Civics_Government Mathematics': 0.25,
'Civics_Government SocialSciences': 0.05555555555555555,
'Civics_Government SpecialNeeds': 0.5,
'Civics_Government TeamSports': 0.0,
'Civics_Government VisualArts': 0.5,
'College_CareerPrep': 0.1111111111111111,
'College_CareerPrep CommunityService': 0.5,
'College_CareerPrep ESL': 0.0,
'College_CareerPrep EarlyDevelopment': 0.0,
'College_CareerPrep EnvironmentalScience': 0.25,
'College_CareerPrep Extracurricular': 0.125,
'College_CareerPrep FinancialLiteracy': 0.3333333333333333,
'College_CareerPrep ForeignLanguages': 0.0,
'College_CareerPrep Health_LifeScience': 0.14285714285714285,
'College_CareerPrep Health_Wellness': 0.25,
'College_CareerPrep History_Geography': 1.0,
'College_CareerPrep Literacy': 0.1388888888888889,
'College_CareerPrep Literature_Writing': 0.043478260869565216,
'College_CareerPrep Mathematics': 0.1864406779661017,
'College_CareerPrep Music': 0.5,
'College_CareerPrep NutritionEducation': 0.0,
'College_CareerPrep Other': 0.3333333333333333,
'College_CareerPrep ParentInvolvement': 0.25,
'College_CareerPrep PerformingArts': 0.5,
'College_CareerPrep SocialSciences': 0.0,
'College_CareerPrep SpecialNeeds': 0.2,
'College_CareerPrep VisualArts': 0.058823529411764705,
'CommunityService': 0.0,
'CommunityService Economics': 1.0,
```

```
 'CommunityService EnvironmentalScience': 0.1111111111111111,
 'CommunityService Extracurricular': 0.16666666666666666,
 'CommunityService Health_LifeScience': 0.0,
 'CommunityService Health_Wellness': 0.2,
 'CommunityService Literacy': 0.5,
 'CommunityService Literature_Writing': 0.0,
 'CommunityService Mathematics': 0.3333333333333333,
 'CommunityService NutritionEducation': 1.0,
 'CommunityService ParentInvolvement': 0.0,
 'CommunityService SocialSciences': 0.0,
 'CommunityService SpecialNeeds': 0.0,
 'CommunityService VisualArts': 0.0,
 'ESL': 0.11764705882352941,
 'ESL EarlyDevelopment': 0.09090909090909091,
 'ESL EnvironmentalScience': 0.0,
 'ESL FinancialLiteracy': 0.0,
 'ESL ForeignLanguages': 0.3076923076923077,
 'ESL Health_LifeScience': 0.0,
 'ESL Health_Wellness': 0.16666666666666666,
 'ESL History_Geography': 0.0,
 'ESL Literacy': 0.1317365269461078,
 'ESL Literature_Writing': 0.13333333333333333,
 'ESL Mathematics': 0.11764705882352941,
 'ESL ParentInvolvement': 0.0,
 'ESL PerformingArts': 0.6666666666666666,
 'ESL SocialSciences': 0.0,
 'ESL SpecialNeeds': 0.1111111111111111,
 'ESL VisualArts': 0.0,
 'EarlyDevelopment': 0.22695035460992907,
 'EarlyDevelopment EnvironmentalScience': 0.2857142857142857,
 'EarlyDevelopment Extracurricular': 0.0,
 'EarlyDevelopment FinancialLiteracy': 0.0,
 'EarlyDevelopment Gym_Fitness': 0.3333333333333333,
 'EarlyDevelopment Health_LifeScience': 0.0,
 'EarlyDevelopment Health_Wellness': 0.20588235294117646,
 'EarlyDevelopment History_Geography': 0.0,
 'EarlyDevelopment Literacy': 0.12121212121212122,
 'EarlyDevelopment Literature_Writing': 0.15789473684210525,
 'EarlyDevelopment Mathematics': 0.21568627450980393,
 'EarlyDevelopment Music': 0.0,
 'EarlyDevelopment NutritionEducation': 0.0,
 'EarlyDevelopment Other': 0.19230769230769232,
 'EarlyDevelopment ParentInvolvement': 0.25,
 'EarlyDevelopment PerformingArts': 0.0,
 'EarlyDevelopment SocialSciences': 0.3333333333333333,
 'EarlyDevelopment SpecialNeeds': 0.15966386554621848,
 'EarlyDevelopment TeamSports': 0.0,
 'EarlyDevelopment VisualArts': 0.3181818181818182,
 'EarlyDevelopment Warmth Care_Hunger': 0.0,
 'Economics': 0.0,
 'Economics EnvironmentalScience': 0.0,
 'Economics FinancialLiteracy': 0.46153846153846156,
 'Economics History_Geography': 0.2,
 'Economics Literacy': 0.0,
 'Economics Literature_Writing': 0.0,
 'Economics Mathematics': 0.0,
 'Economics SocialSciences': 0.0,
 'Economics SpecialNeeds': 0.5,
 'Economics VisualArts': 0.5,
 'EnvironmentalScience': 0.1949685534591195,
 'EnvironmentalScience Extracurricular': 1.0,
 'EnvironmentalScience Health_LifeScience': 0.2230769230769231,
 'EnvironmentalScience Health_Wellness': 0.2,
 'EnvironmentalScience History_Geography': 0.2608695652173913,
 'EnvironmentalScience Literacy': 0.11764705882352941,
 'EnvironmentalScience Literature_Writing': 0.11627906976744186,
 'EnvironmentalScience Mathematics': 0.17424242424242425,
 'EnvironmentalScience Music': 0.0,
 'EnvironmentalScience NutritionEducation': 0.5,
 'EnvironmentalScience Other': 1.0,
 'EnvironmentalScience ParentInvolvement': 0.0,
 'EnvironmentalScience SocialSciences': 0.2,
 'EnvironmentalScience SpecialNeeds': 0.08695652173913043,
 'EnvironmentalScience TeamSports': 0.0,
 'EnvironmentalScience VisualArts': 0.14705882352941177,
 'Extracurricular': 0.05263157894736842,
 'Extracurricular Gym_Fitness': 0.0,
```

```
'Extracurricular Health_LifeScience': 0.5,
'Extracurricular Literacy': 0.3333333333333333,
'Extracurricular Literature_Writing': 0.0,
'Extracurricular Mathematics': 0.18181818181818182,
'Extracurricular Music': 0.4,
'Extracurricular Other': 0.0,
'Extracurricular PerformingArts': 0.0,
'Extracurricular SpecialNeeds': 0.0,
'Extracurricular TeamSports': 0.0,
'Extracurricular VisualArts': 0.11764705882352941,
'FinancialLiteracy': 0.13636363636363635,
'FinancialLiteracy Health_Wellness': 0.0,
'FinancialLiteracy History_Geography': 0.0,
'FinancialLiteracy Literacy': 0.0,
'FinancialLiteracy Literature_Writing': 0.0,
'FinancialLiteracy Mathematics': 0.09523809523809523,
'FinancialLiteracy Other': 0.0,
'FinancialLiteracy SpecialNeeds': 0.3333333333333333,
'ForeignLanguages': 0.19230769230769232,
'ForeignLanguages Health_LifeScience': 0.0,
'ForeignLanguages Health_Wellness': 0.0,
'ForeignLanguages History_Geography': 0.0,
'ForeignLanguages Literacy': 0.09302325581395349,
'ForeignLanguages Literature_Writing': 0.3076923076923077,
'ForeignLanguages Mathematics': 0.42857142857142855,
'ForeignLanguages Music': 0.0,
'ForeignLanguages SocialSciences': 0.0,
'ForeignLanguages SpecialNeeds': 0.0,
'ForeignLanguages VisualArts': 1.0,
'Gym_Fitness': 0.17613636363636365,
'Gym_Fitness Health_Wellness': 0.15249266862170088,
'Gym_Fitness History_Geography': 0.0,
'Gym_Fitness Literacy': 0.5,
'Gym_Fitness Literature_Writing': 0.5,
'Gym_Fitness Mathematics': 0.4,
'Gym_Fitness Music': 0.0,
'Gym_Fitness NutritionEducation': 0.1,
'Gym_Fitness Other': 0.0,
'Gym_Fitness ParentInvolvement': 0.0,
'Gym_Fitness PerformingArts': 1.0,
'Gym_Fitness SpecialNeeds': 0.2,
'Gym_Fitness TeamSports': 0.24719101123595505,
'Gym_Fitness VisualArts': 1.0,
'Health_LifeScience': 0.17985611510791366,
'Health_LifeScience Health_Wellness': 0.2692307692307692,
'Health_LifeScience History_Geography': 0.08333333333333333,
'Health_LifeScience Literacy': 0.14705882352941177,
'Health_LifeScience Literature_Writing': 0.0,
'Health_LifeScience Mathematics': 0.1917808219178082,
'Health_LifeScience Music': 0.0,
'Health_LifeScience NutritionEducation': 0.2,
'Health_LifeScience ParentInvolvement': 0.0,
'Health_LifeScience SocialSciences': 0.09090909090909091,
'Health_LifeScience SpecialNeeds': 0.2222222222222222,
'Health_LifeScience TeamSports': 0.0,
'Health_LifeScience VisualArts': 0.23076923076923078,
'Health_LifeScience Warmth Care_Hunger': 0.0,
'Health_Wellness': 0.13123844731977818,
'Health_Wellness History_Geography': 0.0,
'Health_Wellness Literacy': 0.14492753623188406,
'Health_Wellness Literature_Writing': 0.10869565217391304,
'Health_Wellness Mathematics': 0.20930232558139536,
'Health_Wellness Music': 0.3333333333333333,
'Health_Wellness NutritionEducation': 0.168,
'Health_Wellness Other': 0.16666666666666666,
'Health_Wellness ParentInvolvement': 0.0,
'Health_Wellness PerformingArts': 0.0,
'Health_Wellness SocialSciences': 0.5,
'Health_Wellness SpecialNeeds': 0.1232876712328767,
'Health_Wellness TeamSports': 0.24528301886792453,
'Health_Wellness VisualArts': 0.0,
'Health_Wellness Warmth Care_Hunger': 0.0,
'History_Geography': 0.1609195402298850 6,
'History_Geography Literacy': 0.0759493670886076,
'History_Geography Literature_Writing': 0.09195402298850575,
'History_Geography Mathematics': 0.17391304347826086,
'History_Geography Music': 0.5,
```

```
 'History_Geography Other': 0.5,
 'History_Geography PerformingArts': 0.0,
 'History_Geography SocialSciences': 0.20408163265306123,
 'History_Geography SpecialNeeds': 0.13333333333333333,
 'History_Geography VisualArts': 0.14814814814814814,
 'Literacy': 0.12668463611859837,
 'Literacy Literature_Writing': 0.1173054587688734,
 'Literacy Mathematics': 0.13272311212814644,
 'Literacy Music': 0.16666666666666666,
 'Literacy Other': 0.08695652173913043,
 'Literacy ParentInvolvement': 0.12,
 'Literacy PerformingArts': 0.26666666666666666,
 'Literacy SocialSciences': 0.07547169811320754,
 'Literacy SpecialNeeds': 0.14054054054054055,
 'Literacy TeamSports': 0.25,
 'Literacy VisualArts': 0.17894736842105263,
 'Literature_Writing': 0.1638591117917305,
 'Literature_Writing Mathematics': 0.13990825688073394,
 'Literature_Writing Music': 0.0,
 'Literature_Writing Other': 0.14285714285714285,
 'Literature_Writing ParentInvolvement': 0.2222222222222222,
 'Literature_Writing PerformingArts': 0.16666666666666666,
 'Literature_Writing SocialSciences': 0.11764705882352941,
 'Literature_Writing SpecialNeeds': 0.13471502590673576,
 'Literature_Writing VisualArts': 0.1926605504587156,
 'Mathematics': 0.16299019607843138,
 'Mathematics Music': 0.0,
 'Mathematics Other': 0.16666666666666666,
 'Mathematics ParentInvolvement': 0.07142857142857142,
 'Mathematics PerformingArts': 0.5,
 'Mathematics SocialSciences': 0.14285714285714285,
 'Mathematics SpecialNeeds': 0.17582417582417584,
 'Mathematics VisualArts': 0.14606741573033707,
 'Music': 0.11848341232227488,
 'Music PerformingArts': 0.1456953642384106,
 'Music SocialSciences': 0.5,
 'Music SpecialNeeds': 0.1111111111111111,
 'Music TeamSports': 0.5,
 'Music VisualArts': 0.42857142857142855,
 'NutritionEducation': 0.22727272727272727,
 'NutritionEducation Other': 0.0,
 'NutritionEducation SocialSciences': 0.0,
 'NutritionEducation SpecialNeeds': 0.2,
 'NutritionEducation TeamSports': 1.0,
 'Other': 0.15,
 'Other SocialSciences': 0.0,
 'Other SpecialNeeds': 0.16326530612244897,
 'Other VisualArts': 0.3333333333333333,
 'ParentInvolvement': 0.0,
 'ParentInvolvement PerformingArts': 0.0,
 'ParentInvolvement SpecialNeeds': 0.25,
 'ParentInvolvement VisualArts': 0.2,
 'PerformingArts': 0.140625,
 'PerformingArts SocialSciences': 0.0,
 'PerformingArts SpecialNeeds': 0.25,
 'PerformingArts TeamSports': 0.0,
 'PerformingArts VisualArts': 0.4,
 'SocialSciences': 0.09090909090909091,
 'SocialSciences SpecialNeeds': 0.4,
 'SocialSciences VisualArts': 0.1,
 'SpecialNeeds': 0.18874172185430463,
 'SpecialNeeds TeamSports': 0.6,
 'SpecialNeeds VisualArts': 0.17777777777777778,
 'SpecialNeeds Warmth Care_Hunger': 0.25,
 'TeamSports': 0.2054794520547945,
 'VisualArts': 0.18892508143322476,
 'Warmth Care_Hunger': 0.09844559585492228}
```

In [93]:

```
subcat_1_test
```

Out[93]:

```
{'AppliedSciences': 0.8084507042253521,
 'AppliedSciences CharacterEducation': 0.7777777777777778,
```

'AppliedSciences CharacterEducation': 0.7777777777777778,
'AppliedSciences Civics_Government': 1.0,
'AppliedSciences College_CareerPrep': 0.782608695652174,
'AppliedSciences CommunityService': 1.0,
'AppliedSciences ESL': 0.8,
'AppliedSciences EarlyDevelopment': 0.78125,
'AppliedSciences Economics': 0.0,
'AppliedSciences EnvironmentalScience': 0.8582677165354331,
'AppliedSciences Extracurricular': 0.9047619047619048,
'AppliedSciences ForeignLanguages': 0.0,
'AppliedSciences Gym_Fitness': 0.5,
'AppliedSciences Health_LifeScience': 0.826530612244898,
'AppliedSciences Health_Wellness': 1.0,
'AppliedSciences History_Geography': 0.7777777777777778,
'AppliedSciences Literacy': 0.8131868131868132,
'AppliedSciences Literature_Writing': 0.8771929824561403,
'AppliedSciences Mathematics': 0.8234086242299795,
'AppliedSciences Music': 0.8,
'AppliedSciences NutritionEducation': 1.0,
'AppliedSciences Other': 0.8461538461538461,
'AppliedSciences ParentInvolvement': 0.8,
'AppliedSciences PerformingArts': 1.0,
'AppliedSciences SocialSciences': 1.0,
'AppliedSciences SpecialNeeds': 0.8813559322033898,
'AppliedSciences TeamSports': 1.0,
'AppliedSciences VisualArts': 0.8230088495575221,
'CharacterEducation': 0.7551020408163265,
'CharacterEducation College_CareerPrep': 0.9411764705882353,
'CharacterEducation CommunityService': 0.42857142857142855,
'CharacterEducation ESL': 0.0,
'CharacterEducation EarlyDevelopment': 0.7916666666666666,
'CharacterEducation Economics': 0.0,
'CharacterEducation EnvironmentalScience': 0.5,
'CharacterEducation Extracurricular': 0.7777777777777778,
'CharacterEducation Gym_Fitness': 0.5,
'CharacterEducation Health_LifeScience': 0.6,
'CharacterEducation Health_Wellness': 0.9090909090909091,
'CharacterEducation History_Geography': 1.0,
'CharacterEducation Literacy': 0.8888888888888888,
'CharacterEducation Literature_Writing': 0.7931034482758621,
'CharacterEducation Mathematics': 0.8571428571428571,
'CharacterEducation Music': 0.8333333333333334,
'CharacterEducation NutritionEducation': 1.0,
'CharacterEducation Other': 0.7894736842105263,
'CharacterEducation ParentInvolvement': 1.0,
'CharacterEducation SocialSciences': 0.6666666666666666,
'CharacterEducation SpecialNeeds': 0.7142857142857143,
'CharacterEducation TeamSports': 0.6666666666666666,
'CharacterEducation VisualArts': 0.6923076923076923,
'CharacterEducation Warmth Care_Hunger': 0.0,
'Civics_Government': 0.7272727272727273,
'Civics_Government College_CareerPrep': 1.0,
'Civics_Government CommunityService': 0.6666666666666666,
'Civics_Government Economics': 1.0,
'Civics_Government Extracurricular': 0.0,
'Civics_Government FinancialLiteracy': 0.0,
'Civics_Government Health_LifeScience': 1.0,
'Civics_Government Health_Wellness': 1.0,
'Civics_Government History_Geography': 0.8857142857142857,
'Civics_Government Literacy': 0.9444444444444444,
'Civics_Government Literature_Writing': 0.8461538461538461,
'Civics_Government Mathematics': 0.75,
'Civics_Government SocialSciences': 0.9444444444444444,
'Civics_Government SpecialNeeds': 0.5,
'Civics_Government TeamSports': 1.0,
'Civics_Government VisualArts': 0.5,
'College_CareerPrep': 0.8888888888888888,
'College_CareerPrep CommunityService': 0.5,
'College_CareerPrep ESL': 1.0,
'College_CareerPrep EarlyDevelopment': 1.0,
'College_CareerPrep EnvironmentalScience': 0.75,
'College_CareerPrep Extracurricular': 0.875,
'College_CareerPrep FinancialLiteracy': 0.6666666666666666,
'College_CareerPrep ForeignLanguages': 1.0,
'College_CareerPrep Health_LifeScience': 0.8571428571428571,
'College_CareerPrep Health_Wellness': 0.75,
'College_CareerPrep History_Geography': 0.0,
'College_CareerPrep Literacy': 0.8611111111111112

```
'College_CareerPrep Literacy': 0.8611111111111112,
'College_CareerPrep Literature_Writing': 0.9565217391304348,
'College_CareerPrep Mathematics': 0.8135593220338984,
'College_CareerPrep Music': 0.5,
'College_CareerPrep NutritionEducation': 1.0,
'College_CareerPrep Other': 0.6666666666666666,
'College_CareerPrep ParentInvolvement': 0.75,
'College_CareerPrep PerformingArts': 0.5,
'College_CareerPrep SocialSciences': 1.0,
'College_CareerPrep SpecialNeeds': 0.8,
'College_CareerPrep VisualArts': 0.9411764705882353,
'CommunityService': 1.0,
'CommunityService Economics': 0.0,
'CommunityService EnvironmentalScience': 0.8888888888888888,
'CommunityService Extracurricular': 0.8333333333333334,
'CommunityService Health_LifeScience': 1.0,
'CommunityService Health_Wellness': 0.8,
'CommunityService Literacy': 0.5,
'CommunityService Literature_Writing': 1.0,
'CommunityService Mathematics': 0.6666666666666666,
'CommunityService NutritionEducation': 0.0,
'CommunityService ParentInvolvement': 1.0,
'CommunityService SocialSciences': 1.0,
'CommunityService SpecialNeeds': 1.0,
'CommunityService VisualArts': 1.0,
'ESL': 0.8823529411764706,
'ESL EarlyDevelopment': 0.9090909090909091,
'ESL EnvironmentalScience': 1.0,
'ESL FinancialLiteracy': 1.0,
'ESL ForeignLanguages': 0.6923076923076923,
'ESL Health_LifeScience': 1.0,
'ESL Health_Wellness': 0.8333333333333334,
'ESL History_Geography': 1.0,
'ESL Literacy': 0.8682634730538922,
'ESL Literature_Writing': 0.8666666666666667,
'ESL Mathematics': 0.8823529411764706,
'ESL ParentInvolvement': 1.0,
'ESL PerformingArts': 0.3333333333333333,
'ESL SocialSciences': 1.0,
'ESL SpecialNeeds': 0.8888888888888888,
'ESL VisualArts': 1.0,
'EarlyDevelopment': 0.7730496453900709,
'EarlyDevelopment EnvironmentalScience': 0.7142857142857143,
'EarlyDevelopment Extracurricular': 1.0,
'EarlyDevelopment FinancialLiteracy': 1.0,
'EarlyDevelopment Gym_Fitness': 0.6666666666666666,
'EarlyDevelopment Health_LifeScience': 1.0,
'EarlyDevelopment Health_Wellness': 0.7941176470588235,
'EarlyDevelopment History_Geography': 1.0,
'EarlyDevelopment Literacy': 0.8787878787878788,
'EarlyDevelopment Literature_Writing': 0.8421052631578947,
'EarlyDevelopment Mathematics': 0.7843137254901961,
'EarlyDevelopment Music': 1.0,
'EarlyDevelopment NutritionEducation': 1.0,
'EarlyDevelopment Other': 0.8076923076923077,
'EarlyDevelopment ParentInvolvement': 0.75,
'EarlyDevelopment PerformingArts': 1.0,
'EarlyDevelopment SocialSciences': 0.6666666666666666,
'EarlyDevelopment SpecialNeeds': 0.8403361344537815,
'EarlyDevelopment TeamSports': 1.0,
'EarlyDevelopment VisualArts': 0.6818181818181818,
'EarlyDevelopment Warmth Care_Hunger': 1.0,
'Economics': 1.0,
'Economics EnvironmentalScience': 1.0,
'Economics FinancialLiteracy': 0.5384615384615384,
'Economics History_Geography': 0.8,
'Economics Literacy': 1.0,
'Economics Literature_Writing': 1.0,
'Economics Mathematics': 1.0,
'Economics SocialSciences': 1.0,
'Economics SpecialNeeds': 0.5,
'Economics VisualArts': 0.5,
'EnvironmentalScience': 0.8050314465408805,
'EnvironmentalScience Extracurricular': 0.0,
'EnvironmentalScience Health_LifeScience': 0.7769230769230769,
'EnvironmentalScience Health_Wellness': 0.8,
'EnvironmentalScience History_Geography': 0.7391304347826086,
'EnvironmentalScience Literacy': 0.8823529411764706
```

'EnvironmentalScience Literacy': 0.8823529411764706,
'EnvironmentalScience Literature_Writing': 0.8837209302325582,
'EnvironmentalScience Mathematics': 0.8257575757575758,
'EnvironmentalScience Music': 1.0,
'EnvironmentalScience NutritionEducation': 0.5,
'EnvironmentalScience Other': 0.0,
'EnvironmentalScience ParentInvolvement': 1.0,
'EnvironmentalScience SocialSciences': 0.8,
'EnvironmentalScience SpecialNeeds': 0.9130434782608695,
'EnvironmentalScience TeamSports': 1.0,
'EnvironmentalScience VisualArts': 0.8529411764705882,
'Extracurricular': 0.9473684210526315,
'Extracurricular Gym_Fitness': 1.0,
'Extracurricular Health_LifeScience': 0.5,
'Extracurricular Literacy': 0.6666666666666666,
'Extracurricular Literature_Writing': 1.0,
'Extracurricular Mathematics': 0.8181818181818182,
'Extracurricular Music': 0.6,
'Extracurricular Other': 1.0,
'Extracurricular PerformingArts': 1.0,
'Extracurricular SpecialNeeds': 1.0,
'Extracurricular TeamSports': 1.0,
'Extracurricular VisualArts': 0.8823529411764706,
'FinancialLiteracy': 0.8636363636363636,
'FinancialLiteracy Health_Wellness': 1.0,
'FinancialLiteracy History_Geography': 1.0,
'FinancialLiteracy Literacy': 1.0,
'FinancialLiteracy Literature_Writing': 1.0,
'FinancialLiteracy Mathematics': 0.9047619047619048,
'FinancialLiteracy Other': 1.0,
'FinancialLiteracy SpecialNeeds': 0.6666666666666666,
'ForeignLanguages': 0.8076923076923077,
'ForeignLanguages Health_LifeScience': 1.0,
'ForeignLanguages Health_Wellness': 1.0,
'ForeignLanguages History_Geography': 1.0,
'ForeignLanguages Literacy': 0.9069767441860465,
'ForeignLanguages Literature_Writing': 0.6923076923076923,
'ForeignLanguages Mathematics': 0.5714285714285714,
'ForeignLanguages Music': 1.0,
'ForeignLanguages SocialSciences': 1.0,
'ForeignLanguages SpecialNeeds': 1.0,
'ForeignLanguages VisualArts': 0.0,
'Gym_Fitness': 0.8238636363636364,
'Gym_Fitness Health_Wellness': 0.8475073313782991,
'Gym_Fitness History_Geography': 1.0,
'Gym_Fitness Literacy': 0.5,
'Gym_Fitness Literature_Writing': 0.5,
'Gym_Fitness Mathematics': 0.6,
'Gym_Fitness Music': 1.0,
'Gym_Fitness NutritionEducation': 0.9,
'Gym_Fitness Other': 1.0,
'Gym_Fitness ParentInvolvement': 1.0,
'Gym_Fitness PerformingArts': 0.0,
'Gym_Fitness SpecialNeeds': 0.8,
'Gym_Fitness TeamSports': 0.7528089887640449,
'Gym_Fitness VisualArts': 0.0,
'Health_LifeScience': 0.8201438848920863,
'Health_LifeScience Health_Wellness': 0.7307692307692307,
'Health_LifeScience History_Geography': 0.9166666666666666,
'Health_LifeScience Literacy': 0.8529411764705882,
'Health_LifeScience Literature_Writing': 1.0,
'Health_LifeScience Mathematics': 0.8082191780821918,
'Health_LifeScience Music': 1.0,
'Health_LifeScience NutritionEducation': 0.8,
'Health_LifeScience ParentInvolvement': 1.0,
'Health_LifeScience SocialSciences': 0.9090909090909091,
'Health_LifeScience SpecialNeeds': 0.7777777777777778,
'Health_LifeScience TeamSports': 1.0,
'Health_LifeScience VisualArts': 0.7692307692307693,
'Health_LifeScience Warmth Care_Hunger': 1.0,
'Health_Wellness': 0.8687615526802218,
'Health_Wellness History_Geography': 1.0,
'Health_Wellness Literacy': 0.855072463768116,
'Health_Wellness Literature_Writing': 0.8913043478260869,
'Health_Wellness Mathematics': 0.7906976744186046,
'Health_Wellness Music': 0.6666666666666666,
'Health_Wellness NutritionEducation': 0.832,
'Health_Wellness Other': 0.8333333333333334,

```
'Health_Wellness Other': 0.8333333333333334,
'Health_Wellness ParentInvolvement': 1.0,
'Health_Wellness PerformingArts': 1.0,
'Health_Wellness SocialSciences': 0.5,
'Health_Wellness SpecialNeeds': 0.8767123287671232,
'Health_Wellness TeamSports': 0.7547169811320755,
'Health_Wellness VisualArts': 1.0,
'Health_Wellness Warmth Care_Hunger': 1.0,
'History_Geography': 0.8390804597701149,
'History_Geography Literacy': 0.9240506329113924,
'History_Geography Literature_Writing': 0.9080459770114943,
'History_Geography Mathematics': 0.8260869565217391,
'History_Geography Music': 0.5,
'History_Geography Other': 0.5,
'History_Geography PerformingArts': 1.0,
'History_Geography SocialSciences': 0.7959183673469388,
'History_Geography SpecialNeeds': 0.8666666666666667,
'History_Geography VisualArts': 0.8518518518518519,
'Literacy': 0.8733153638814016,
'Literacy Literature_Writing': 0.8826945412311266,
'Literacy Mathematics': 0.8672768878718535,
'Literacy Music': 0.8333333333333334,
'Literacy Other': 0.9130434782608695,
'Literacy ParentInvolvement': 0.88,
'Literacy PerformingArts': 0.7333333333333333,
'Literacy SocialSciences': 0.9245283018867925,
'Literacy SpecialNeeds': 0.8594594594594595,
'Literacy TeamSports': 0.75,
'Literacy VisualArts': 0.8210526315789474,
'Literature_Writing': 0.8361408882082695,
'Literature_Writing Mathematics': 0.8600917431192661,
'Literature_Writing Music': 1.0,
'Literature_Writing Other': 0.8571428571428571,
'Literature_Writing ParentInvolvement': 0.7777777777777778,
'Literature_Writing PerformingArts': 0.8333333333333334,
'Literature_Writing SocialSciences': 0.8823529411764706,
'Literature_Writing SpecialNeeds': 0.8652849740932642,
'Literature_Writing VisualArts': 0.8073394495412844,
'Mathematics': 0.8370098039215687,
'Mathematics Music': 1.0,
'Mathematics Other': 0.8333333333333334,
'Mathematics ParentInvolvement': 0.9285714285714286,
'Mathematics PerformingArts': 0.5,
'Mathematics SocialSciences': 0.8571428571428571,
'Mathematics SpecialNeeds': 0.8241758241758241,
'Mathematics VisualArts': 0.8539325842696629,
'Music': 0.8815165876777251,
'Music PerformingArts': 0.8543046357615894,
'Music SocialSciences': 0.5,
'Music SpecialNeeds': 0.8888888888888888,
'Music TeamSports': 0.5,
'Music VisualArts': 0.5714285714285714,
'NutritionEducation': 0.7727272727272727,
'NutritionEducation Other': 1.0,
'NutritionEducation SocialSciences': 1.0,
'NutritionEducation SpecialNeeds': 0.8,
'NutritionEducation TeamSports': 0.0,
'Other': 0.85,
'Other SocialSciences': 1.0,
'Other SpecialNeeds': 0.8367346938775511,
'Other VisualArts': 0.6666666666666666,
'ParentInvolvement': 1.0,
'ParentInvolvement PerformingArts': 1.0,
'ParentInvolvement SpecialNeeds': 0.75,
'ParentInvolvement VisualArts': 0.8,
'PerformingArts': 0.859375,
'PerformingArts SocialSciences': 1.0,
'PerformingArts SpecialNeeds': 0.75,
'PerformingArts TeamSports': 1.0,
'PerformingArts VisualArts': 0.6,
'SocialSciences': 0.9090909090909091,
'SocialSciences SpecialNeeds': 0.6,
'SocialSciences VisualArts': 0.9,
'SpecialNeeds': 0.8112582781456954,
'SpecialNeeds TeamSports': 0.4,
'SpecialNeeds VisualArts': 0.8222222222222222,
'SpecialNeeds Warmth Care_Hunger': 0.75,
```

```
'TeamSports': 0.7945205479452054,
'VisualArts': 0.8110749185667753,
'Warmth Care_Hunger': 0.9015544041450777}
```

In [94]:

```
subcat_neg_test = []
subcat_pos_test = []
for i in project_data_test['clean_subcategories']:
    subcat_neg_test.append(subcat_0_test[i])
    subcat_pos_test.append(subcat_1_test[i])
project_data_test['subcat_0'] = subcat_neg_test
project_data_test['subcat_1'] = subcat_pos_test
```

In [95]:

```
state_0_test
```

Out[95]:

```
{'AK': 0.22916666666666666,
 'AL': 0.12692307692307692,
 'AR': 0.12857142857142856,
 'AZ': 0.17613636363636365,
 'CA': 0.15104166666666666,
 'CO': 0.17714285714285713,
 'CT': 0.14782608695652175,
 'DC': 0.2,
 'DE': 0.10204081632653061,
 'FL': 0.18039624608967675,
 'GA': 0.15645161290322582,
 'HI': 0.09876543209876543,
 'IA': 0.18811881188118812,
 'ID': 0.2018348623853211,
 'IL': 0.1435114503816794,
 'IN': 0.15306122448979592,
 'KS': 0.14705882352941177,
 'KY': 0.13488372093023257,
 'LA': 0.17473118279569894,
 'MA': 0.12146892655367232,
 'MD': 0.11627906976744186,
 'ME': 0.2,
 'MI': 0.1517509727626459,
 'MN': 0.11797752808988764,
 'MO': 0.144772117962465,
 'MS': 0.15028901734104047,
 'MT': 0.2702702702702703,
 'NC': 0.148963730569948,
 'ND': 0.13636363636363635,
 'NE': 0.134615384615384,
 'NH': 0.16326530612244897,
 'NJ': 0.17751479289940827,
 'NM': 0.09523809523809523,
 'NV': 0.14220183486238533,
 'NY': 0.15541740674955595,
 'OH': 0.13695090439276486,
 'OK': 0.1440443213296399,
 'OR': 0.1736842105263158,
 'PA': 0.11740041928721175,
 'RI': 0.15384615384615385,
 'SC': 0.12706270627062707,
 'SD': 0.13636363636363635,
 'TN': 0.15503875968992248,
 'TX': 0.21668264621284755,
 'UT': 0.15636363636363637,
 'VA': 0.15714285714285714,
 'VT': 0.18181818181818182,
 'WA': 0.0972972972972973,
 'WI': 0.160142348754484,
 'WV': 0.14285714285714285,
 'WY': 0.11764705882352941}
```

In [96]:

```
state_1_test
```

Out[96]:

```
{'AK': 0.7708333333333334,
 'AL': 0.8730769230769231,
 'AR': 0.8714285714285714,
 'AZ': 0.8238636363636364,
 'CA': 0.8489583333333334,
 'CO': 0.8228571428571428,
 'CT': 0.8521739130434782,
 'DC': 0.8,
 'DE': 0.8979591836734694,
 'FL': 0.8196037539103233,
 'GA': 0.8435483870967742,
 'HI': 0.9012345679012346,
 'IA': 0.8118811881188119,
 'ID': 0.7981651376146789,
 'IL': 0.8564885496183207,
 'IN': 0.8469387755102041,
 'KS': 0.8529411764705882,
 'KY': 0.8651162790697674,
 'LA': 0.8252688172043011,
 'MA': 0.8785310734463276,
 'MD': 0.8837209302325582,
 'ME': 0.8,
 'MI': 0.8482490272373541,
 'MN': 0.8820224719101124,
 'MO': 0.8552278820375335,
 'MS': 0.8497109826589595,
 'MT': 0.7297297297297297,
 'NC': 0.8510362694300518,
 'ND': 0.8636363636363636,
 'NE': 0.8653846153846154,
 'NH': 0.8367346938775511,
 'NJ': 0.8224852071005917,
 'NM': 0.9047619047619048,
 'NV': 0.8577981651376146,
 'NY': 0.844582593250444,
 'OH': 0.8630490956072352,
 'OK': 0.8559556786703602,
 'OR': 0.8263157894736842,
 'PA': 0.8825995807127882,
 'RI': 0.8461538461538461,
 'SC': 0.8729372937293729,
 'SD': 0.8636363636363636,
 'TN': 0.8449612403100775,
 'TX': 0.7833173537871524,
 'UT': 0.8436363636363636,
 'VA': 0.8428571428571429,
 'VT': 0.8181818181818182,
 'WA': 0.9027027027027027,
 'WI': 0.8398576512455516,
 'WV': 0.8571428571428571,
 'WY': 0.8823529411764706}
```

In [97]:

```
state_neg_test = []
state_pos_test = []
for i in project_data_test['school_state']:
    state_neg_test.append(state_0_test[i])
    state_pos_test.append(state_1_test[i])
project_data_test['state_0'] = state_neg_test
project_data_test['state_1'] = state_pos_test
```

In [98]:

```
prefix_0_test
```

Out[98]:

```
{'Mr': 0.16375,
 'Mrs': 0.15122568765105304,
 'Ms': 0.1531362160314476l
```

```
 'Ms': 0.15313621603144761,
 'Teacher': 0.20555555555555555}
```

In [99]:

```python
prefix_1_test
```

Out[99]:

```
{'Mr': 0.83625,
 'Mrs': 0.848774312348947,
 'Ms': 0.8468637839685523,
 'Teacher': 0.7944444444444444}
```

In [100]:

```python
prefix_neg_test = []
prefix_pos_test = []
for i in project_data_test['teacher_prefix']:
    prefix_neg_test.append(prefix_0_test[i])
    prefix_pos_test.append(prefix_1_test[i])
project_data_test['prefix_0'] = prefix_neg_test
project_data_test['prefix_1'] = prefix_pos_test
```

In [101]:

```python
grad_cat_0_test
```

Out[101]:

```
{'Grades_3_5': 0.1445739257101238,
 'Grades_6_8': 0.1543186180422265,
 'Grades_9_12': 0.16021765417170497,
 'Grades_PreK_2': 0.16076455771225368}
```

In [102]:

```python
grad_cat_1_test
```

Out[102]:

```
{'Grades_3_5': 0.8554260742898762,
 'Grades_6_8': 0.8456813819577735,
 'Grades_9_12': 0.839782345828295,
 'Grades_PreK_2': 0.8392354422877464}
```

In [103]:

```python
grade_neg_test = []
grade_pos_test = []
for i in project_data_test['project_grade_category']:
    grade_neg_test.append(grad_cat_0_test[i])
    grade_pos_test.append(grad_cat_1_test[i])
project_data_test['grade_0'] = grade_neg_test
project_data_test['grade_1'] = grade_pos_test
```

In [104]:

```python
project_data_test.columns
```

Out[104]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'school_state',
       'project_submitted_datetime', 'project_title',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'clean_categories',
       'clean_subcategories', 'project_grade_category', 'teacher_prefix',
       'essay', 'preprocessed_essays', 'essay_word_count',
       'preprocessed_titles', 'title_word_count', 'neg', 'neu', 'pos',
       'compound', 'price', 'quantity', 'cat_0', 'cat_1', 'subcat_0',
       'subcat_1', 'state_0', 'state_1', 'prefix_0', 'prefix_1', 'grade_0'
```

```
       'grade_1'],
      dtype='object')
```

In [105]:

```
project_data_test.head()
```

Out[105]:

	Unnamed: 0	id	teacher_id	school_state	project_submitted_datetime	project_title	project_resource_s
0	11624	p199334	9ca29f5ac93bd54d86a17c068dc06be2	OH	2017-01-07 13:37:32	we want to wobble while we learn!	My students ne option for sea
1	97849	p057423	a0920c721954b6332753616983e4ebdc	CA	2016-10-05 04:00:23	dry, dry, dry!!!	My students ne drying rack to
2	12662	p145706	2abc7240adb4199ebae24b35fb31cb17	NY	2016-11-02 11:43:29	everyone loves laminating!	My students ne laminated! Ma
3	12758	p238190	c04d98e5631410f9dadaa226163a500a	PA	2016-08-01 20:08:12	active bodies promote active minds	My students need Pedal Exerc
4	26057	p027307	3f031704f900c4a6893128abaca2ff98	CA	2016-09-01 05:33:15	a technologically advanced american history ex...	My students nee mini and an Ap

5 rows × 33 columns

In [106]:

```
from sklearn.preprocessing import Normalizer

normalizer = Normalizer()

# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

normalizer.fit(project_data_train["cat_0"].values.reshape(-1,1))  #fit has to be done only on Train
data

cat_0_train_normalized = normalizer.transform(project_data_train["cat_0"].values.reshape(1,-1))
cat_0_test_normalized = normalizer.transform(project_data_test["cat_0"].values.reshape(1,-1))

#reshaping after normalizing
cat_0_train_normalized = cat_0_train_normalized.reshape(-1,1)
cat_0_test_normalized = cat_0_test_normalized.reshape(-1,1)

print("After vectorizations")
print(cat_0_train_normalized.shape, y_train.shape)
print(cat_0_test_normalized.shape, y_test.shape)
```

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

In [107]:

```
cat_0_train_normalized
```

Out[107]:

```
array([[0.00476633],
       [0.0051798 ],
       [0.0051798 ],
       ...,
```

```
        ...,
        [0.00476633],
        [0.0051798 ],
        [0.00563073]])
```

In [108]:

```
from sklearn.preprocessing import Normalizer

normalizer = Normalizer()

# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

normalizer.fit(project_data_train["cat_1"].values.reshape(-1,1))   #fit has to be done only on Trai
n data

cat_1_train_normalized = normalizer.transform(project_data_train["cat_1"].values.reshape(1,-1))
cat_1_test_normalized = normalizer.transform(project_data_test["cat_1"].values.reshape(1,-1))


#reshaping after normalizing
cat_1_train_normalized = cat_1_train_normalized.reshape(-1,1)
cat_1_test_normalized = cat_1_test_normalized.reshape(-1,1)

print("After vectorizations")
print(cat_1_train_normalized.shape, y_train.shape)
print(cat_1_test_normalized.shape, y_test.shape)
```

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

In [109]:

```
from sklearn.preprocessing import Normalizer

normalizer = Normalizer()

# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

normalizer.fit(project_data_train["subcat_0"].values.reshape(-1,1))  #fit has to be done only on Tr
ain data

subcat_0_train_normalized = normalizer.transform(project_data_train["subcat_0"].values.reshape(1,-1
))
subcat_0_test_normalized = normalizer.transform(project_data_test["subcat_0"].values.reshape(1,-1))

#reshaping after normalizing
subcat_0_train_normalized = subcat_0_train_normalized.reshape(-1,1)
subcat_0_test_normalized = subcat_0_test_normalized.reshape(-1,1)



print("After vectorizations")
print(subcat_0_train_normalized.shape, y_train.shape)
print(subcat_0_test_normalized.shape, y_test.shape)
```

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

In [110]:

```
from sklearn.preprocessing import Normalizer

normalizer = Normalizer()

# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

normalizer.fit(project_data_train["subcat_1"].values.reshape(-1,1))  #fit has to be done only on Tr
ain data

subcat_1_train_normalized = normalizer.transform(project_data_train["subcat_1"].values.reshape(1,-1
))
subcat_1_test_normalized = normalizer.transform(project_data_test["subcat_1"].values.reshape(1,-1))


#reshaping after normalizing
subcat_1_train_normalized = subcat_1_train_normalized.reshape(-1,1)
subcat_1_test_normalized = subcat_1_test_normalized.reshape(-1,1)

print("After vectorizations")
print(subcat_1_train_normalized.shape, y_train.shape)
print(subcat_1_test_normalized.shape, y_test.shape)
```

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

In [111]:

```
from sklearn.preprocessing import Normalizer

normalizer = Normalizer()

# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

normalizer.fit(project_data_train["state_0"].values.reshape(-1,1))  #fit has to be done only on
Train data

state_0_train_normalized = normalizer.transform(project_data_train["state_0"].values.reshape(1,-1))
state_0_test_normalized = normalizer.transform(project_data_test["state_0"].values.reshape(1,-1))

#reshaping after normalizing
state_0_train_normalized = state_0_train_normalized.reshape(-1,1)
state_0_test_normalized = state_0_test_normalized.reshape(-1,1)

print("After vectorizations")
print(state_0_train_normalized.shape, y_train.shape)
print(state_0_test_normalized.shape, y_test.shape)
```

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

In [112]:

```
from sklearn.preprocessing import Normalizer

normalizer = Normalizer()

# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
```

```
normalizer.fit(project_data_train["state_1"].values.reshape(-1,1))  #fit has to be done only on 
Train data

state_1_train_normalized = normalizer.transform(project_data_train["state_1"].values.reshape(1,-1))
state_1_test_normalized = normalizer.transform(project_data_test["state_1"].values.reshape(1,-1))


#reshaping after normalizing
state_1_train_normalized = state_1_train_normalized.reshape(-1,1)
state_1_test_normalized = state_1_test_normalized.reshape(-1,1)

print("After vectorizations")
print(state_1_train_normalized.shape, y_train.shape)
print(state_1_test_normalized.shape, y_test.shape)
```

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

In [113]:

```
from sklearn.preprocessing import Normalizer

normalizer = Normalizer()

# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead: 
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using 
# array.reshape(-1, 1) if your data has a single feature 
# array.reshape(1, -1)  if it contains a single sample.

normalizer.fit(project_data_train["prefix_0"].values.reshape(-1,1))  #fit has to be done only on Tr
ain data

prefix_0_train_normalized = normalizer.transform(project_data_train["prefix_0"].values.reshape(1,-1
))
prefix_0_test_normalized = normalizer.transform(project_data_test["prefix_0"].values.reshape(1,-1))

#reshaping after normalizing
prefix_0_train_normalized = prefix_0_train_normalized.reshape(-1,1)
prefix_0_test_normalized = prefix_0_test_normalized.reshape(-1,1)

print("After vectorizations")
print(prefix_0_train_normalized.shape, y_train.shape)
print(prefix_0_test_normalized.shape, y_test.shape)
```

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

In [114]:

```
from sklearn.preprocessing import Normalizer

normalizer = Normalizer()

# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead: 
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using 
# array.reshape(-1, 1) if your data has a single feature 
# array.reshape(1, -1)  if it contains a single sample.

normalizer.fit(project_data_train["prefix_1"].values.reshape(-1,1))  #fit has to be done only on Tr
ain data

prefix_1_train_normalized = normalizer.transform(project_data_train["prefix_1"].values.reshape(1,-1
))
prefix_1_test_normalized = normalizer.transform(project_data_test["prefix_1"].values.reshape(1,-1))


#reshaping after normalizing
```

```
#reshaping after normalizing
prefix_1_train_normalized = prefix_1_train_normalized.reshape(-1,1)
prefix_1_test_normalized = prefix_1_test_normalized.reshape(-1,1)


print("After vectorizations")
print(prefix_1_train_normalized.shape, y_train.shape)
print(prefix_1_test_normalized.shape, y_test.shape)
```

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

In [115]:

```
from sklearn.preprocessing import Normalizer

normalizer = Normalizer()

# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

normalizer.fit(project_data_train["grade_0"].values.reshape(-1,1))  #fit has to be done only on
Train data

grade_0_train_normalized = normalizer.transform(project_data_train["grade_0"].values.reshape(1,-1))
grade_0_test_normalized = normalizer.transform(project_data_test["grade_0"].values.reshape(1,-1))

#reshaping after normalizing
grade_0_train_normalized = grade_0_train_normalized.reshape(-1,1)
grade_0_test_normalized = grade_0_test_normalized.reshape(-1,1)




print("After vectorizations")
print(grade_0_train_normalized.shape, y_train.shape)
print(grade_0_test_normalized.shape, y_test.shape)
```

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

In [116]:

```
from sklearn.preprocessing import Normalizer

normalizer = Normalizer()

# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

normalizer.fit(project_data_train["grade_1"].values.reshape(-1,1))  #fit has to be done only on
Train data

grade_1_train_normalized = normalizer.transform(project_data_train["grade_1"].values.reshape(1,-1))
grade_1_test_normalized = normalizer.transform(project_data_test["grade_1"].values.reshape(1,-1))


#reshaping after normalizing
grade_1_train_normalized = grade_1_train_normalized.reshape(-1,1)
grade_1_test_normalized = grade_1_test_normalized.reshape(-1,1)

print("After vectorizations")
print(grade_1_train_normalized.shape, y_train.shape)
print(grade_1_test_normalized.shape, y_test.shape)
```

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

Assignment 9: RF and GBDT

Response Coding: Example

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

1. **Apply both Random Forrest and GBDT on these feature sets**

   - Set 1: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(BOW) + preprocessed_eassay (BOW)
   - Set 2: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
   - Set 3: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
   - Set 4: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

2. **The hyper paramter tuning (Consider any two hyper parameters preferably n_estimators, max_depth)**

   - Consider the following range for hyperparameters **n_estimators** = [10, 50, 100, 150, 200, 300, 500, 1000], **max_depth** = [2, 3, 4, 5, 6, 7, 8, 9, 10]
   - Find the best hyper parameter which will give the maximum AUC value
   - find the best hyper paramter using k-fold cross validation/simple cross validation data
   - use gridsearch cv or randomsearch cv or you can write your own for loops to do this task

3. **Representation of results**

   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure
     with X-axis as **n_estimators**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

   or

   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure
     seaborn heat maps with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**
   - You can choose either of the plotting techniques: 3d plot or heat map
   - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
   - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points

4. **Conclusion**

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

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.

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

2. Random Forest and GBDT

2.4 Applying Random Forest

Apply Random Forest 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 instrucations

SET 1: categorical(instead of one hot encoding, try response coding(using probability values), numerical features + project_title(BOW) + preprocessed_eassay (BOW)

In [117]:

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

X_train = hstack((cat_0_train_normalized, cat_1_train_normalized, subcat_0_train_normalized,
subcat_1_train_normalized, state_0_train_normalized, state_1_train_normalized,
grade_0_train_normalized, grade_1_train_normalized, prefix_0_train_normalized,
prefix_1_train_normalized, price_normalized_train, quantity_normalized_train,
previously_posted_projects_normalized_train, title_word_count_normalized_train,
essay_word_count_normalized_train, sent_pos_train, sent_neg_train, sent_neu_train, sent_compound_tr
ain, train_title_bow, train_essay_bow)).tocsr()
X_test =  hstack((cat_0_test_normalized, cat_1_test_normalized, subcat_0_test_normalized, subcat_1_
test_normalized, state_0_test_normalized, state_1_test_normalized, grade_0_test_normalized,
grade_1_test_normalized, prefix_0_test_normalized, prefix_1_test_normalized, price_normalized_test
, quantity_normalized_test, previously_posted_projects_normalized_test,
title_word_count_normalized_test, essay_word_count_normalized_test, sent_pos_test, sent_neg_test,
sent_neu_test, sent_compound_test, test_title_bow, test_essay_bow)).tocsr()
```

In [118]:

```
print(X_train.shape)
print(X_test.shape)
```

```
(33500, 11886)
(16500, 11886)
```

In [119]:

```
# https://medium.com/@erikgreenj/k-neighbors-classifier-with-gridsearchcv-basics-3c445ddeb657

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()

grid_params = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth':[2, 3, 4, 5, 6,
7, 8, 9, 10]}

gs = GridSearchCV(rf, grid_params, cv=3, scoring='roc_auc',n_jobs=-1)
gs_results = gs.fit(X_train, y_train)
print(gs_results.best_score_)
print(gs_results.best_estimator_)
print(gs_results.best_params_)
```

```
0.6945542251837132
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=10, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=1000, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

```
            warm_start=False,
{'max_depth': 10, 'n_estimators': 1000}
```

In [120]:

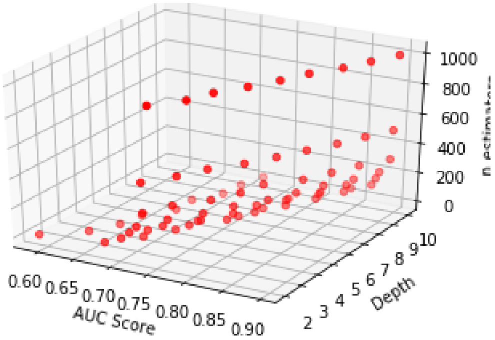
```
#Output of GridSearchCV
print('Best score: ',gs_results.best_score_)
print('k value with best score: ',gs_results.best_params_)
print('='*75)
print('Train AUC scores')
print(gs.cv_results_['mean_train_score'])
print('CV AUC scores')
print(gs.cv_results_['mean_test_score'])
```

```
Best score:  0.6945542251837132
k value with best score:  {'max_depth': 10, 'n_estimators': 1000}
===========================================================================
Train AUC scores
[0.58853489 0.67944103 0.70224551 0.71373403 0.72324539 0.73206266
 0.73070567 0.74079754 0.63323161 0.7087846  0.71508764 0.73566086
 0.74212832 0.748455   0.75500214 0.76888324 0.65262022 0.72951518
 0.74816536 0.76415537 0.76730534 0.77773381 0.77336134 0.78126645
 0.65941007 0.73720188 0.77712411 0.78575433 0.7862262  0.79528025
 0.79997579 0.80404327 0.68441132 0.77095423 0.79307966 0.79552511
 0.81051242 0.80432755 0.82092999 0.82403399 0.68381901 0.78274149
 0.81342954 0.81690495 0.827263   0.83620188 0.83988294 0.84047833
 0.70530294 0.79296768 0.83342114 0.83934473 0.84335118 0.85360487
 0.85949363 0.86371645 0.70874917 0.81709163 0.85098553 0.85406452
 0.86339658 0.8701462  0.87665151 0.8797016  0.71852951 0.84103172
 0.8685624  0.87348981 0.8784004  0.89176727 0.8941225  0.89724093]
CV AUC scores
[0.56819918 0.653053   0.65589413 0.66260294 0.67113425 0.68002796
 0.67564136 0.6802075  0.61200882 0.65510707 0.65467079 0.66978863
 0.67234144 0.67770336 0.68287778 0.68968944 0.61772203 0.66286716
 0.66828335 0.67856104 0.67933769 0.68770159 0.68291505 0.68548756
 0.61995752 0.65581487 0.68236871 0.67957645 0.68107023 0.68771452
 0.68707406 0.68842762 0.63073385 0.66688596 0.67188535 0.67668718
 0.67895048 0.68435864 0.68907198 0.68940834 0.63183601 0.66294838
 0.67601301 0.6792186  0.68525426 0.68806438 0.69207524 0.69179155
 0.63309311 0.65540099 0.68479242 0.68145273 0.68609788 0.69088083
 0.68970752 0.69142257 0.63981271 0.66286219 0.67761845 0.68164653
 0.6870535  0.69145764 0.69051283 0.69338098 0.63089957 0.67357884
 0.67866074 0.68920971 0.68267953 0.6924755  0.6930018  0.69455423]
```

In [121]:

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt



fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

g1 = list(gs.cv_results_['mean_train_score'])      #Train AUC Score
g2 = [2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,4,4,4,4,4,4,4,4,5,5,5,5,5,5,5,5,6,6,6,6,6,6,6,6,7,7,7,7,7,7,7,7,8
,8,8,8,8,8,8,8,9,9,9,9,9,9,9,9,10,10,10,10,10,10,10,10] #Depth
g3 = [10, 50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150,
200, 300, 500, 1000,10, 50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150, 200, 300, 500, 1000,10,
50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150, 200, 300,
500, 1000,10, 50, 100, 150, 200, 300, 500, 1000] #n_estimators



ax.scatter(g1, g2, g3, c='r', marker='o')

ax.set_xlabel('AUC Score')
ax.set_ylabel('Depth')
ax.set_zlabel('n_estimators')

plt.title('3D Scatter plot on Train AUC scores')
plt.show()
```

3D Scatter plot on Train AUC scores



In [122]:

```
gs.cv_results_
```

Out[122]:

```
{'mean_fit_time': array([ 0.17462111,  0.40856457,  0.58339262,  0.85232123,  1.04593237,
         1.47726425,  2.33632247,  4.68881194,  0.20552675,  0.44023967,
         0.82226435,  0.97478596,  1.25126537,  1.82502675,  2.84294748,
         5.66242687,  0.17601903,  0.46765423,  0.78500279,  1.15159607,
         1.45887693,  2.26988689,  3.70074503,  7.23951705,  0.17746512,
         0.5901792 ,  0.89494824,  1.34517741,  1.98432144,  2.47935406,
         4.24433303,  7.59056473,  0.18518265,  0.48118575,  0.96234846,
         1.49781354,  2.13988264,  2.65318576,  4.59198912,  8.92926455,
         0.20373297,  0.56341918,  1.08323995,  1.77550181,  2.33535647,
         3.30094258,  5.21841304, 10.07233485,  0.21428148,  0.62889576,
         1.15403533,  2.00249537,  2.70537345,  3.6570553 ,  5.83130646,
        11.43976235,  0.23694142,  0.68791358,  1.28471979,  2.16474875,
         3.04585441,  3.88365841,  6.57448006, 12.7804269 ,  0.25987275,
         0.78368346,  1.45728461,  2.29497862,  3.47994582,  4.58132084,
         7.11564215, 13.28637314]),
 'std_fit_time': array([0.01206881, 0.00243625, 0.00994902, 0.02818356, 0.01239522,
        0.00964171, 0.0268255 , 0.1293312 , 0.01212754, 0.01954155,
        0.10016718, 0.00459716, 0.01818824, 0.06806194, 0.07097189,
        0.15066619, 0.00230218, 0.02006647, 0.00249959, 0.0362854 ,
        0.00700026, 0.05573816, 0.05352439, 0.21271684, 0.00539201,
        0.09331323, 0.05436418, 0.03676741, 0.13766126, 0.06026132,
        0.04687898, 0.07339038, 0.00236114, 0.01738021, 0.07477928,
        0.0298644 , 0.1266079 , 0.02130385, 0.17483076, 0.41856709,
        0.00587905, 0.00675271, 0.02672791, 0.11105147, 0.175308  ,
        0.0903708 , 0.24487577, 0.10669124, 0.00863109, 0.00402015,
        0.0324066 , 0.09863488, 0.10954041, 0.08214032, 0.06453391,
        0.11475233, 0.01711116, 0.04618761, 0.08338881, 0.121734  ,
        0.18199975, 0.11892576, 0.15976988, 0.11900439, 0.00431042,
        0.05006792, 0.08093911, 0.03645672, 0.12624111, 0.1389107 ,
        0.07093896, 0.10056761]),
 'mean_score_time': array([0.09183033, 0.23396794, 0.49359441, 0.67618203, 0.96135314,
        1.18791866, 2.00033164, 4.85687296, 0.09213495, 0.25951457,
        0.49911086, 0.74318647, 1.04970169, 1.28323523, 1.89941168,
        4.33605528, 0.07349984, 0.25159947, 0.48254212, 0.69990166,
        0.92761819, 1.31597034, 2.13084769, 4.68985828, 0.07475654,
        0.22760312, 0.4912858 , 0.70620346, 1.02825634, 1.41761525,
        1.75203983, 4.19320019, 0.07297802, 0.21110948, 0.49284252,
        0.71976407, 0.98944688, 1.12918862, 1.84745073, 4.36465414,
        0.07225617, 0.20365771, 0.42762502, 0.72361978, 1.02107255,
        1.08685795, 1.77962629, 4.18576535, 0.0698053 , 0.20343788,
        0.38542064, 0.71581491, 0.97435323, 1.04736773, 1.82623927,
        4.03186719, 0.07819915, 0.21673592, 0.40594403, 0.72763928,
        0.95462775, 1.03713536, 1.76161647, 4.13079794, 0.07530514,
        0.21146266, 0.39204804, 0.76312304, 1.01780144, 1.10444506,
        1.7733314 , 2.94136747]),
 'std_score_time': array([0.00874968, 0.00823434, 0.01112669, 0.00476186, 0.00863113,
        0.01664371, 0.02457843, 0.25800192, 0.00113668, 0.01126643,
        0.01370748, 0.01494573, 0.06135446, 0.01575221, 0.07884723,
        0.07230305, 0.00161622, 0.00802484, 0.01689091, 0.0433611 ,
        0.06823774, 0.14028226, 0.0547136 , 0.01211235, 0.00195827,
        0.01316965, 0.02772838, 0.0378552 , 0.06834951, 0.09817869,
        0.03171482, 0.00873868, 0.0005124 , 0.00950221, 0.06917165,
        0.02409291, 0.0300763 , 0.06446047, 0.12981379, 0.29603213,
        0.00122575, 0.01723534, 0.04061798, 0.01593866, 0.08733722,
        0.05268183, 0.1369887 , 0.07527657, 0.00359333, 0.00674985,
        0.0318702 , 0.0746036 , 0.11343016, 0.0932718 , 0.0721402 ,
        0.27897001, 0.00131226, 0.00959273, 0.03919373, 0.0748077
```

```
       0.27897001, 0.00131226, 0.00939275, 0.03919373, 0.0748077 ,
       0.08053157, 0.03254355, 0.01934469, 0.06322226, 0.00436664,
       0.00683993, 0.01447174, 0.03529463, 0.07036967, 0.03613392,
       0.00458123, 0.06438854]),
 'param_max_depth': masked_array(data=[2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4,
                    4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6,
                    6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7, 7, 8, 8, 8, 8, 8, 8,
                    8, 8, 9, 9, 9, 9, 9, 9, 9, 9, 10, 10, 10, 10, 10, 10,
                    10, 10],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_n_estimators': masked_array(data=[10, 50, 100, 150, 200, 300, 500, 1000, 10, 50, 100,
                    150, 200, 300, 500, 1000, 10, 50, 100, 150, 200, 300,
                    500, 1000, 10, 50, 100, 150, 200, 300, 500, 1000, 10,
                    50, 100, 150, 200, 300, 500, 1000, 10, 50, 100, 150,
                    200, 300, 500, 1000, 10, 50, 100, 150, 200, 300, 500,
                    1000, 10, 50, 100, 150, 200, 300, 500, 1000, 10, 50,
                    100, 150, 200, 300, 500, 1000],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 2, 'n_estimators': 10},
  {'max_depth': 2, 'n_estimators': 50},
  {'max_depth': 2, 'n_estimators': 100},
  {'max_depth': 2, 'n_estimators': 150},
  {'max_depth': 2, 'n_estimators': 200},
  {'max_depth': 2, 'n_estimators': 300},
  {'max_depth': 2, 'n_estimators': 500},
  {'max_depth': 2, 'n_estimators': 1000},
  {'max_depth': 3, 'n_estimators': 10},
  {'max_depth': 3, 'n_estimators': 50},
  {'max_depth': 3, 'n_estimators': 100},
  {'max_depth': 3, 'n_estimators': 150},
  {'max_depth': 3, 'n_estimators': 200},
  {'max_depth': 3, 'n_estimators': 300},
  {'max_depth': 3, 'n_estimators': 500},
  {'max_depth': 3, 'n_estimators': 1000},
  {'max_depth': 4, 'n_estimators': 10},
  {'max_depth': 4, 'n_estimators': 50},
  {'max_depth': 4, 'n_estimators': 100},
  {'max_depth': 4, 'n_estimators': 150},
  {'max_depth': 4, 'n_estimators': 200},
  {'max_depth': 4, 'n_estimators': 300},
  {'max_depth': 4, 'n_estimators': 500},
  {'max_depth': 4, 'n_estimators': 1000},
  {'max_depth': 5, 'n_estimators': 10},
  {'max_depth': 5, 'n_estimators': 50},
  {'max_depth': 5, 'n_estimators': 100},
  {'max_depth': 5, 'n_estimators': 150},
  {'max_depth': 5, 'n_estimators': 200},
  {'max_depth': 5, 'n_estimators': 300},
  {'max_depth': 5, 'n_estimators': 500},
  {'max_depth': 5, 'n_estimators': 1000},
  {'max_depth': 6, 'n_estimators': 10},
  {'max_depth': 6, 'n_estimators': 50},
  {'max_depth': 6, 'n_estimators': 100},
  {'max_depth': 6, 'n_estimators': 150},
  {'max_depth': 6, 'n_estimators': 200},
  {'max_depth': 6, 'n_estimators': 300},
  {'max_depth': 6, 'n_estimators': 500},
  {'max_depth': 6, 'n_estimators': 1000}
```

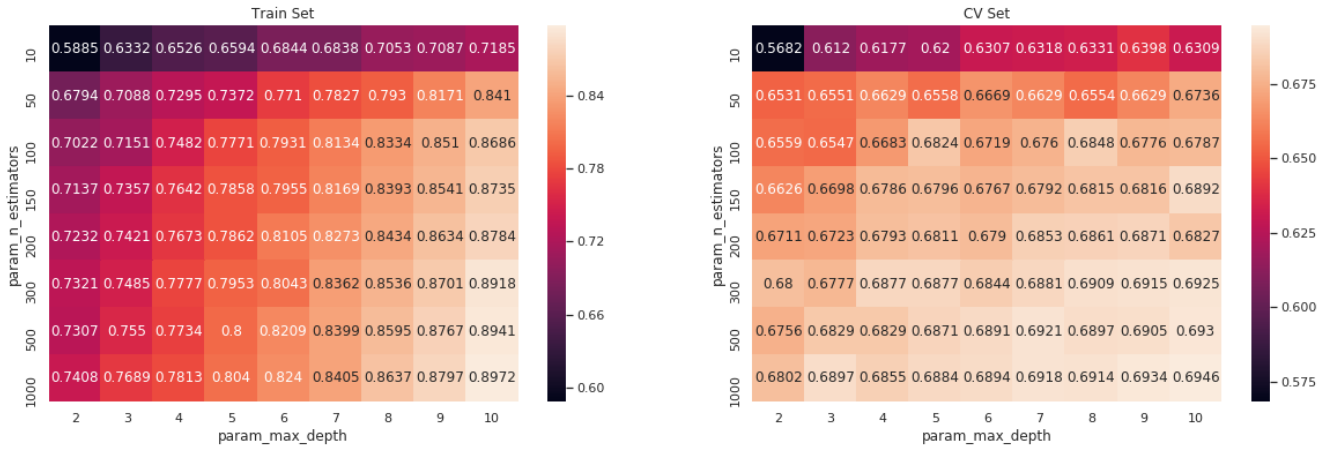
```
 {'max_depth': 6, 'n_estimators': 1000},
 {'max_depth': 7, 'n_estimators': 10},
 {'max_depth': 7, 'n_estimators': 50},
 {'max_depth': 7, 'n_estimators': 100},
 {'max_depth': 7, 'n_estimators': 150},
 {'max_depth': 7, 'n_estimators': 200},
 {'max_depth': 7, 'n_estimators': 300},
 {'max_depth': 7, 'n_estimators': 500},
 {'max_depth': 7, 'n_estimators': 1000},
 {'max_depth': 8, 'n_estimators': 10},
 {'max_depth': 8, 'n_estimators': 50},
 {'max_depth': 8, 'n_estimators': 100},
 {'max_depth': 8, 'n_estimators': 150},
 {'max_depth': 8, 'n_estimators': 200},
 {'max_depth': 8, 'n_estimators': 300},
 {'max_depth': 8, 'n_estimators': 500},
 {'max_depth': 8, 'n_estimators': 1000},
 {'max_depth': 9, 'n_estimators': 10},
 {'max_depth': 9, 'n_estimators': 50},
 {'max_depth': 9, 'n_estimators': 100},
 {'max_depth': 9, 'n_estimators': 150},
 {'max_depth': 9, 'n_estimators': 200},
 {'max_depth': 9, 'n_estimators': 300},
 {'max_depth': 9, 'n_estimators': 500},
 {'max_depth': 9, 'n_estimators': 1000},
 {'max_depth': 10, 'n_estimators': 10},
 {'max_depth': 10, 'n_estimators': 50},
 {'max_depth': 10, 'n_estimators': 100},
 {'max_depth': 10, 'n_estimators': 150},
 {'max_depth': 10, 'n_estimators': 200},
 {'max_depth': 10, 'n_estimators': 300},
 {'max_depth': 10, 'n_estimators': 500},
 {'max_depth': 10, 'n_estimators': 1000}],
'split0_test_score': array([0.57963044, 0.64465768, 0.65537624, 0.66113272, 0.67214712,
       0.67188759, 0.66634526, 0.67097234, 0.57901819, 0.66243922,
       0.64384702, 0.66604118, 0.66240714, 0.6720282 , 0.66749773,
       0.67624655, 0.59280549, 0.64623333, 0.65886106, 0.66173347,
       0.67475079, 0.67577126, 0.67027765, 0.67446908, 0.60449854,
       0.6457647 , 0.67812991, 0.66699791, 0.6684064 , 0.67411037,
       0.67307454, 0.6792232 , 0.60901528, 0.65151   , 0.65979309,
       0.6710526 , 0.66712715, 0.66901241, 0.68396102, 0.67621509,
       0.62825074, 0.64966121, 0.65315531, 0.672305  , 0.67523985,
       0.67697197, 0.6831138 , 0.67764847, 0.61631284, 0.64755895,
       0.67148936, 0.66947124, 0.6696148 , 0.67611006, 0.67733437,
       0.6788096 , 0.62050437, 0.66451358, 0.67258149, 0.6598664 ,
       0.6672586 , 0.68340467, 0.6751693 , 0.68055296, 0.625094  ,
       0.65271646, 0.671191  , 0.6786194 , 0.6671773 , 0.68113458,
       0.6856736 , 0.6822447 ]),
'split1_test_score': array([0.55842486, 0.65225109, 0.64823075, 0.65170189, 0.6559097 ,
       0.66630986, 0.6763694 , 0.67747332, 0.63856784, 0.63503109,
       0.6489742 , 0.66015106, 0.67570575, 0.66818578, 0.6783761 ,
       0.68923493, 0.61717859, 0.67607675, 0.6742786 , 0.68048862,
       0.66989055, 0.68643423, 0.67913771, 0.68214158, 0.63156661,
       0.66043207, 0.6792283 , 0.67527203, 0.68172356, 0.67876154,
       0.68424833, 0.68087093, 0.64664711, 0.66504532, 0.66737918,
       0.67353834, 0.67332134, 0.68255302, 0.67478933, 0.68436276,
       0.62516937, 0.6770215 , 0.67984537, 0.67228017, 0.68129774,
       0.68329921, 0.68663414, 0.68669216, 0.63276941, 0.6493355 ,
       0.68465688, 0.68194314, 0.68193472, 0.68891739, 0.68610833,
       0.68809389, 0.64643822, 0.66371719, 0.6681623 , 0.69170438,
       0.68382416, 0.68769812, 0.69107649, 0.68922878, 0.6332712 ,
       0.67826689, 0.67779719, 0.6835309 , 0.68250269, 0.68776781,
       0.6869231 , 0.69125318]),
'split2_test_score': array([0.5665421 , 0.66225107, 0.66407615, 0.67497532, 0.6853472 ,
       0.70188838, 0.68421021, 0.69217792, 0.61844101, 0.66785203,
       0.67119261, 0.68317485, 0.67891203, 0.69289746, 0.7027613 ,
       0.70358808, 0.64318427, 0.6662917 , 0.6717107 , 0.69346237,
       0.693373  , 0.70090047, 0.69933125, 0.69985331, 0.62380775,
       0.66124833, 0.6897486 , 0.69646092, 0.69308181, 0.71027368,
       0.70390082, 0.70519023, 0.63653969, 0.6841041 , 0.68848527,
       0.68547138, 0.6964045 , 0.70151202, 0.70846732, 0.70764882,
       0.64208884, 0.66216237, 0.69504005, 0.69307188, 0.69922644,
       0.70392339, 0.70647908, 0.71103574, 0.6501986 , 0.66930976,
       0.69823222, 0.69294484, 0.70674598, 0.70761653, 0.70568129,
       0.70736565, 0.65249667, 0.66035558, 0.69211286, 0.69336986,
       0.7100798 , 0.70327119, 0.70529402, 0.71036272, 0.63433383,
       0.68975463, 0.68699479, 0.70548028, 0.69836   , 0.70852555,
       0.7064099 , 0.71016619]),
```

```
       0.7064099 , 0.71016619]),
 'mean_test_score': array([0.56819918, 0.653053  , 0.65589413, 0.66260294, 0.67113425,
       0.68002796, 0.67564136, 0.6802075 , 0.61200882, 0.65510707,
       0.65467079, 0.66978863, 0.67234144, 0.67770336, 0.68287778,
       0.68968944, 0.61772203, 0.66286716, 0.66828335, 0.67856104,
       0.67933769, 0.68770159, 0.68291505, 0.68548756, 0.61995752,
       0.65581487, 0.68236871, 0.67957645, 0.68107023, 0.68771452,
       0.68707406, 0.68842762, 0.63073385, 0.66688596, 0.67188535,
       0.67668718, 0.67895048, 0.68435864, 0.68907198, 0.68940834,
       0.63183601, 0.66294838, 0.67601301, 0.6792186 , 0.68525426,
       0.68806438, 0.69207524, 0.69179155, 0.63309311, 0.65540099,
       0.68479242, 0.68145273, 0.68609788, 0.69088083, 0.68970752,
       0.69142257, 0.63981271, 0.66286219, 0.67761845, 0.68164653,
       0.6870535 , 0.69145764, 0.69051283, 0.69338098, 0.63089957,
       0.67357884, 0.67866074, 0.68920971, 0.68267953, 0.6924755 ,
       0.6930018 , 0.69455423]),
 'std_test_score': array([0.0087362 , 0.00720477, 0.00647917, 0.00955797, 0.01203904,
       0.01562379, 0.00731143, 0.00887035, 0.02473316, 0.01436713,
       0.01186822, 0.00976575, 0.00714568, 0.01085729, 0.01474392,
       0.01116668, 0.02057049, 0.01242199, 0.00674469, 0.01302469,
       0.01012043, 0.01029795, 0.01215803, 0.01062966, 0.01138106,
       0.00711451, 0.00523737, 0.01240727, 0.0100842 , 0.01606364,
       0.01274232, 0.0118715 , 0.01590236, 0.01336989, 0.01213911,
       0.00629345, 0.01259772, 0.0133291 , 0.01421595, 0.01331942,
       0.00735786, 0.01118378, 0.01731265, 0.00979531, 0.01018424,
       0.01150719, 0.0102855 , 0.0140991 , 0.01383559, 0.00986126,
       0.01091807, 0.00958926, 0.01544184, 0.01293708, 0.01184905,
       0.01189312, 0.01387558, 0.00180194, 0.01040623, 0.01541622,
       0.01763006, 0.00853499, 0.01230473, 0.01251887, 0.00412811,
       0.01547978, 0.00648066, 0.01167795, 0.0127308 , 0.01166722,
       0.00949425, 0.01163536]),
 'rank_test_score': array([72, 63, 58, 57, 50, 35, 46, 34, 71, 61, 62, 51, 48, 42, 28, 12, 70,
       55, 52, 41, 37, 19, 27, 23, 69, 59, 30, 36, 33, 18, 20, 16, 68, 53,
       49, 44, 39, 26, 15, 13, 66, 54, 45, 38, 24, 17,  5,  6, 65, 60, 25,
       32, 22,  9, 11,  8, 64, 56, 43, 31, 21,  7, 10,  2, 67, 47, 40, 14,
       29,  4,  3,  1], dtype=int32),
 'split0_train_score': array([0.59930084, 0.67985808, 0.71208238, 0.72462773, 0.73800099,
       0.74081814, 0.73028042, 0.74503819, 0.61312772, 0.71711603,
       0.72632847, 0.74257346, 0.74616025, 0.75605411, 0.75147252,
       0.76694967, 0.64721437, 0.73053061, 0.75489499, 0.76331828,
       0.78127989, 0.77754762, 0.77447677, 0.78229319, 0.64136204,
       0.74160141, 0.78959678, 0.7727031 , 0.78536373, 0.79936122,
       0.79967316, 0.80757585, 0.68234389, 0.76629177, 0.78589409,
       0.80234879, 0.81549434, 0.80449142, 0.82792329, 0.82252245,
       0.70238265, 0.78341721, 0.81225155, 0.81492574, 0.8268336 ,
       0.8422709 , 0.84200249, 0.83933598, 0.70705808, 0.79442819,
       0.82974043, 0.84237997, 0.83546261, 0.85138161, 0.86127871,
       0.86869546, 0.70500464, 0.8296622 , 0.85966681, 0.84305414,
       0.85602001, 0.87453216, 0.87714495, 0.8783951 , 0.70875741,
       0.83897865, 0.87036532, 0.87240639, 0.86920695, 0.89126465,
       0.90254691, 0.89632726]),
 'split1_train_score': array([0.5888566 , 0.68017613, 0.70130002, 0.70848476, 0.70902364,
       0.72296096, 0.73787719, 0.74476598, 0.66104303, 0.70769191,
       0.70105346, 0.73120057, 0.74717214, 0.74054856, 0.7543232 ,
       0.77127184, 0.65246019, 0.73777402, 0.75175184, 0.76898203,
       0.76181754, 0.78275522, 0.77675809, 0.78099916, 0.66676477,
       0.73938876, 0.77747431, 0.78788675, 0.78664755, 0.78166351,
       0.80117219, 0.80077607, 0.70119865, 0.77493398, 0.78697561,
       0.79682219, 0.79865774, 0.80401086, 0.80886737, 0.82423388,
       0.67888865, 0.79181724, 0.81435625, 0.81706421, 0.8222919 ,
       0.83591124, 0.8391159 , 0.83693827, 0.70059065, 0.78899758,
       0.84029761, 0.83679093, 0.8396218 , 0.85352775, 0.85559173,
       0.86317074, 0.71004776, 0.81384864, 0.83931891, 0.86657703,
       0.86437042, 0.86596395, 0.87876418, 0.88071939, 0.72996153,
       0.83983452, 0.8725186 , 0.87619694, 0.88628416, 0.89159907,
       0.89334367, 0.90081799]),
 'split2_train_score': array([0.57744725, 0.67828887, 0.69335412, 0.70808959, 0.72271153,
       0.73240888, 0.7239594 , 0.73258845, 0.62552409, 0.70154586,
       0.717881  , 0.73320855, 0.73305256, 0.74876233, 0.7592107 ,
       0.76842822, 0.65818609, 0.7202409 , 0.73784926, 0.76016581,
       0.75881859, 0.77289859, 0.76884914, 0.78050701, 0.67010339,
       0.73061546, 0.76430124, 0.79667315, 0.78666734, 0.80481602,
       0.79908204, 0.80377791, 0.66969143, 0.77163696, 0.80636927,
       0.78740436, 0.81738517, 0.80448037, 0.82599931, 0.82534565,
       0.67018574, 0.77299001, 0.81368082, 0.81872491, 0.83266349,
       0.83042351, 0.83853042, 0.84516076, 0.70826009, 0.79547728,
       0.83022538, 0.83886328, 0.85496913, 0.85590525, 0.86161046,
       0.85928314, 0.71119513, 0.80776405, 0.85307086, 0.85256339,
```

```
        0.85928314, 0.71119513, 0.80776405, 0.85397086, 0.85256239,
        0.86979932, 0.86994248, 0.87404542, 0.87999032, 0.71686959,
        0.84428198, 0.86280328, 0.87186608, 0.87971009, 0.89243809,
        0.88647692, 0.89457754]),
 'mean_train_score': array([0.58853489, 0.67944103, 0.70224551, 0.71373403, 0.72324539,
        0.73206266, 0.73070567, 0.74079754, 0.63323161, 0.7087846 ,
        0.71508764, 0.73566086, 0.74212832, 0.748455  , 0.75500214,
        0.76888324, 0.65262022, 0.72951518, 0.74816536, 0.76415537,
        0.76730534, 0.77773381, 0.77336134, 0.78126645, 0.65941007,
        0.73720188, 0.77712411, 0.78575433, 0.7862262 , 0.79528025,
        0.79997579, 0.80404327, 0.68441132, 0.77095423, 0.79307966,
        0.79552511, 0.81051242, 0.80432755, 0.82092999, 0.82403399,
        0.68381901, 0.78274149, 0.81342954, 0.81690495, 0.827263  ,
        0.83620188, 0.83988294, 0.84047833, 0.70530294, 0.79296768,
        0.83342114, 0.83934473, 0.84335118, 0.85360487, 0.85949363,
        0.86371645, 0.70874917, 0.81709163, 0.85098553, 0.85406452,
        0.86339658, 0.8701462 , 0.87665151, 0.8797016 , 0.71852951,
        0.84103172, 0.8685624 , 0.87348981, 0.8784004 , 0.89176727,
        0.8941225 , 0.89724093]),
 'std_train_score': array([0.00892459, 0.00082498, 0.00767496, 0.0077047 , 0.01183598,
        0.00729427, 0.00568986, 0.00580577, 0.02030638, 0.00640328,
        0.01050583, 0.00495621, 0.00643081, 0.00633384, 0.00319537,
        0.00179361, 0.00448062, 0.00719379, 0.00740659, 0.00364755,
        0.00995706, 0.0040261 , 0.00332375, 0.0007533 , 0.01283446,
        0.00474409, 0.01032983, 0.00990122, 0.00060992, 0.00988266,
        0.00087973, 0.00278233, 0.01294558, 0.00356104, 0.00940754,
        0.00616959, 0.00841799, 0.00022398, 0.00856565, 0.0011612 ,
        0.01359881, 0.00770102, 0.00087742, 0.00155509, 0.00424506,
        0.00484104, 0.00151769, 0.00345264, 0.00336803, 0.00283977,
        0.00486643, 0.00230697, 0.00838877, 0.00184757, 0.00276239,
        0.00386189, 0.0026889 , 0.0092293 , 0.00857101, 0.00966174,
        0.00566737, 0.00350092, 0.00195777, 0.0009706 , 0.00873576,
        0.00232469, 0.00416612, 0.0019269 , 0.00703298, 0.0004936 ,
        0.00658362, 0.00262829])}
```

In [123]:

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



In [124]:

```python
gs_results.best_params_
```

Out[124]:

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

In [125]:

In [125]:

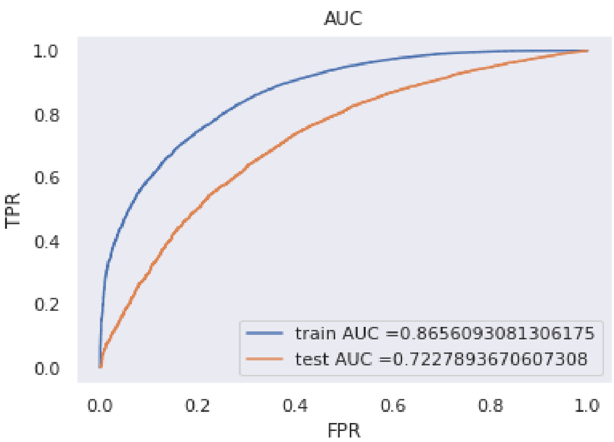
```
max_d = gs_results.best_params_['max_depth']
n_est = gs_results.best_params_['n_estimators']
```

In [126]:

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

In [127]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
model = RandomForestClassifier(max_depth = 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 [128]:

```
# 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", np.round(t,3))
    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 [129]:

```
#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) 0.5990830500929055 for threshold 0.838
Train confusion matrix
[[ 3862  1306]
 [ 5619 22713]]
```

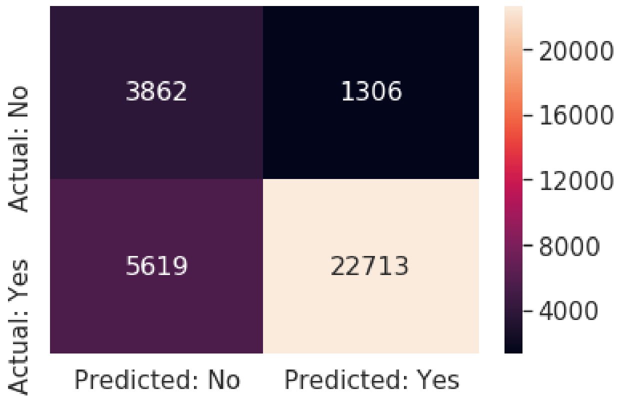
In [130]:

```
#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,
predict_with_best_t(y_train_pred, best_t)), ['Actual: No','Actual: Yes'],['Predicted:
No','Predicted: Yes'])
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[130]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f90a2c55d30>
```



In [131]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
Test confusion matrix
[[ 1255  1291]
 [ 2586 11368]]
```

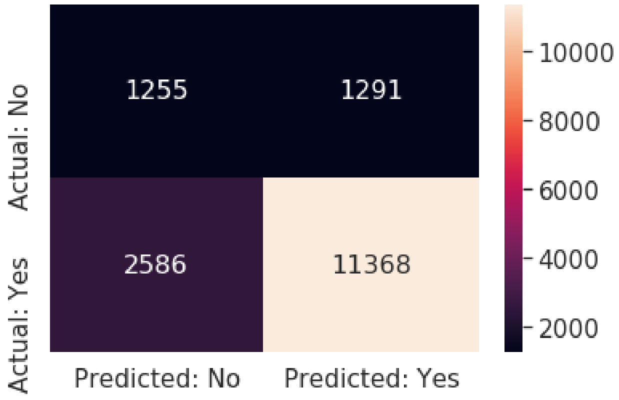
In [132]:

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

confusion_matrix_df_test = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, b
est_t)), ['Actual: No','Actual: Yes'],['Predicted: No','Predicted: Yes'])
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[132]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f90a2962198>
```



SET 2 categorical (with response coding), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)

In [133]:

```
# Please write all the code with proper documentation
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_train = hstack((cat_0_train_normalized, cat_1_train_normalized, subcat_0_train_normalized,
subcat_1_train_normalized, state_0_train_normalized, state_1_train_normalized,
grade_0_train_normalized, grade_1_train_normalized, prefix_0_train_normalized,
prefix_1_train_normalized, price_normalized_train, quantity_normalized_train,
previously_posted_projects_normalized_train, title_word_count_normalized_train,
essay_word_count_normalized_train, sent_pos_train, sent_neg_train, sent_neu_train, sent_compound_tr
ain, train_title_tfidf, train_essay_tfidf)).tocsr()
X_test =  hstack((cat_0_test_normalized, cat_1_test_normalized, subcat_0_test_normalized, subcat_1_
test_normalized, state_0_test_normalized, state_1_test_normalized, grade_0_test_normalized,
grade_1_test_normalized, prefix_0_test_normalized, prefix_1_test_normalized, price_normalized_test
, quantity_normalized_test, previously_posted_projects_normalized_test,
title_word_count_normalized_test, essay_word_count_normalized_test, sent_pos_test, sent_neg_test,
sent_neu_test, sent_compound_test, test_title_tfidf, test_essay_tfidf)).tocsr()
```

In [134]:

```
print(X_train.shape)
print(X_test.shape)
```

```
(33500, 11886)
(16500, 11886)
```

In [135]:

```
# https://medium.com/@erikgreenj/k-neighbors-classifier-with-gridsearchcv-basics-3c445ddeb657

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()

grid_params = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth':[2, 3, 4, 5, 6,
7, 8, 9, 10]}

gs = GridSearchCV(rf, grid_params, cv=3, scoring='roc_auc',n_jobs=-1)
gs_results = gs.fit(X_train, y_train)
print(gs_results.best_score_)
print(gs_results.best_estimator_)
print(gs_results.best_params_)
```

```
0.6993463225163405
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=10, max_features='auto', max_leaf_nodes=None,
```

```
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=1000, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
{'max_depth': 10, 'n_estimators': 1000}
```

In [136]:

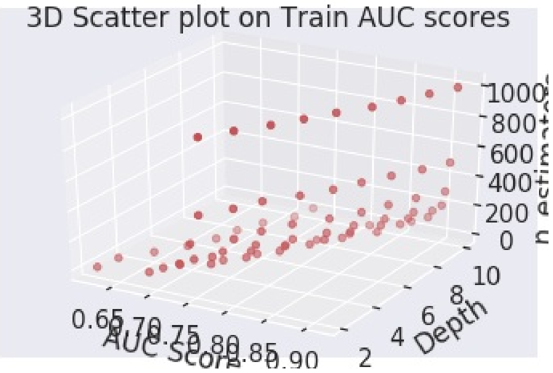
```
#Output of GridSearchCV
print('Best score: ',gs_results.best_score_)
print('k value with best score: ',gs_results.best_params_)
print('='*75)
print('Train AUC scores')
print(gs.cv_results_['mean_train_score'])
print('CV AUC scores')
print(gs.cv_results_['mean_test_score'])
```

```
Best score:  0.6993463225163405
k value with best score:  {'max_depth': 10, 'n_estimators': 1000}
===========================================================================
Train AUC scores
[0.62054276 0.68964259 0.70806712 0.72935894 0.74839948 0.74419981
 0.75545765 0.75703022 0.62520024 0.70694993 0.74780455 0.7495672
 0.76293286 0.77757226 0.77710342 0.77920006 0.64953316 0.74107092
 0.77119772 0.77762997 0.7835705  0.79109179 0.79230465 0.80304778
 0.67194971 0.76381961 0.79303772 0.80045827 0.80964932 0.80910508
 0.81876065 0.82278297 0.68764153 0.77839127 0.81097428 0.8297809
 0.83270185 0.83604444 0.83987844 0.84239281 0.69416709 0.79966976
 0.83246064 0.84297722 0.84744121 0.85674806 0.85587531 0.86613062
 0.70714682 0.82199335 0.85762577 0.86215473 0.8671582  0.8710567
 0.87370296 0.88239591 0.72852792 0.82862873 0.868257   0.87826779
 0.88476812 0.88723934 0.89337032 0.89804785 0.73319916 0.86353169
 0.88502218 0.88996647 0.9016299  0.90550456 0.90964464 0.91419483]
CV AUC scores
[0.59709032 0.64897489 0.65283018 0.66192487 0.67722283 0.67645755
 0.68212417 0.68350964 0.59958842 0.64661354 0.66886342 0.66684349
 0.67429443 0.68390446 0.68378489 0.68841621 0.61867954 0.66073068
 0.67311223 0.67679362 0.67868323 0.6820571  0.68078582 0.69106643
 0.62871612 0.66422094 0.67669327 0.67896229 0.68352563 0.6866238
 0.68931881 0.69255111 0.62387034 0.66564226 0.67978501 0.6865066
 0.68623153 0.68730584 0.6924547  0.69192835 0.61762407 0.66979593
 0.67424565 0.68871457 0.68677485 0.69080886 0.6905532  0.69727695
 0.62292326 0.66678393 0.67696308 0.68376933 0.68984325 0.68935398
 0.69231944 0.69667254 0.62193992 0.65446518 0.6829541  0.68582945
 0.69041161 0.68991146 0.69430382 0.69846896 0.63718363 0.66840856
 0.6824518  0.68669275 0.69227548 0.69314862 0.69626647 0.69934632]
```

In [137]:

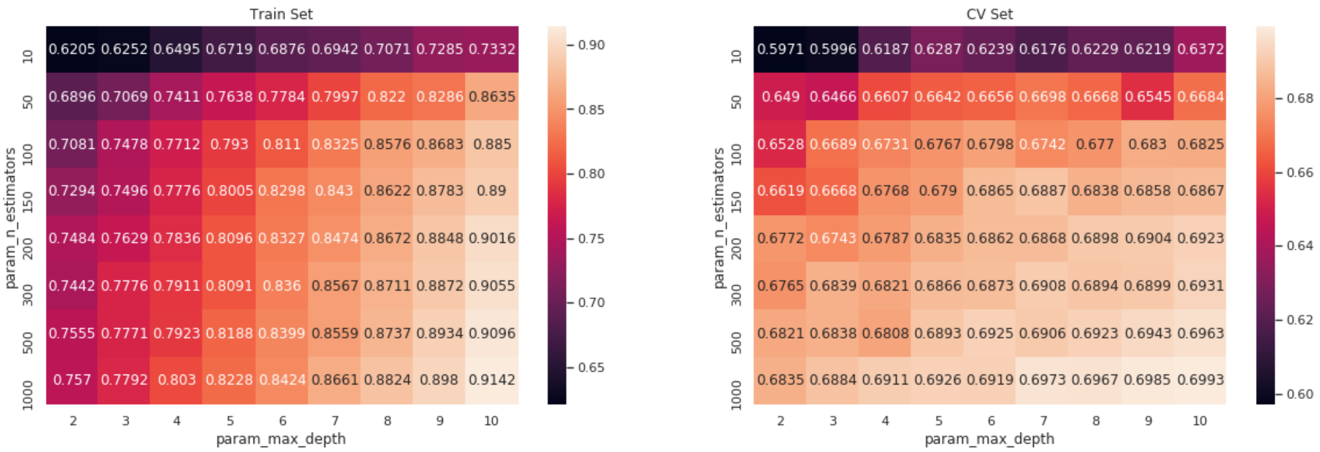
```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt



fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

g1 = list(gs.cv_results_['mean_train_score'])      #Train AUC Score
g2 = [2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,4,4,4,4,4,4,4,4,5,5,5,5,5,5,5,5,6,6,6,6,6,6,6,6,7,7,7,7,7,7,7,7,8
,8,8,8,8,8,8,8,9,9,9,9,9,9,9,9,10,10,10,10,10,10,10,10] #Depth
g3 = [10, 50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150,
200, 300, 500, 1000,10, 50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150, 200, 300, 500, 1000,10,
50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150, 200, 300,
500, 1000,10, 50, 100, 150, 200, 300, 500, 1000] #n_estimators



ax.scatter(g1, g2, g3, c='r', marker='o')

ax.set_xlabel('AUC Score')
ax.set_ylabel('Depth')
ax.set_zlabel('n_estimators')
```

```
plt.title('3D Scatter plot on Train AUC scores')
plt.show()
```



In [138]:

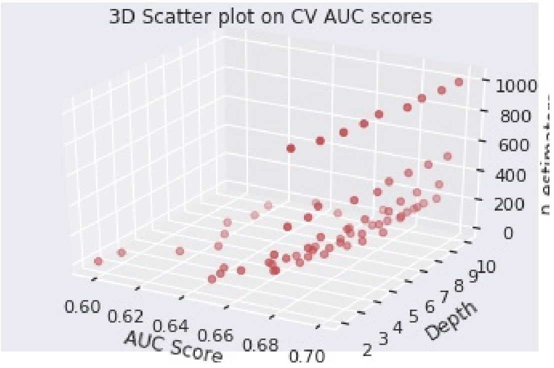
```
import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(gs.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max(
).unstack()[['mean_test_score', 'mean_train_score']]
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])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



In [139]:

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt



fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

g1 = list(gs.cv_results_['mean_test_score'])     #Train AUC Score
g2 = [2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,4,4,4,4,4,4,4,4,5,5,5,5,5,5,5,5,6,6,6,6,6,6,6,6,7,7,7,7,7,7,7,7,8
,8,8,8,8,8,8,8,9,9,9,9,9,9,9,9,10,10,10,10,10,10,10,10] #Depth
g3 = [10, 50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150,
200, 300, 500, 1000,10, 50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150, 200, 300, 500, 1000,10,
50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150, 200, 300,
500, 1000,10, 50, 100, 150, 200, 300, 500, 1000] #n_estimators




ax.scatter(g1, g2, g3, c='r', marker='o')

ax.set_xlabel('AUC Score')
ax.set_ylabel('Depth')
ax.set_zlabel('n_estimators')
```

```
plt.title('3D Scatter plot on CV AUC scores')
plt.show()
```



In [140]:

```
gs_results.best_params_
```

Out[140]:

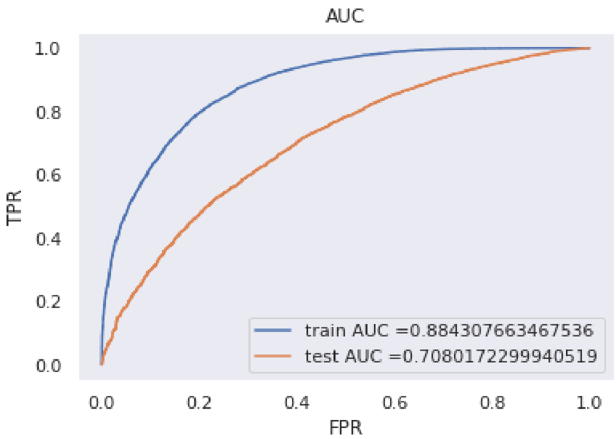
```
{'max_depth': 10, 'n_estimators': 1000}
```

In [141]:

```
max_d = gs_results.best_params_['max_depth']
n_est = gs_results.best_params_['n_estimators']
```

In [142]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
model = RandomForestClassifier(max_depth = 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 [143]:

```
#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) 0.6387695744050311 for threshold 0.84
Train confusion matrix
[[ 4011  1157]
 [ 5014 23318]]
```

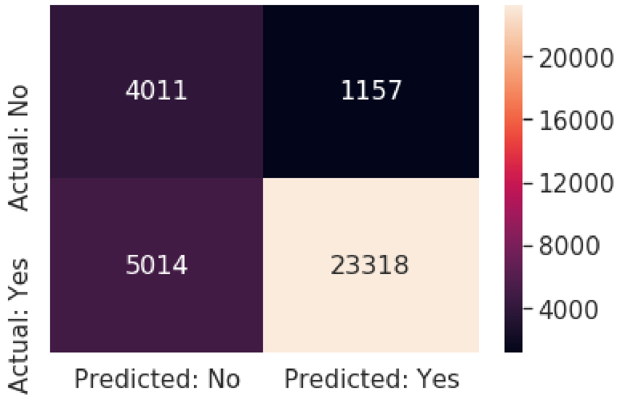
In [144]:

```
#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,
predict_with_best_t(y_train_pred, best_t)), ['Actual: No','Actual: Yes'],['Predicted:
No','Predicted: Yes'])
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[144]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f90a2640ba8>
```



In [145]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
Test confusion matrix
[[ 1305  1241]
 [ 3173 10781]]
```

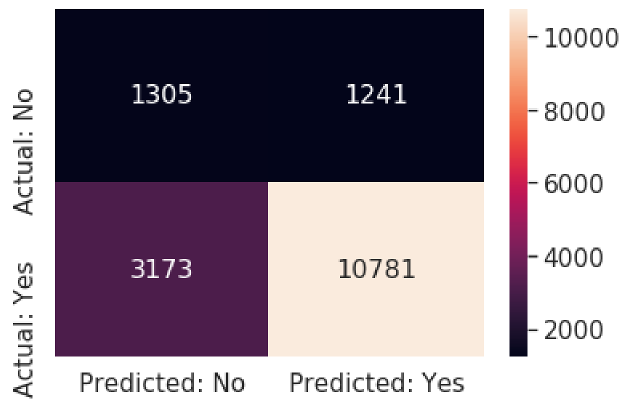
In [146]:

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

confusion_matrix_df_test = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, b
est_t)), ['Actual: No','Actual: Yes'],['Predicted: No','Predicted: Yes'])
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[146]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f90a2418c18>
```



2.4.3 Applying Random Forests on AVG W2V, SET 3

In [147]:

```
train_avg_w2v_essays_np = np.array(train_avg_w2v_essays)
train_avg_w2v_titles_np = np.array(train_avg_w2v_titles)
test_avg_w2v_essays_np  = np.array(test_avg_w2v_essays)
test_avg_w2v_titles_np  = np.array(test_avg_w2v_titles)
```

In [148]:

```
print(cat_0_train_normalized.shape)
print(cat_1_train_normalized.shape)
print(subcat_0_train_normalized.shape)
print(subcat_1_train_normalized.shape)
print(state_0_train_normalized.shape)
print(state_1_train_normalized.shape)
print(grade_0_train_normalized.shape)
print(grade_1_train_normalized.shape)
print(prefix_0_train_normalized.shape)
print(prefix_1_train_normalized.shape)
print(price_normalized_train.shape)
print(quantity_normalized_train.shape)
print(previously_posted_projects_normalized_train.shape)
print(title_word_count_normalized_train.shape)
print(essay_word_count_normalized_train.shape)
print(sent_pos_train.shape)
print(sent_neg_train.shape)
print(sent_neu_train.shape)
print(sent_compound_train.shape)
print(train_avg_w2v_essays_np.shape)
print(train_avg_w2v_titles_np.shape)
```

```
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
(33500, 1)
```

```
(33500, 300)
(33500, 300)
```

In [149]:

```
#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(cat_0_train_normalized)
tr2 = coo_matrix(cat_1_train_normalized)
tr3 = coo_matrix(subcat_0_train_normalized)
tr4 = coo_matrix(subcat_1_train_normalized)
tr5 = coo_matrix(state_0_train_normalized)
tr6 = coo_matrix(state_1_train_normalized)
tr7 = coo_matrix(grade_0_train_normalized)
tr8 = coo_matrix(grade_1_train_normalized)
tr9 = coo_matrix(prefix_0_train_normalized)
tr10 = coo_matrix(prefix_1_train_normalized)
tr11 = coo_matrix(price_normalized_train)
tr12 = coo_matrix(quantity_normalized_train)
tr13 = coo_matrix(previously_posted_projects_normalized_train)
tr14 = coo_matrix(title_word_count_normalized_train)
tr15 = coo_matrix(essay_word_count_normalized_train)
tr16 = coo_matrix(sent_pos_train)
tr17 = coo_matrix(sent_neg_train)
tr18 = coo_matrix(sent_neu_train)
tr19 = coo_matrix(sent_compound_train)
tr20 = coo_matrix(train_avg_w2v_essays_np)
tr21 = coo_matrix(train_avg_w2v_titles_np)
```

In [150]:

```
X_train = hstack([tr1,tr2,tr3,tr4,tr5,tr6,tr7,tr8,tr9,tr10,tr11,tr12,tr13,tr14,tr15,tr16,tr17,tr18,
tr19,tr20,tr21]).tocsr()
```

In [151]:

```
te1 = coo_matrix(cat_0_test_normalized)
te2 = coo_matrix(cat_1_test_normalized)
te3 = coo_matrix(subcat_0_test_normalized)
te4 = coo_matrix(subcat_1_test_normalized)
te5 = coo_matrix(state_0_test_normalized)
te6 = coo_matrix(state_1_test_normalized)
te7 = coo_matrix(grade_0_test_normalized)
te8 = coo_matrix(grade_1_test_normalized)
te9 = coo_matrix(prefix_0_test_normalized)
te10 = coo_matrix(prefix_1_test_normalized)
te11 = coo_matrix(price_normalized_test)
te12 = coo_matrix(quantity_normalized_test)
te13 = coo_matrix(previously_posted_projects_normalized_test)
te14 = coo_matrix(title_word_count_normalized_test)
te15 = coo_matrix(essay_word_count_normalized_test)
te16 = coo_matrix(sent_pos_test)
te17 = coo_matrix(sent_neg_test)
te18 = coo_matrix(sent_neu_test)
te19 = coo_matrix(sent_compound_test)
te20 = coo_matrix(test_avg_w2v_essays_np)
te21 = coo_matrix(test_avg_w2v_titles_np)
```

In [152]:

```
X_test = hstack([te1,te2,te3,te4,te5,te6,te7,te8,te9,te10,te11,te12,te13,te14,te15,te16,te17,te18,t
e19,te20,te21]).tocsr()
```

In [153]:

```
print(X_train.shape)
print(X_test.shape)
```

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

In [154]:

```
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV

rf = RandomForestClassifier()

grid_params = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth':[2, 3, 4, 5, 6,
7, 8, 9, 10]}

rs = RandomizedSearchCV(rf,grid_params ,cv=3, scoring='roc_auc',n_jobs=-1)
rs.fit(X_train, y_train)
```

Out[154]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
          estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
          fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth'
: [2, 3, 4, 5, 6, 7, 8, 9, 10]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring='roc_auc', verbose=0)
```

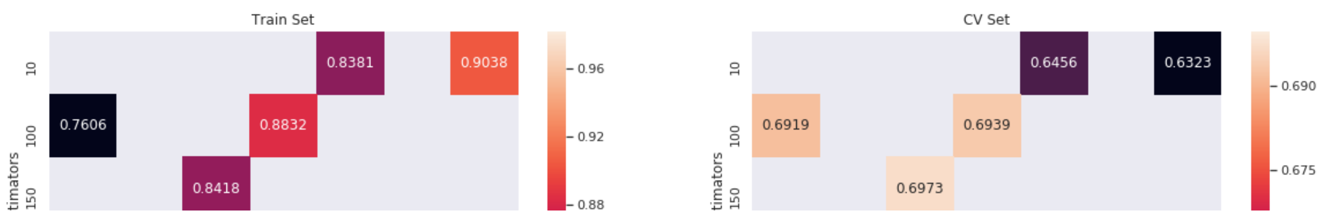
In [155]:

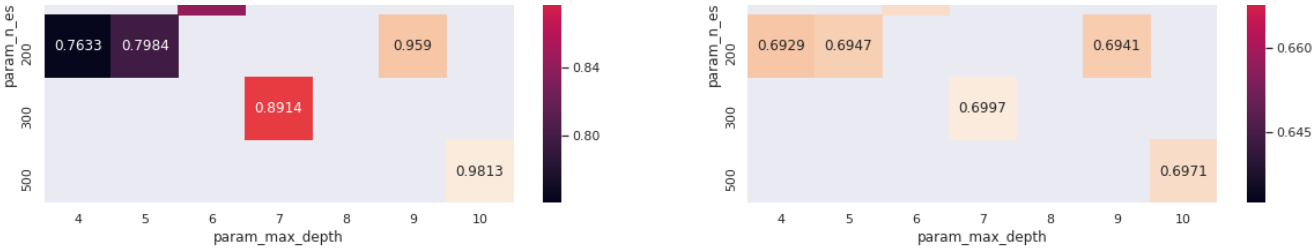
```
print('Best score: ',rs.best_score_)
print('k value with best score: ',rs.best_params_)
print('='*75)
print('Train AUC scores')
print(rs.cv_results_['mean_train_score'])
print('CV AUC scores')
print(rs.cv_results_['mean_test_score'])
```

```
Best score:  0.6997477968139622
k value with best score:  {'n_estimators': 300, 'max_depth': 7}
===========================================================================
Train AUC scores
[0.84175025 0.83813186 0.95897765 0.76061834 0.89142588 0.90383558
 0.98134637 0.79837164 0.88316523 0.76330023]
CV AUC scores
[0.69731014 0.64563891 0.69407684 0.69192079 0.6997478  0.63231555
 0.697081   0.69470458 0.69386893 0.69293507]
```

In [156]:

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



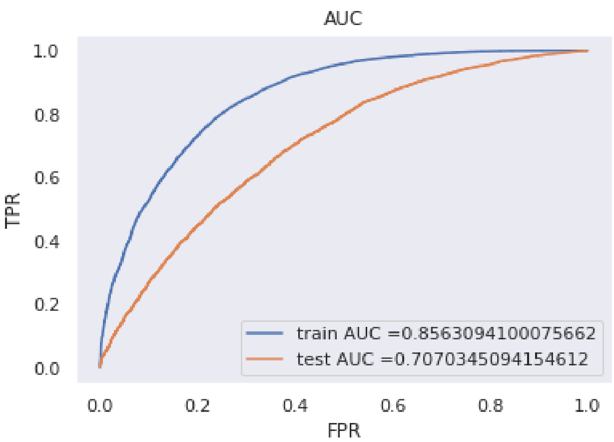


In [157]:

```
max_d = rs.best_params_['max_depth']
n_est = rs.best_params_['n_estimators']
```

In [158]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
model = RandomForestClassifier(max_depth = 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 [159]:

```
#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) 0.6017356835732354 for threshold 0.831
Train confusion matrix
[[ 3798  1370]
 [ 5134 23198]]
```

In [160]:

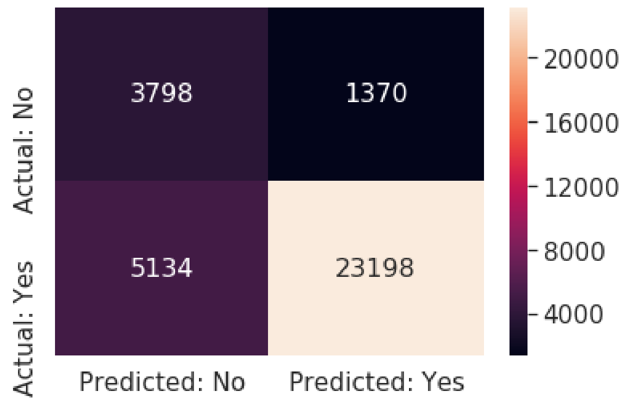
In [160]:

```
#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,
predict_with_best_t(y_train_pred, best_t)), ['Actual: No','Actual: Yes'],['Predicted:
No','Predicted: Yes'])
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[160]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f90a298be80>
```



In [161]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
Test confusion matrix
[[ 1462  1084]
 [ 3748 10206]]
```

In [162]:

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

confusion_matrix_df_test = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, b
est_t)), ['Actual: No','Actual: Yes'],['Predicted: No','Predicted: Yes'])
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[162]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f90a2903c18>
```

2.4.4 Applying Random Forests on TFIDF W2V, SET 4

In [163]:

```
train_tfidf_w2v_essays_np = np.array(train_tfidf_w2v_essays)
train_tfidf_w2v_titles_np = np.array(train_tfidf_w2v_titles)
test_tfidf_w2v_essays_np  = np.array(test_tfidf_w2v_essays)
test_tfidf_w2v_titles_np  = np.array(test_tfidf_w2v_titles)
```

In [164]:

```
#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(cat_0_train_normalized)
tr2 = coo_matrix(cat_1_train_normalized)
tr3 = coo_matrix(subcat_0_train_normalized)
tr4 = coo_matrix(subcat_1_train_normalized)
tr5 = coo_matrix(state_0_train_normalized)
tr6 = coo_matrix(state_1_train_normalized)
tr7 = coo_matrix(grade_0_train_normalized)
tr8 = coo_matrix(grade_1_train_normalized)
tr9 = coo_matrix(prefix_0_train_normalized)
tr10 = coo_matrix(prefix_1_train_normalized)
tr11 = coo_matrix(price_normalized_train)
tr12 = coo_matrix(quantity_normalized_train)
tr13 = coo_matrix(previously_posted_projects_normalized_train)
tr14 = coo_matrix(title_word_count_normalized_train)
tr15 = coo_matrix(essay_word_count_normalized_train)
tr16 = coo_matrix(sent_pos_train)
tr17 = coo_matrix(sent_neg_train)
tr18 = coo_matrix(sent_neu_train)
tr19 = coo_matrix(sent_compound_train)
tr20 = coo_matrix(train_tfidf_w2v_essays_np)
tr21 = coo_matrix(train_tfidf_w2v_titles_np)
```

In [165]:

```
X_train = hstack([tr1,tr2,tr3,tr4,tr5,tr6,tr7,tr8,tr9,tr10,tr11,tr12,tr13,tr14,tr15,tr16,tr17,tr18,
tr19,tr20,tr21]).tocsr()
```

In [166]:

```
te1 = coo_matrix(cat_0_test_normalized)
te2 = coo_matrix(cat_1_test_normalized)
te3 = coo_matrix(subcat_0_test_normalized)
te4 = coo_matrix(subcat_1_test_normalized)
te5 = coo_matrix(state_0_test_normalized)
te6 = coo_matrix(state_1_test_normalized)
te7 = coo_matrix(grade_0_test_normalized)
te8 = coo_matrix(grade_1_test_normalized)
te9 = coo_matrix(prefix_0_test_normalized)
te10 = coo_matrix(prefix_1_test_normalized)
te11 = coo_matrix(price_normalized_test)
te12 = coo_matrix(quantity_normalized_test)
te13 = coo_matrix(previously_posted_projects_normalized_test)
te14 = coo_matrix(title_word_count_normalized_test)
te15 = coo_matrix(essay_word_count_normalized_test)
te16 = coo_matrix(sent_pos_test)
te17 = coo_matrix(sent_neg_test)
te18 = coo_matrix(sent_neu_test)
te19 = coo_matrix(sent_compound_test)
te20 = coo_matrix(test_tfidf_w2v_essays_np)
te21 = coo_matrix(test_tfidf_w2v_titles_np)
```

In [167]:

```
X_test = hstack([te1,te2,te3,te4,te5,te6,te7,te8,te9,te10,te11,te12,te13,te14,te15,te16,te17,te18,t
e19,te20,te21]).tocsr()
```

In [168]:

```
print(X_train.shape)
print(X_test.shape)
```

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

In [169]:

```
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV

rf = RandomForestClassifier()

grid_params = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth':[2, 3, 4, 5, 6,
7, 8, 9, 10]}

rs = RandomizedSearchCV(rf,grid_params ,cv=3, scoring='roc_auc',n_jobs=-1)
rs.fit(X_train, y_train)
```

Out[169]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
          estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
          fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth'
: [2, 3, 4, 5, 6, 7, 8, 9, 10]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring='roc_auc', verbose=0)
```

In [170]:

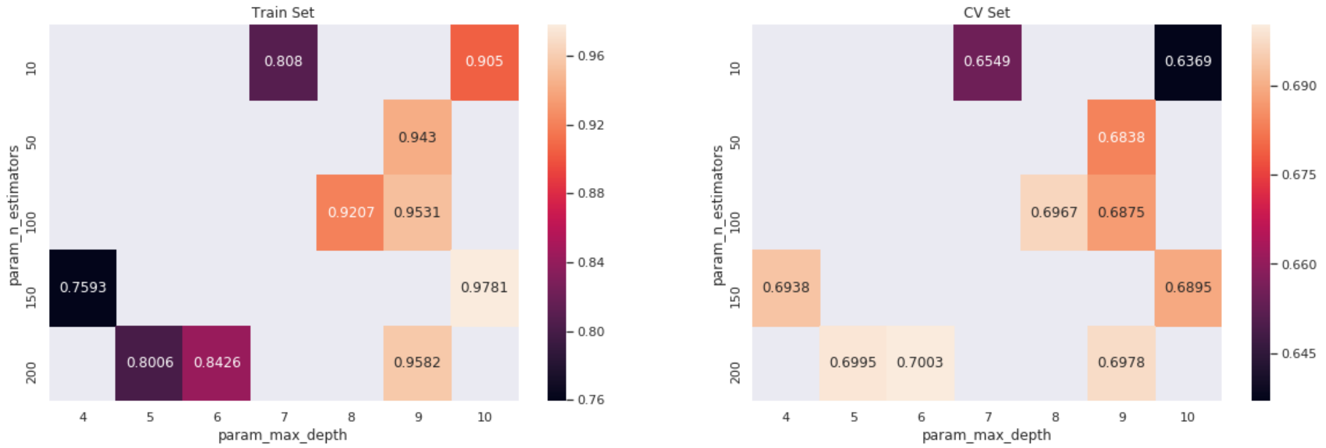
```
print('Best score: ',rs.best_score_)
print('k value with best score: ',rs.best_params_)
print('='*75)
print('Train AUC scores')
print(rs.cv_results_['mean_train_score'])
print('CV AUC scores')
print(rs.cv_results_['mean_test_score'])
```

```
Best score:  0.7002994051565903
k value with best score:  {'n_estimators': 200, 'max_depth': 6}
===========================================================================
Train AUC scores
[0.90499463 0.97805071 0.94297009 0.95821307 0.80803184 0.80055883
 0.92068165 0.84256625 0.75930702 0.95314876]
CV AUC scores
[0.63689366 0.68947082 0.68378449 0.69780267 0.65486867 0.69950839
 0.69673793 0.70029941 0.6938092  0.68754331]
```

In [171]:

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

```
ax[1].set_title('CV Set')
plt.show()
```

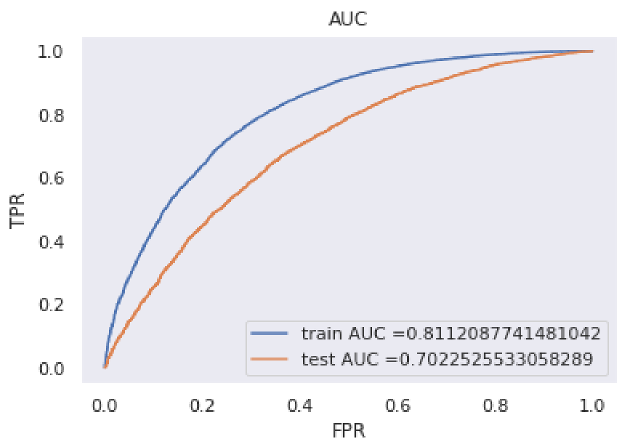


In [172]:

```
max_d = rs.best_params_['max_depth']
n_est = rs.best_params_['n_estimators']
```

In [173]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
model = RandomForestClassifier(max_depth = 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 [174]:

```
#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) 0.5430566086919842 for threshold 0.831
Train confusion matrix
[[ 3679  1489]
 [ 6719 21613]]
```

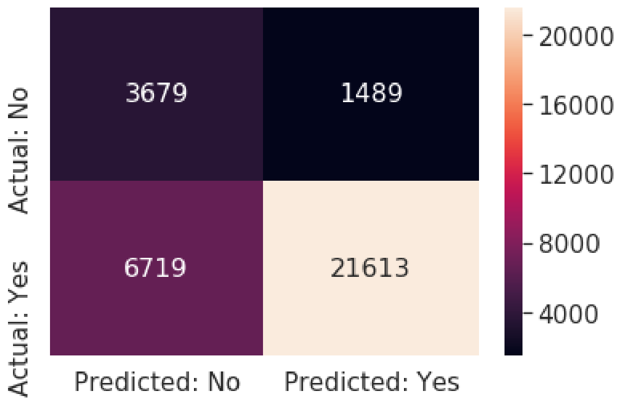
In [175]:

```
#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,
predict_with_best_t(y_train_pred, best_t)), ['Actual: No','Actual: Yes'],['Predicted:
No','Predicted: Yes'])
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[175]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f90a249a9b0>
```



In [176]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
Test confusion matrix
[[1519 1027]
 [4115 9839]]
```

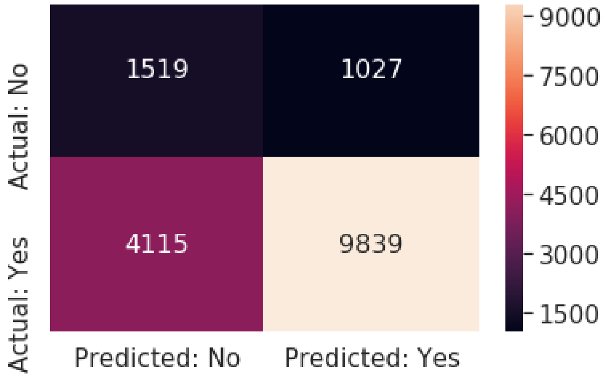
In [177]:

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

confusion_matrix_df_test = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, b
est_t)), ['Actual: No','Actual: Yes'],['Predicted: No','Predicted: Yes'])
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[177]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f90a21af4e0>
```



2.5 Applying GBDT

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 instrucations

2.5.1 Applying XGBOOST on BOW, SET 1

In [178]:

```
# Please write all the code with proper documentation
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_train = hstack((cat_0_train_normalized, cat_1_train_normalized, subcat_0_train_normalized,
subcat_1_train_normalized, state_0_train_normalized, state_1_train_normalized,
grade_0_train_normalized, grade_1_train_normalized, prefix_0_train_normalized,
prefix_1_train_normalized, price_normalized_train, quantity_normalized_train,
previously_posted_projects_normalized_train, title_word_count_normalized_train,
essay_word_count_normalized_train, sent_pos_train, sent_neg_train, sent_neu_train, sent_compound_tr
ain, train_title_bow, train_essay_bow)).tocsr()
X_test =  hstack((cat_0_test_normalized, cat_1_test_normalized, subcat_0_test_normalized, subcat_1_
test_normalized, state_0_test_normalized, state_1_test_normalized, grade_0_test_normalized,
grade_1_test_normalized, prefix_0_test_normalized, prefix_1_test_normalized, price_normalized_test
, quantity_normalized_test, previously_posted_projects_normalized_test,
title_word_count_normalized_test, essay_word_count_normalized_test, sent_pos_test, sent_neg_test,
sent_neu_test, sent_compound_test, test_title_bow, test_essay_bow)).tocsr()
```

In [179]:

```
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBClassifier

gbdt = XGBClassifier()

grid_params = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth':[2, 3, 4, 5, 6,
7, 8, 9, 10]}

rs = RandomizedSearchCV(gbdt,grid_params ,cv=3, scoring='roc_auc',n_jobs=-1)
rs.fit(X_train, y_train)
```

Out[179]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
          estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=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, 150, 200, 300, 500, 1000], 'max_depth'
: [2, 3, 4, 5, 6, 7, 8, 9, 10]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring='roc_auc', verbose=0)
```

```
        return_train_score='warn', scoring='roc_auc', verbose=0)
```

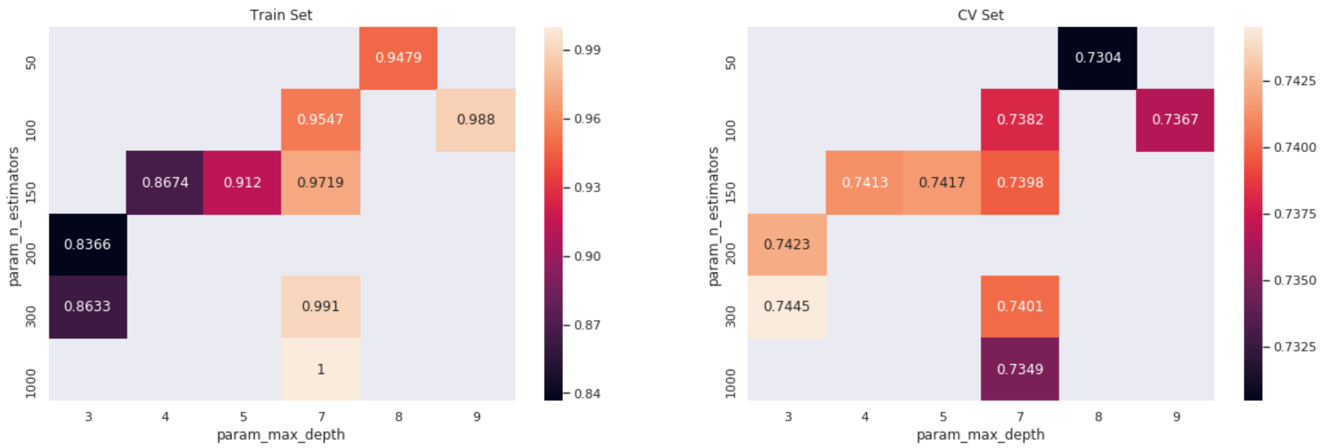
In [180]:

```
print('Best score: ',rs.best_score_)
print('k value with best score: ',rs.best_params_)
print('='*75)
print('Train AUC scores')
print(rs.cv_results_['mean_train_score'])
print('CV AUC scores')
print(rs.cv_results_['mean_test_score'])
```

```
Best score:  0.7445428354107052
k value with best score:  {'n_estimators': 300, 'max_depth': 3}
===========================================================================
Train AUC scores
[0.97187364 0.94792116 0.98796361 0.99998029 0.86738756 0.99097955
 0.9119555  0.8365766  0.95465996 0.86331461]
CV AUC scores
[0.73980478 0.73043996 0.73673138 0.73489207 0.74132502 0.74013574
 0.7416625  0.74225172 0.73815023 0.74454284]
```

In [181]:

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



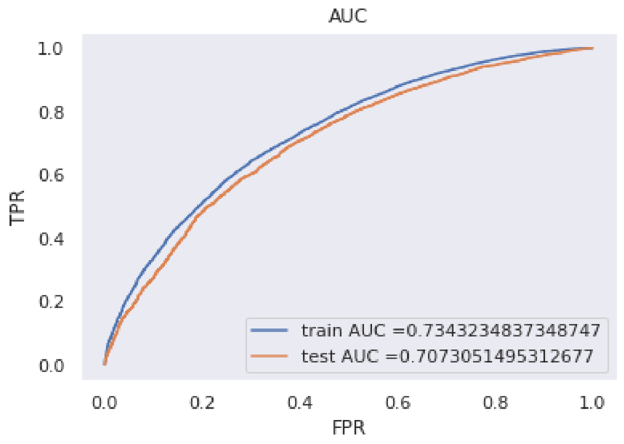
In [182]:

```
max_d = rs.best_params_['max_depth']
n_est = rs.best_params_['n_estimators']
```

In [183]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
model = RandomForestClassifier(max_depth = 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 [184]:

```
#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) 0.4505599025093441 for threshold 0.844
Train confusion matrix
[[ 3620  1548]
 [10108 18224]]
```

In [185]:

```
#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,
predict_with_best_t(y_train_pred, best_t)), ['Actual: No','Actual: Yes'],['Predicted:
No','Predicted: Yes'])
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[185]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f90a21f2978>
```

In [186]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
Test confusion matrix
[[1772  774]
 [5534 8420]]
```

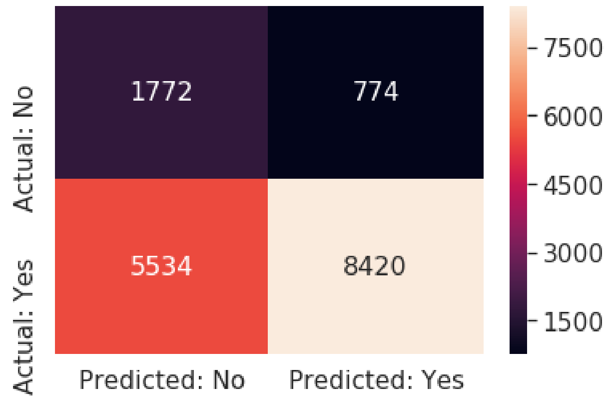
In [187]:

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

confusion_matrix_df_test = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, b
est_t)), ['Actual: No','Actual: Yes'],['Predicted: No','Predicted: Yes'])
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[187]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f90a29036d8>
```



2.5.2 Applying XGBOOST on TFIDF, SET 2

In [188]:

```
# Please write all the code with proper documentation
# Please write all the code with proper documentation
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_train = hstack((cat_0_train_normalized, cat_1_train_normalized, subcat_0_train_normalized,
subcat_1_train_normalized, state_0_train_normalized, state_1_train_normalized,
grade_0_train_normalized, grade_1_train_normalized, prefix_0_train_normalized,
prefix_1_train_normalized, price_normalized_train, quantity_normalized_train,
previously_posted_projects_normalized_train, title_word_count_normalized_train,
essay_word_count_normalized_train, sent_pos_train, sent_neg_train, sent_neu_train, sent_compound_tr
ain, train_title_tfidf, train_essay_tfidf)).tocsr()
X_test =  hstack((cat_0_test_normalized, cat_1_test_normalized, subcat_0_test_normalized, subcat_1_
test_normalized, state_0_test_normalized, state_1_test_normalized, grade_0_test_normalized,
grade_1_test_normalized, prefix_0_test_normalized, prefix_1_test_normalized, price_normalized_test
, quantity_normalized_test, previously_posted_projects_normalized_test,
title_word_count_normalized_test, essay_word_count_normalized_test, sent_pos_test, sent_neg_test,
sent_neu_test, sent_compound_test, test_title_tfidf, test_essay_tfidf)).tocsr()
```

In [189]:

```
from scipy.stats import randint as sp_randint
```

```python
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier

gbdt = XGBClassifier()

grid_params = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth':[2, 3, 4, 5, 6,
7, 8, 9, 10]}

gs = GridSearchCV(gbdt,grid_params ,cv=3, scoring='roc_auc',n_jobs=-1)
gs.fit(X_train, y_train)
```

Out[189]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=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_jobs=-1,
       param_grid={'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth': [2, 3, 4,
5, 6, 7, 8, 9, 10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [190]:

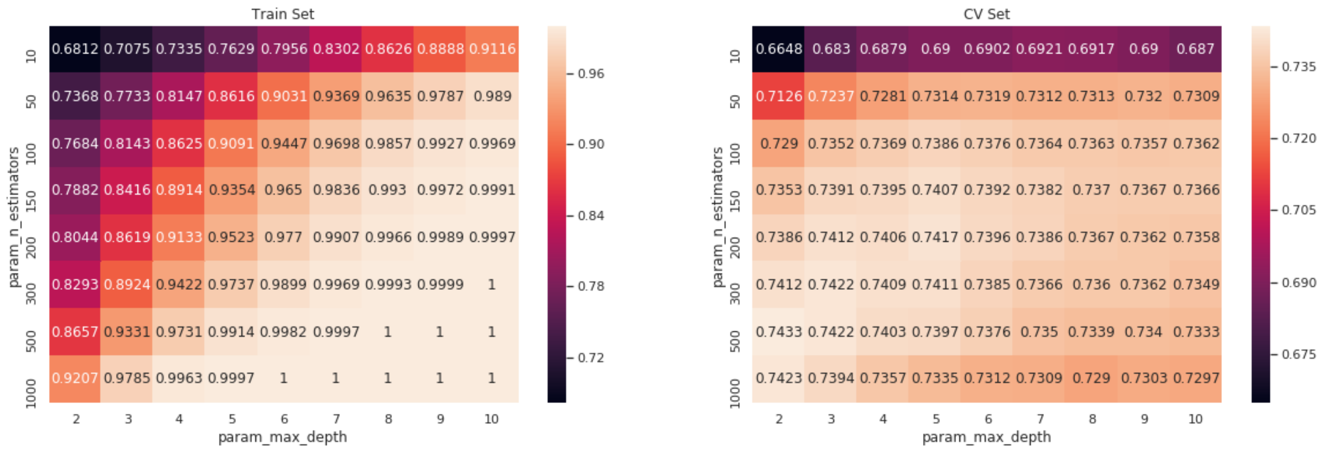
```python
print('Best score: ',gs.best_score_)
print('k value with best score: ',gs.best_params_)
print('='*75)
print('Train AUC scores')
print(gs.cv_results_['mean_train_score'])
print('CV AUC scores')
print(gs.cv_results_['mean_test_score'])
```

```
Best score:  0.7433302305643302
k value with best score:  {'max_depth': 2, 'n_estimators': 500}
===========================================================================
Train AUC scores
[0.68115019 0.73676062 0.76836407 0.78823845 0.80441724 0.82932553
 0.86570704 0.92070117 0.70752485 0.77334008 0.81427974 0.84163643
 0.86192643 0.89244153 0.93309575 0.97849695 0.73352769 0.8147087
 0.86245649 0.89140106 0.91325322 0.94216504 0.97306833 0.99632215
 0.76286364 0.86157073 0.90911482 0.93543796 0.95234262 0.97366148
 0.99141673 0.99972318 0.79560297 0.90306109 0.94469705 0.96498241
 0.97697677 0.98990733 0.99816021 0.99999035 0.83019117 0.93693913
 0.96981303 0.9836378  0.99072661 0.99693097 0.99974837 0.99999993
 0.86260292 0.96346486 0.98571197 0.99303822 0.99659839 0.99925442
 0.99998244 1.         0.88884922 0.97874784 0.9926957  0.99719582
 0.99890029 0.99985403 0.99999928 1.         0.91157387 0.98900437
 0.99694603 0.99910166 0.99974033 0.99998617 0.99999998 1.        ]
CV AUC scores
[0.66484381 0.71260835 0.72903438 0.73532406 0.7385961  0.74116106
 0.74333023 0.74226007 0.68301173 0.72368573 0.73517478 0.73908484
 0.7411675  0.7421667  0.74216047 0.73937124 0.68793084 0.72805897
 0.73693765 0.73952553 0.74055945 0.74094694 0.74025206 0.73574681
 0.69001933 0.73141227 0.73861582 0.74066747 0.74173229 0.74114642
 0.73972829 0.73346807 0.69018755 0.73191118 0.73756961 0.73919831
 0.73956543 0.73852259 0.73762986 0.73121086 0.69208561 0.73124825
 0.73638432 0.73821942 0.73861821 0.73661982 0.73496519 0.73093021
 0.6917223  0.73134097 0.73633469 0.73698713 0.73666056 0.73598628
 0.7338588  0.72897787 0.68999615 0.7319635  0.73573339 0.73672046
 0.73624823 0.73616801 0.73399777 0.7303095  0.6869721  0.73093094
 0.73621124 0.73658562 0.73582606 0.73489962 0.7332812  0.72968985]
```

In [191]:

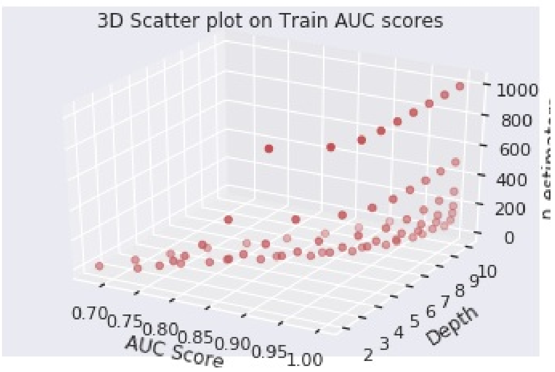
```python
import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(gs.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max(
).unstack()[['mean_test_score', 'mean_train_score']]
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
```

```
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



In [192]:

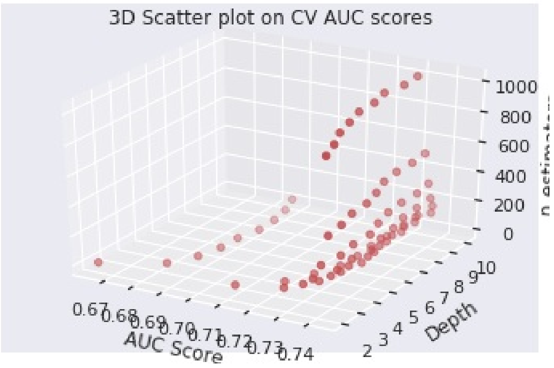
```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt



fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

g1 = list(gs.cv_results_['mean_train_score'])      #Train AUC Score
g2 = [2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,4,4,4,4,4,4,4,4,5,5,5,5,5,5,5,5,6,6,6,6,6,6,6,6,7,7,7,7,7,7,7,7,8
,8,8,8,8,8,8,8,9,9,9,9,9,9,9,9,10,10,10,10,10,10,10,10] #Depth
g3 = [10, 50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150,
200, 300, 500, 1000,10, 50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150, 200, 300, 500, 1000,10,
50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150, 200, 300,
500, 1000,10, 50, 100, 150, 200, 300, 500, 1000] #n_estimators



ax.scatter(g1, g2, g3, c='r', marker='o')

ax.set_xlabel('AUC Score')
ax.set_ylabel('Depth')
ax.set_zlabel('n_estimators')

plt.title('3D Scatter plot on Train AUC scores')
plt.show()
```



In [193]:

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt



fig = plt.figure()
```

```
ax = fig.add_subplot(111, projection='3d')

g1 = list(gs.cv_results_['mean_test_score'])      #Train AUC Score
g2 = [2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,4,4,4,4,4,4,4,4,5,5,5,5,5,5,5,5,6,6,6,6,6,6,6,6,7,7,7,7,7,7,7,7,8
,8,8,8,8,8,8,8,9,9,9,9,9,9,9,9,10,10,10,10,10,10,10,10] #Depth
g3 = [10, 50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150,
200, 300, 500, 1000,10, 50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150, 200, 300, 500, 1000,10,
50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150, 200, 300, 500, 1000,10, 50, 100, 150, 200, 300,
500, 1000,10, 50, 100, 150, 200, 300, 500, 1000] #n_estimators



ax.scatter(g1, g2, g3, c='r', marker='o')

ax.set_xlabel('AUC Score')
ax.set_ylabel('Depth')
ax.set_zlabel('n_estimators')

plt.title('3D Scatter plot on CV AUC scores')
plt.show()
```



In [197]:

```
gs.best_params_
```

Out[197]:

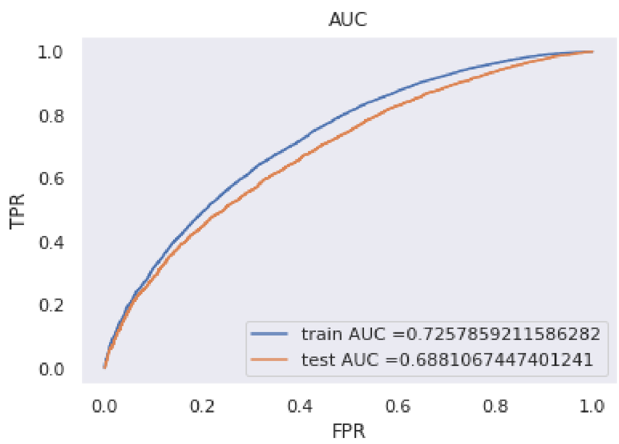
```
{'max_depth': 2, 'n_estimators': 500}
```

In [198]:

```
max_d = gs.best_params_['max_depth']
n_est = gs.best_params_['n_estimators']
```

In [201]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
model = RandomForestClassifier(max_depth = 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 [202]:

```
#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) 0.4396031790131956 for threshold 0.845
Train confusion matrix
[[ 3531  1637]
 [10103 18229]]
```

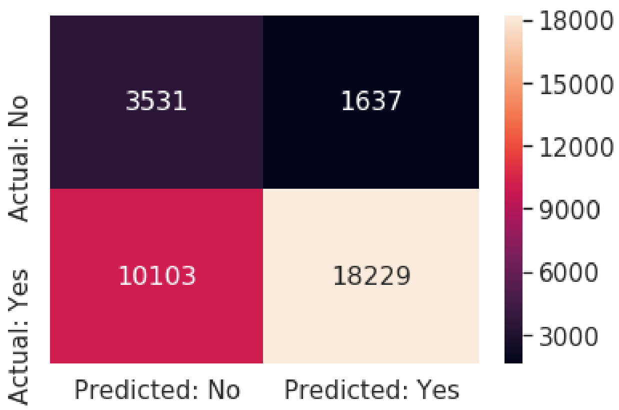
In [203]:

```
#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,
predict_with_best_t(y_train_pred, best_t)), ['Actual: No','Actual: Yes'],['Predicted:
No','Predicted: Yes'])
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[203]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f90a227aef0>
```



In [204]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
Test confusion matrix
[[ 1230  1316]
 [ 3322 10632]]
```

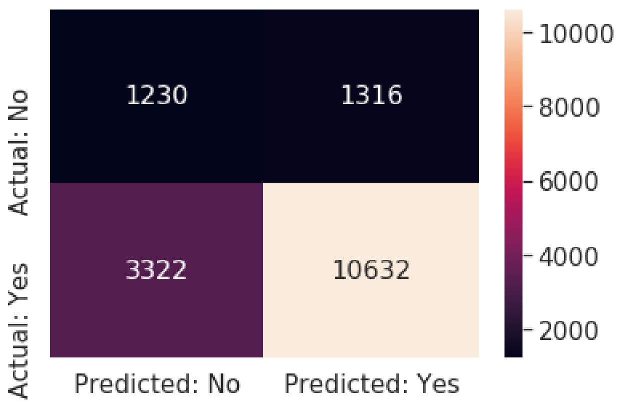
In [205]:

```python
print("Test data confusion matrix")

confusion_matrix_df_test = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, b
est_t)), ['Actual: No','Actual: Yes'],['Predicted: No','Predicted: Yes'])
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[205]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f90a1c674a8>
```



2.5.3 Applying XGBOOST on AVG W2V, SET 3

In [206]:

```python
# Please write all the code with proper documentation
train_avg_w2v_essays_np = np.array(train_avg_w2v_essays)
train_avg_w2v_titles_np = np.array(train_avg_w2v_titles)
test_avg_w2v_essays_np  = np.array(test_avg_w2v_essays)
test_avg_w2v_titles_np  = np.array(test_avg_w2v_titles)
```

In [207]:

```python
#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(cat_0_train_normalized)
tr2 = coo_matrix(cat_1_train_normalized)
tr3 = coo_matrix(subcat_0_train_normalized)
tr4 = coo_matrix(subcat_1_train_normalized)
tr5 = coo_matrix(state_0_train_normalized)
tr6 = coo_matrix(state_1_train_normalized)
tr7 = coo_matrix(grade_0_train_normalized)
tr8 = coo_matrix(grade_1_train_normalized)
tr9 = coo_matrix(prefix_0_train_normalized)
tr10 = coo_matrix(prefix_1_train_normalized)
tr11 = coo_matrix(price_normalized_train)
tr12 = coo_matrix(quantity_normalized_train)
tr13 = coo_matrix(previously_posted_projects_normalized_train)
tr14 = coo_matrix(title_word_count_normalized_train)
tr15 = coo_matrix(essay_word_count_normalized_train)
tr16 = coo_matrix(sent_pos_train)
tr17 = coo_matrix(sent_neg_train)
tr18 = coo_matrix(sent_neu_train)
```

```
tr19 = coo_matrix(sent_compound_train)
tr20 = coo_matrix(train_avg_w2v_essays_np)
tr21 = coo_matrix(train_avg_w2v_titles_np)
```

In [208]:

```
X_train = hstack([tr1,tr2,tr3,tr4,tr5,tr6,tr7,tr8,tr9,tr10,tr11,tr12,tr13,tr14,tr15,tr16,tr17,tr18,
tr19,tr20,tr21]).tocsr()
```

In [210]:

```
te1 = coo_matrix(cat_0_test_normalized)
te2 = coo_matrix(cat_1_test_normalized)
te3 = coo_matrix(subcat_0_test_normalized)
te4 = coo_matrix(subcat_1_test_normalized)
te5 = coo_matrix(state_0_test_normalized)
te6 = coo_matrix(state_1_test_normalized)
te7 = coo_matrix(grade_0_test_normalized)
te8 = coo_matrix(grade_1_test_normalized)
te9 = coo_matrix(prefix_0_test_normalized)
te10 = coo_matrix(prefix_1_test_normalized)
te11 = coo_matrix(price_normalized_test)
te12 = coo_matrix(quantity_normalized_test)
te13 = coo_matrix(previously_posted_projects_normalized_test)
te14 = coo_matrix(title_word_count_normalized_test)
te15 = coo_matrix(essay_word_count_normalized_test)
te16 = coo_matrix(sent_pos_test)
te17 = coo_matrix(sent_neg_test)
te18 = coo_matrix(sent_neu_test)
te19 = coo_matrix(sent_compound_test)
te20 = coo_matrix(test_avg_w2v_essays_np)
te21 = coo_matrix(test_avg_w2v_titles_np)
```

In [211]:

```
X_test = hstack([te1,te2,te3,te4,te5,te6,te7,te8,te9,te10,te11,te12,te13,te14,te15,te16,te17,te18,t
e19,te20,te21]).tocsr()
```

In [212]:

```
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBClassifier

gbdt = XGBClassifier()

grid_params = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth':[2, 3, 4, 5, 6,
7, 8, 9, 10]}

rs = RandomizedSearchCV(gbdt,grid_params ,cv=3, scoring='roc_auc',n_jobs=-1)
rs.fit(X_train, y_train)
```

Out[212]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
          estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=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, 150, 200, 300, 500, 1000], 'max_depth'
: [2, 3, 4, 5, 6, 7, 8, 9, 10]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring='roc_auc', verbose=0)
```

In [213]:

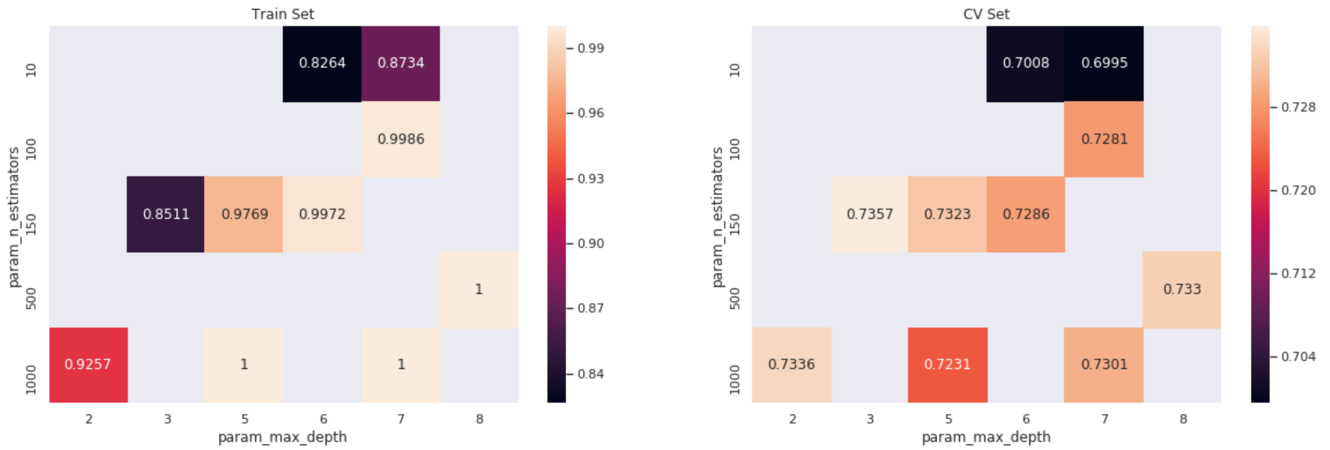
```
print('Best score: ',rs.best_score_)
print('k value with best score: ',rs.best_params_)
```

```
print('k value with best score: ',rs.best_params_)
print('='*75)
print('Train AUC scores')
print(rs.cv_results_['mean_train_score'])
print('CV AUC scores')
print(rs.cv_results_['mean_test_score'])
```

```
Best score:  0.7357391660102338
k value with best score:  {'n_estimators': 150, 'max_depth': 3}
===========================================================================
Train AUC scores
[0.99859411 0.97690542 0.99718371 1.         1.         1.
 0.8511252  0.87344236 0.82639463 0.92565958]
CV AUC scores
[0.72813414 0.7323405  0.72856256 0.73014832 0.72306499 0.73296479
 0.73573917 0.69952303 0.70084636 0.73363487]
```

In [214]:

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



In [215]:

```
rs.best_params_
```

Out[215]:

```
{'n_estimators': 150, 'max_depth': 3}
```

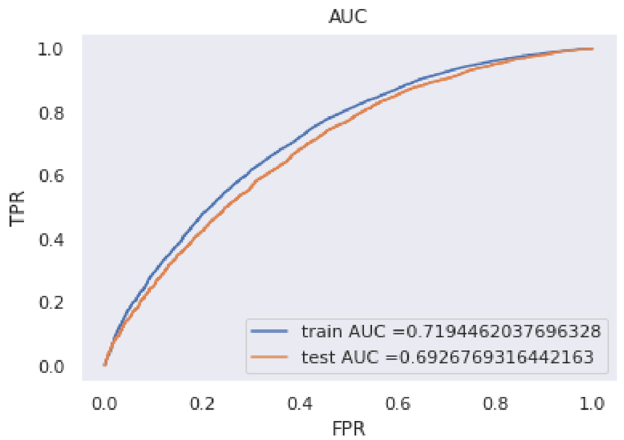
In [216]:

```
max_d = rs.best_params_['max_depth']
n_est = rs.best_params_['n_estimators']
```

In [217]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
model = RandomForestClassifier(max_depth = 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)
```

```
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 [218]:

```
#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) 0.4366130706278365 for threshold 0.841
Train confusion matrix
[[ 3311  1857]
 [ 9024 19308]]
```

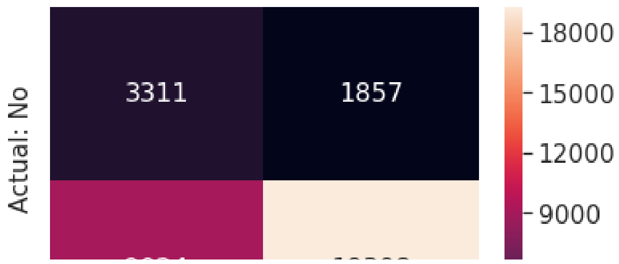
In [219]:

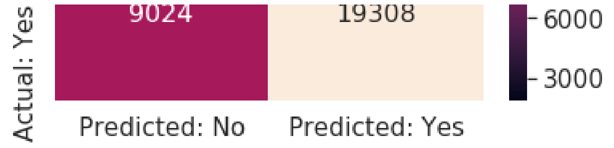
```
#plotting confusion matrix using seaborn's heatmap
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

print("Train data confusion matrix")

confusion_matrix_df_train = pd.DataFrame(confusion_matrix(y_train,
predict_with_best_t(y_train_pred, best_t)), ['Actual: No','Actual: Yes'],['Predicted:
No','Predicted: Yes'])
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[219]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f90a1bd7978>
```





In [220]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
Test confusion matrix
[[1579  967]
 [4821 9133]]
```

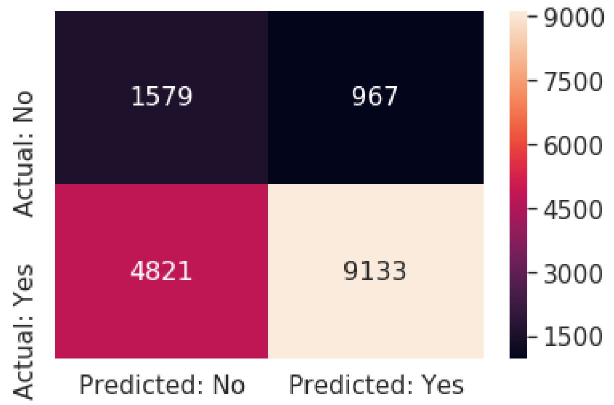
In [221]:

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

confusion_matrix_df_test = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, b
est_t)), ['Actual: No','Actual: Yes'],['Predicted: No','Predicted: Yes'])
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[221]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f90a1b221d0>
```



2.5.4 Applying XGBOOST on TFIDF W2V, SET 4

In [222]:

```
# Please write all the code with proper documentation
train_tfidf_w2v_essays_np = np.array(train_tfidf_w2v_essays)
train_tfidf_w2v_titles_np = np.array(train_tfidf_w2v_titles)
test_tfidf_w2v_essays_np  = np.array(test_tfidf_w2v_essays)
test_tfidf_w2v_titles_np  = np.array(test_tfidf_w2v_titles)
```

In [223]:

```
#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(cat_0_train_normalized)
tr2 = coo_matrix(cat_1_train_normalized)
tr3 = coo_matrix(subcat_0_train_normalized)
tr4 = coo_matrix(subcat_1_train_normalized)
tr5 = coo_matrix(state_0_train_normalized)
tr6 = coo_matrix(state_1_train_normalized)
```

```
tr7 = coo_matrix(grade_0_train_normalized)
tr8 = coo_matrix(grade_1_train_normalized)
tr9 = coo_matrix(prefix_0_train_normalized)
tr10 = coo_matrix(prefix_1_train_normalized)
tr11 = coo_matrix(price_normalized_train)
tr12 = coo_matrix(quantity_normalized_train)
tr13 = coo_matrix(previously_posted_projects_normalized_train)
tr14 = coo_matrix(title_word_count_normalized_train)
tr15 = coo_matrix(essay_word_count_normalized_train)
tr16 = coo_matrix(sent_pos_train)
tr17 = coo_matrix(sent_neg_train)
tr18 = coo_matrix(sent_neu_train)
tr19 = coo_matrix(sent_compound_train)
tr20 = coo_matrix(train_tfidf_w2v_essays_np)
tr21 = coo_matrix(train_tfidf_w2v_titles_np)
```

In [224]:

```
X_train = hstack([tr1,tr2,tr3,tr4,tr5,tr6,tr7,tr8,tr9,tr10,tr11,tr12,tr13,tr14,tr15,tr16,tr17,tr18,
tr19,tr20,tr21]).tocsr()
```

In [225]:

```
te1 = coo_matrix(cat_0_test_normalized)
te2 = coo_matrix(cat_1_test_normalized)
te3 = coo_matrix(subcat_0_test_normalized)
te4 = coo_matrix(subcat_1_test_normalized)
te5 = coo_matrix(state_0_test_normalized)
te6 = coo_matrix(state_1_test_normalized)
te7 = coo_matrix(grade_0_test_normalized)
te8 = coo_matrix(grade_1_test_normalized)
te9 = coo_matrix(prefix_0_test_normalized)
te10 = coo_matrix(prefix_1_test_normalized)
te11 = coo_matrix(price_normalized_test)
te12 = coo_matrix(quantity_normalized_test)
te13 = coo_matrix(previously_posted_projects_normalized_test)
te14 = coo_matrix(title_word_count_normalized_test)
te15 = coo_matrix(essay_word_count_normalized_test)
te16 = coo_matrix(sent_pos_test)
te17 = coo_matrix(sent_neg_test)
te18 = coo_matrix(sent_neu_test)
te19 = coo_matrix(sent_compound_test)
te20 = coo_matrix(test_tfidf_w2v_essays_np)
te21 = coo_matrix(test_tfidf_w2v_titles_np)
```

In [226]:

```
X_test = hstack([te1,te2,te3,te4,te5,te6,te7,te8,te9,te10,te11,te12,te13,te14,te15,te16,te17,te18,t
e19,te20,te21]).tocsr()
```

In [227]:

```
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBClassifier

gbdt = XGBClassifier()

grid_params = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth':[2, 3, 4, 5, 6,
7, 8, 9, 10]}

rs = RandomizedSearchCV(gbdt,grid_params ,cv=3, scoring='roc_auc',n_jobs=-1)
rs.fit(X_train, y_train)
```

Out[227]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
          estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=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, 150, 200, 300, 500, 1000], 'max_depth'
: [2, 3, 4, 5, 6, 7, 8, 9, 10]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring='roc_auc', verbose=0)
```

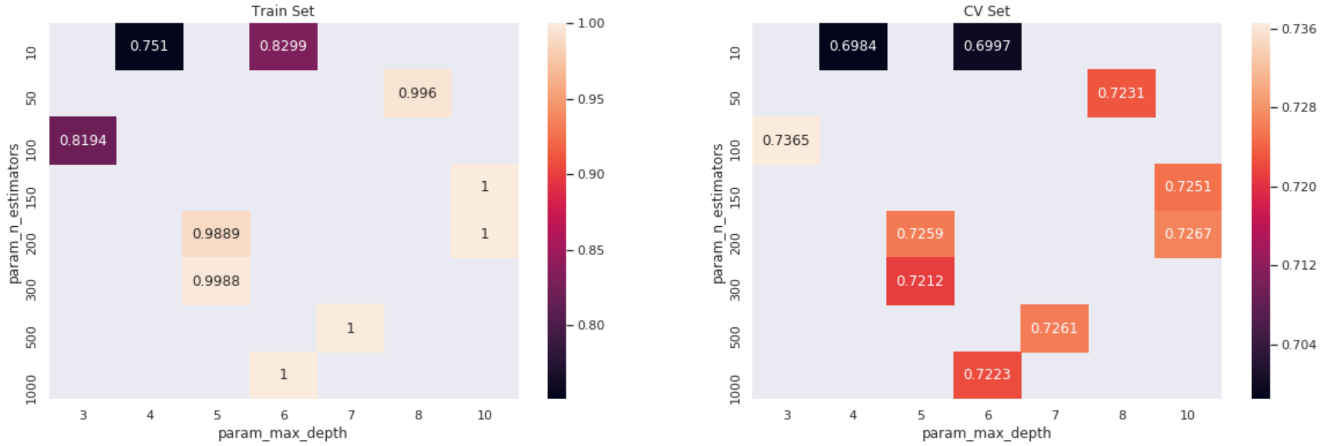
In [228]:

```
print('Best score: ',rs.best_score_)
print('k value with best score: ',rs.best_params_)
print('='*75)
print('Train AUC scores')
print(rs.cv_results_['mean_train_score'])
print('CV AUC scores')
print(rs.cv_results_['mean_test_score'])
```

```
Best score:  0.736547312206274
k value with best score:  {'n_estimators': 100, 'max_depth': 3}
===========================================================================
Train AUC scores
[0.82991714 0.98885468 1.         0.75100817 0.8194429  1.
 0.996027   1.         1.         0.99876219]
CV AUC scores
[0.69965447 0.72589447 0.72509977 0.69844041 0.73654731 0.72613819
 0.72311922 0.72228901 0.72665804 0.72117126]
```

In [229]:

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



In [230]:

```
rs.best_params_
```

Out[230]:

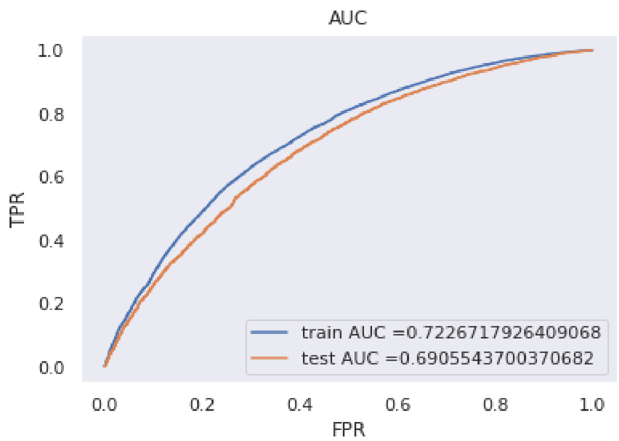
```
{'n_estimators': 100, 'max_depth': 3}
```

In [231]:

```
max_d = rs.best_params_['max_depth']
n_est = rs.best_params_['n_estimators']
```

In [232]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
model = RandomForestClassifier(max_depth = 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 [233]:

```
#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) 0.4442493751663709 for threshold 0.84
Train confusion matrix
[[ 3421  1747]
 [ 9318 19014]]
```

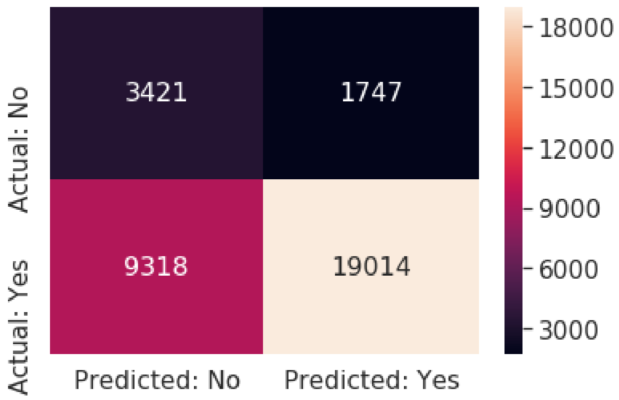
In [234]:

```
#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,
predict_with_best_t(y_train_pred, best_t)), ['Actual: No','Actual: Yes'],['Predicted:
No','Predicted: Yes'])
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[234]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f90a1b166d8>
```



In [235]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
Test confusion matrix
[[1628  918]
 [4982 8972]]
```

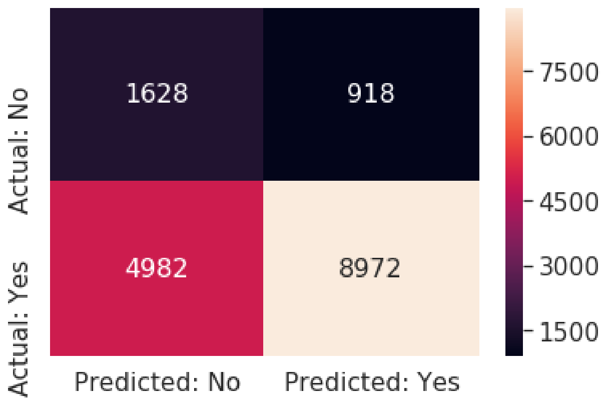
In [236]:

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

confusion_matrix_df_test = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, b
est_t)), ['Actual: No','Actual: Yes'],['Predicted: No','Predicted: Yes'])
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[236]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f90a1ba3358>
```



3. Conclusion

In [238]:

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

# Please compare all your models using Prettytable library

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

from prettytable import PrettyTable
```

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

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyperparameters(n_estimators,max_depth)", "Test AUC"]

x.add_row(["BOW", "RF","(1000, 10)", 0.722])
x.add_row(["TFIDF", "RF", "(1000, 10)",  0.708])
x.add_row(["AVG W2V", "RF", "(300, 7)",  0.707])
x.add_row(["TFIDF W2V", "RF", "(200, 6)",  0.702])

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

x.add_row(["BOW", "GBDT","(300, 3)",  0.707])
x.add_row(["TFIDF", "GBDT", "(500, 2)",  0.688])
x.add_row(["AVG W2V", "GBDT", "(150, 3)",  0.69])
x.add_row(["TFIDF W2V", "GBDT", "(100, 3)", 0.69])



print(x)
```

```
+------------+-------+-----------------------------------------+-----------+
| Vectorizer | Model | Hyperparameters(n_estimators,max_depth) |  Test AUC |
+------------+-------+-----------------------------------------+-----------+
|    BOW     |   RF  |                (1000, 10)               |   0.722   |
|   TFIDF    |   RF  |                (1000, 10)               |   0.708   |
|  AVG W2V   |   RF  |                 (300, 7)                |   0.707   |
| TFIDF W2V  |   RF  |                 (200, 6)                |   0.702   |
| ----- ---  |  ---- |     --------------------------------    | --------- |
|    BOW     |  GBDT |                 (300, 3)                |   0.707   |
|   TFIDF    |  GBDT |                 (500, 2)                |   0.688   |
|  AVG W2V   |  GBDT |                 (150, 3)                |    0.69   |
| TFIDF W2V  |  GBDT |                 (100, 3)                |    0.69   |
+------------+-------+-----------------------------------------+-----------+
```