

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
<code>project_id</code>		A unique identifier for the proposed project. Example: 123456789
<code>project_title</code>		Title of the project. Example: Art Will Make You a Better Person
<code>project_grade_category</code>		Grade level of students for which the project is targeted. One of the following enumerated list of categories: <ul style="list-style-type: none">• Grades K-2• Grades 3-5• Grades 6-8• Grades 9-12
<code>project_subject_categories</code>		One or more (comma-separated) subject categories for the project from the following enumerated list of categories: <ul style="list-style-type: none">• Applied & Design• Care & Safety• Health & Physical Education• History & Social Studies• Literacy & Language• Math & Science• Music & Arts• Special Education
<code>project_subject_subcategories</code>		One or more (comma-separated) subject subcategories for the project from the following enumerated list of categories: <ul style="list-style-type: none">• Music & Arts• Literacy & Language, Math & Science
<code>school_state</code>		State where school is located (Two-letter U.S. postal abbreviations) (https://en.wikipedia.org/wiki/List of U.S. state abbreviations#Postal abbreviations)
<code>project_resource_summary</code>		An explanation of the resources needed for the project. Example: My students need hands on literacy materials to enhance their sensory
<code>project_essay_1</code>		First application essay
<code>project_essay_2</code>		Second application essay
<code>project_essay_3</code>		Third application essay
<code>project_essay_4</code>		Fourth application essay
<code>project_submitted_datetime</code>		Datetime when project application was submitted. Example: 2018-01-12T12:43:00Z
<code>teacher_id</code>		A unique identifier for the teacher of the proposed project. Example: bdf8baa8fedef6bfeec7ae4f1

teacher_prefix

-

Number of project applications previously submitted by the sam

Exe

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:

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):

Notes on the Essay Data

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

- __project_essay_1: __ "Introduce us to your classroom"
- __project_essay_2: __ "Tell us more about your students"
- __project_essay_3: __ "Describe how your students will use the materials you're requesting"
- project_essay_3: __ "Close by sharing why your project will make a difference"

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

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

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

In [1]:

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

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

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

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

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

from tqdm import tqdm
import os

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

1.1 Reading Data

In [2]:

```
project_data = pd.read_csv('../train_data.csv')
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 (109248, 17)

The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix'
'school_state'
'project_submitted_datetime' 'project_grade_category'
'project_subject_categories' 'project_subject_subcategories'
'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
'project_essay_4' 'project_resource_summary'
'teacher_number_of_previously_posted_projects' 'project_is_approved']

In [4]:

```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.col
umns)]

#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/40840
39
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
project_data.drop('project_submitted_datetime', axis=1, inplace=True)
project_data.sort_values(by=['Date'], inplace=True)

# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
project_data = project_data[cols]

project_data.head(2)
```

Out[4]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT



In [5]:

```
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[5]:

	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 [6]:

```
categories = 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 categories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the category based on space "Math & Science" => "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 replacing 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 [7]:

```
sub_categories = 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_categories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the category based on space "Math & Science" => "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 replacing all the ' ' (space) with '' (empty) ex: "Math & Science" => "Math&Science"
            temp += j.strip() + " #"
    # abc ".strip() will return "abc", remove the trailing spaces
    temp = temp.replace('&', '_')
    sub_cat_list.append(temp.strip())

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

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

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

1.3 Text preprocessing

In [8]:

```
# 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 [9]:

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

Out[9]:

Unnamed: 0		id	teacher_id	teacher_prefix	school_state
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT

In [10]:

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


In [11]:

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

I have been fortunate enough to use the Fairy Tale STEM kits in my classroom as well as the STEM journals, which my students really enjoyed. I would love to implement more of the Lakeshore STEM kits in my classroom for the next school year as they provide excellent and engaging STEM lessons. My students come from a variety of backgrounds, including language and socioeconomic status. Many of them don't have a lot of experience in science and engineering and these kits give me the materials to provide these exciting opportunities for my students. Each month I try to do several science or STEM/STEAM projects. I would use the kits and robot to help guide my science instruction in engaging and meaningful ways. I can adapt the kits to my current language arts pacing guide where we already teach some of the material in the kits like tall tales (Paul Bunyan) or Johnny Appleseed. The following units will be taught in the next school year where I will implement these kits: magnets, motion, sink vs. float, robots. I often get to these units and don't know if I am teaching the right way or using the right materials. The kits will give me additional ideas, strategies, and lessons to prepare my students in science. It is challenging to develop high quality science activities. These kits give me the materials I need to provide my students with science activities that will go along with the curriculum in my classroom. Although I have some things (like magnets) in my classroom, I don't know how to use them effectively. The kits will provide me with the right amount of materials and show me how to use them in an appropriate way.

=====

I teach high school English to students with learning and behavioral disabilities. My students all vary in their ability level. However, the ultimate goal is to increase all students literacy levels. This includes their reading, writing, and communication levels. I teach a really dynamic group of students. However, my students face a lot of challenges. My students all live in poverty and in a dangerous neighborhood. Despite these challenges, I have students who have the desire to defeat these challenges. My students all have learning disabilities and currently all are performing below grade level. My students are visual learners and will benefit from a classroom that fulfills their preferred learning style. The materials I am requesting will allow my students to be prepared for the classroom with the necessary supplies. Too often I am challenged with students who come to school unprepared for class due to economic challenges. I want my students to be able to focus on learning and not how they will be able to get school supplies. The supplies will last all year. Students will be able to complete written assignments and maintain a classroom journal. The chart paper will be used to make learning more visual in class and to create posters to aid students in their learning. The students have access to a classroom printer. The toner will be used to print student work that is completed on the classroom Chromebooks. I want to try and remove all barriers for the students learning and create opportunities for learning. One of the biggest barriers is the students not having the resources to get pens, paper, and folders. My students will be able to increase their literacy skills because of this project.

=====

"Life moves pretty fast. If you don't stop and look around once in awhile, you could miss it." from the movie, Ferris Bueller's Day Off. Think back...what do you remember about your grandparents? How amazing would it be to be able to flip through a book to see a day in their lives? My second graders are voracious readers! They love to read both fiction and nonfiction books. Their favorite characters include Pete the Cat, Fly Guy, Piggie and Elephant, and Mercy Watson. They also love to read about insects, space and plants. My students are hungry bookworms! My students are eager to learn and read about the world around them. My kids love to be at school and are like little sponges absorbing everything around them. Their parents work long hours and usually do not see their children. My students are usually cared for by their grandparents or a family friend. Most of my students

ts do not have someone who speaks English at home. Thus it is difficult for my students to acquire language. Now think forward... wouldn't it mean a lot to your kids, nieces or nephews or grandchildren, to be able to see a day in your life today 30 years from now? Memories are so precious to us and being able to share these memories with future generations will be a rewarding experience. As part of our social studies curriculum, students will be learning about changes over time. Students will be studying photos to learn about how their community has changed over time. In particular, we will look at photos to study how the land, buildings, clothing, and schools have changed over time. As a culminating activity, my students will capture a slice of their history and preserve it through scrap booking. Key important events in their young lives will be documented with the date, location, and names. Students will be using photos from home and from school to create their second grade memories. Their scrap books will preserve their unique stories for future generations to enjoy. Your donation to this project will provide my second graders with an opportunity to learn about social studies in a fun and creative manner. Through their scrapbooks, children will share their story with others and have a historical document for the rest of their lives.

=====

"A person's a person, no matter how small." (Dr. Seuss) I teach the smallest students with the biggest enthusiasm for learning. My students learn in many different ways using all of our senses and multiple intelligences. I use a wide range of techniques to help all my students succeed. Students in my class come from a variety of different backgrounds which makes for wonderful sharing of experiences and cultures, including Native Americans. Our school is a caring community of successful learners which can be seen through collaborative student project based learning in and out of the classroom. Kindergarteners in my class love to work with hands-on materials and have many different opportunities to practice a skill before it is mastered. Having the social skills to work cooperatively with friends is a crucial aspect of the kindergarten curriculum. Montana is the perfect place to learn about agriculture and nutrition. My students love to role play in our pretend kitchen in the early childhood classroom. I have had several kids ask me, "Can we try cooking with REAL food?" I will take their idea and create "Common Core Cooking Lessons" where we learn important math and writing concepts while cooking delicious healthy food for snack time. My students will have a grounded appreciation for the work that went into making the food and knowledge of where the ingredients came from as well as how it's healthy for their bodies. This project would expand our learning of nutrition and agricultural cooking recipes by having us peel our own apples to make homemade applesauce, make our own bread, and mix up healthy plants from our classroom garden in the spring. We will also create our own cookbooks to be printed and shared with families. Students will gain math and literature skills as well as a life long enjoyment for healthy cooking. nanan

=====

My classroom consists of twenty-two amazing sixth graders from different cultures and backgrounds. They are a social bunch who enjoy working in partners and working with groups. They are hard-working and eager to head to middle school next year. My job is to get them ready to make this transition and make it as smooth as possible. In order to do this, my students need to come to school every day and feel safe and ready to learn. Because they are getting ready to head to middle school, I give them lots of choice- choice on where to sit and work, the order to complete assignments, choice of projects, etc. Part of the students feeling safe is the ability for them to come into a welcoming, encouraging environment. My room is colorful and the atmosphere is casual. I want them to take ownership of the classroom because we ALL share it together. Because my time with them is limited, I want to ensure they get the most of this time and enjoy it to the best of their abilities. Currently, we have twenty-two desks of differing sizes, yet

the desks are similar to the ones the students will use in middle school. We also have a kidney table with crates for seating. I allow my students to choose their own spots while they are working independently or in groups. More often than not, most of them move out of their desks and onto the crates. Believe it or not, this has proven to be more successful than making them stay at their desks! It is because of this that I am looking toward the "Flexible Seating" option for my classroom.\r\n The students look forward to their work time so they can move around the room. I would like to get rid of the constricting desks and move toward more "fun" seating options. I am requesting various seating so my students have more options to sit. Currently, I have a stool and a papasan chair I inherited from the previous sixth-grade teacher as well as five milk crate seats I made, but I would like to give them more options and reduce the competition for the "good seats". I am also requesting two rugs as not only more seating options but to make the classroom more welcoming and appealing. In order for my students to be able to write and complete work without desks, I am requesting a class set of clipboards. Finally, due to curriculum that requires groups to work together, I am requesting tables that we can fold up when we are not using them to leave more room for our flexible seating options.\r\nI know that with more seating options, they will be that much more excited about coming to school! Thank you for your support in making my classroom one students will remember forever!nannan

=====

In [12]:

```
# 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 [13]:

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

\nA person is a person, no matter how small.\n (Dr.Seuss) I teach the smallest students with the biggest enthusiasm for learning. My students learn in many different ways using all of our senses and multiple intelligences. I use a wide range of techniques to help all my students succeed. \n\nStudents in my class come from a variety of different backgrounds which makes for wonderful sharing of experiences and cultures, including Native Americans.\n\nOur school is a caring community of successful learners which can be seen through collaborative student project based learning in and out of the classroom. Kindergarteners in my class love to work with hands-on materials and have many different opportunities to practice a skill before it is mastered. Having the social skills to work cooperatively with friends is a crucial aspect of the kindergarten curriculum.Montana is the perfect place to learn about agriculture and nutrition. My students love to role play in our pretend kitchen in the early childhood classroom. I have had several kids ask me, \n\n"Can we try cooking with REAL food?" I will take their idea and create \n\n"Common Core Cooking Lessons" where we learn important math and writing concepts while cooking delicious healthy food for snack time. My students will have a grounded appreciation for the work that went into making the food and knowledge of where the ingredients came from as well as how it is healthy for their bodies. This project would expand our learning of nutrition and agricultural cooking recipes by having us peel our own apples to make homemade applesauce, make our own bread, and mix up healthy plants from our classroom garden in the spring. We will also create our own cookbooks to be printed and shared with families. \n\nStudents will gain math and literature skills as well as a life long enjoyment for healthy cooking.nannan

=====

In [14]:

```
# \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)
```

A person is a person, no matter how small. (Dr.Seuss) I teach the smallest students with the biggest enthusiasm for learning. My students learn in many different ways using all of our senses and multiple intelligences. I use a wide range of techniques to help all my students succeed. Students in my class come from a variety of different backgrounds which makes for wonderful sharing of experiences and cultures, including Native Americans. Our school is a caring community of successful learners which can be seen through collaborative student project based learning in and out of the classroom. Kindergarteners in my class love to work with hands-on materials and have many different opportunities to practice a skill before it is mastered. Having the social skills to work cooperatively with friends is a crucial aspect of the kindergarten curriculum. Montana is the perfect place to learn about agriculture and nutrition. My students love to role play in our pretend kitchen in the early childhood classroom. I have had several kids ask me, Can we try cooking with REAL food? I will take their idea and create Common Core Cooking Lessons where we learn important math and writing concepts while cooking delicious healthy food for snack time. My students will have a grounded appreciation for the work that went into making the food and knowledge of where the ingredients came from as well as how it is healthy for their bodies. This project would expand our learning of nutrition and agricultural cooking recipes by having us peel our own apples to make homemade applesauce, make our own bread, and mix up healthy plants from our classroom garden in the spring. We will also create our own cookbooks to be printed and shared with families. Students will gain math and literature skills as well as a life long enjoyment for healthy cooking.

nan

In [15]:

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

A person is a person no matter how small Dr Seuss I teach the smallest students with the biggest enthusiasm for learning My students learn in many different ways using all of our senses and multiple intelligences I use a wide range of techniques to help all my students succeed Students in my class come from a variety of different backgrounds which makes for wonderful sharing of experiences and cultures including Native Americans Our school is a caring community of successful learners which can be seen through collaborative student project based learning in and out of the classroom Kindergarten teachers in my class love to work with hands on materials and have many different opportunities to practice a skill before it is mastered Having the social skills to work cooperatively with friends is a crucial aspect of the kindergarten curriculum Montana is the perfect place to learn about agriculture and nutrition My students love to role play in our pretend kitchen in the early childhood classroom I have had several kids ask me Can we try cooking with REAL food I will take their idea and create Common Core Cooking Lessons where we learn important math and writing concepts while cooking delicious healthy food for snack time My students will have a grounded appreciation for the work that went into making the food and knowledge of where the ingredients came from as well as how it is healthy for their bodies This project would expand our learning of nutrition and agricultural cooking recipes by having us peel our own apples to make homemade applesauce make our own bread and mix up healthy plants from our classroom garden in the spring We will also create our own cookbooks to be printed and shared with families Students will gain math and literature skills as well as a life long enjoyment for healthy cooking nannan

In [16]:

```
# 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'r
e", "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', 't
hey', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
at'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
d', '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', 'ov
er', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
y', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too'
, 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
w', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'migh
tn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
asn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [17]:

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

100%|██████████| 109248/109248 [01:22<00:00, 1317.17it/s]

In [18]:

```
# after preprocessing
preprocessed_essays[20000]
```

Out[18]:

```
'person person no matter small dr seuss teach smallest students biggest en
thusiasm learning students learn many different ways using senses multiple
intelligences use wide range techniques help students succeed students cla
ss come variety different backgrounds makes wonderful sharing experiences
cultures including native americans school caring community successful lea
rners seen collaborative student project based learning classroom kinderga
rteners class love work hands materials many different opportunities pract
ice skill mastered social skills work cooperatively friends crucial aspect
kindergarten curriculum montana perfect place learn agriculture nutrition
students love role play pretend kitchen early childhood classroom several
kids ask try cooking real food take idea create common core cooking lesson
s learn important math writing concepts cooking delicious healthy food sna
ck time students grounded appreciation work went making food knowledge ing
redients came well healthy bodies project would expand learning nutrition
agricultural cooking recipes us peel apples make homemade applesauce make
bread mix healthy plants classroom garden spring also create cookbooks pri
nted shared families students gain math literature skills well life long e
njoyment healthy cooking nannan'
```

1.4 Preprocessing of `project_title`

In [19]:

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

In [20]:

```
pt = project_data['project_title']
```

In [21]:

```
pt.head()
```

Out[21]:

```
55660      Engineering STEAM into the Primary Classroom
76127                        Sensory Tools for Focus
51140      Mobile Learning with a Mobile Listening Center
473                Flexible Seating for Flexible Learning
41558                Going Deep: The Art of Inner Thinking!
Name: project_title, dtype: object
```

In [22]:

```
pt.nunique()
```

Out[22]:

```
100851
```

In [23]:

```
pt.values[100]
```

Out[23]:

```
'iCan with iPads...and YOU!'
```

In [24]:

```
# 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 [25]:

```
sent = decontracted(pt.values[2000])
print(sent)
print("="*50)
```

Empowering Students through Art in the Makerspace
=====

In [26]:

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

Empowering Students through Art in the Makerspace

In [27]:

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

Empowering Students through Art in the Makerspace

In [28]:

```
# 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'r
e", "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', 't
hey', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
at'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
d', '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', 'ov
er', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
y', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too'
, 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
w', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'migh
tn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
asn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [29]:

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

100%|██████████| 109248/109248 [00:03<00:00, 28275.65it/s]

In [30]:

```
preprocessed_titles[2000:2010]
```

Out[30]:

```
['empowering students art makerspace',  
'tablet astic',  
'election fall 2016 materials',  
'whole brain learning lounge',  
'calculators help us fractions algebra geometry more',  
'just basics',  
'alternative seating guru need rugs thanksteach',  
'capture experiences',  
'breakout ordinary',  
'21st century classroom makeover']
```

In []:

1.5 Preparing data for models

In [31]:

```
project_data.columns
```

Out[31]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',  
      'Date', 'project_grade_category', 'project_title', 'project_essay_1',  
      'project_essay_2', 'project_essay_3', 'project_essay_4',  
      'project_resource_summary',  
      'teacher_number_of_previously_posted_projects', 'project_is_approved',  
      'clean_categories', 'clean_subcategories', 'essay'],  
      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 (optional)
- quantity : numerical (optional)
- teacher_number_of_previously_posted_projects : numerical
- price : numerical

In [32]:

```
project_data.columns
```

Out[32]:

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

1.5.1 Vectorizing Categorical data

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

In [33]:

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

Out[33]:

```
'\n# we use count vectorizer to convert the values into one \nfrom sklearn  
n.feature_extraction.text import CountVectorizer\nvectorizer = CountVector  
izer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=Tru  
e)\ncategories_one_hot = vectorizer.fit_transform(project_data['clean_cat  
egories'].values)\nprint(vectorizer.get_feature_names())\nprint("Shape of  
matrix after one hot encoding ",categories_one_hot.shape)\n'
```

In [34]:

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

'''
```

Out[34]:

```
'\n# we use count vectorizer to convert the values into one \nvectorizer =
CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=True)\nsub_categories_one_hot = vectorizer.fit_transform(project_data['clean_subcategories'].values)\nprint(vectorizer.get_feature_names())\nprint("Shape of matrix after one hot encoding ", sub_categories_one_hot.shape)\n\n'
```

In [35]:

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

school state

In [36]:

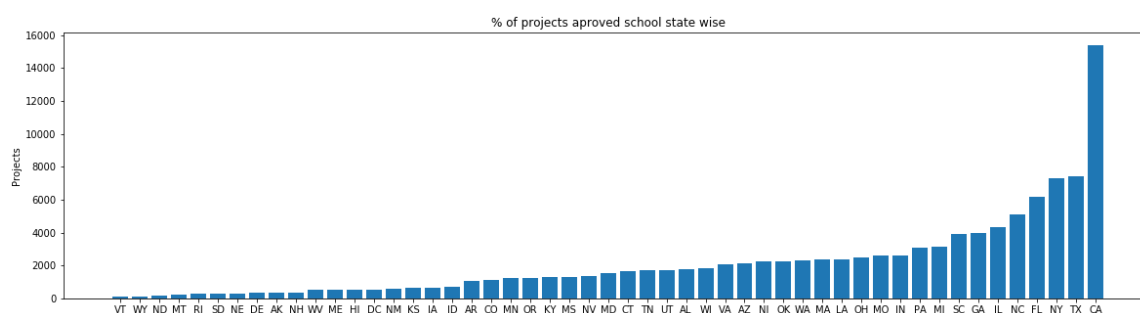
```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
from collections import Counter
my_counter = Counter()
for word in project_data['school_state'].values:
    my_counter.update(word.split())
```

In [37]:

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

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

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



In [38]:

```
'''
# we use count vectorizer to convert the values into one hot encoded features
vectorizer = CountVectorizer(vocabulary=list(sorted_scl_dict.keys()), lowercase=False,
    binary=True)
vectorizer.fit(project_data['school_state'].values)
print(vectorizer.get_feature_names())

sub_categories_one_hot_1 = vectorizer.transform(project_data['school_state'].values)
print("Shape of matrix after one hot encodig ",sub_categories_one_hot_1.shape)
'''
```

Out[38]:

```
'\n# we use count vectorizer to convert the values into one hot encoded fe
atures\nvectorizer = CountVectorizer(vocabulary=list(sorted_scl_dict.keys
()), lowercase=False, binary=True)\nvectorizer.fit(project_data['school_s
tate\n'].values)\nprint(vectorizer.get_feature_names())\n\nsub_categories
_one_hot_1 = vectorizer.transform(project_data['school_state\n'].values)\n
print("Shape of matrix after one hot encodig ",sub_categories_one_hot_1.sh
ape)\n\n'
```

teacher prefix

In [39]:

```
project_data.groupby(['teacher_prefix'])['teacher_prefix'].count()
```

Out[39]:

```
teacher_prefix
Dr.           13
Mr.          10648
Mrs.          57269
Ms.           38955
Teacher       2360
Name: teacher_prefix, dtype: int64
```

In [40]:

```
project_data['teacher_prefix'][project_data['teacher_prefix'].isnull()==True]
```

Out[40]:

```
30368    NaN
57654    NaN
7820     NaN
Name: teacher_prefix, dtype: object
```

In [41]:

```
project_data['teacher_prefix'].fillna(project_data['teacher_prefix'].mode()[0],inplace=True)
```

In [42]:

```
project_data['teacher_prefix'][project_data['teacher_prefix'].isnull()==True]
```

Out[42]:

```
Series([], Name: teacher_prefix, dtype: object)
```

In [43]:

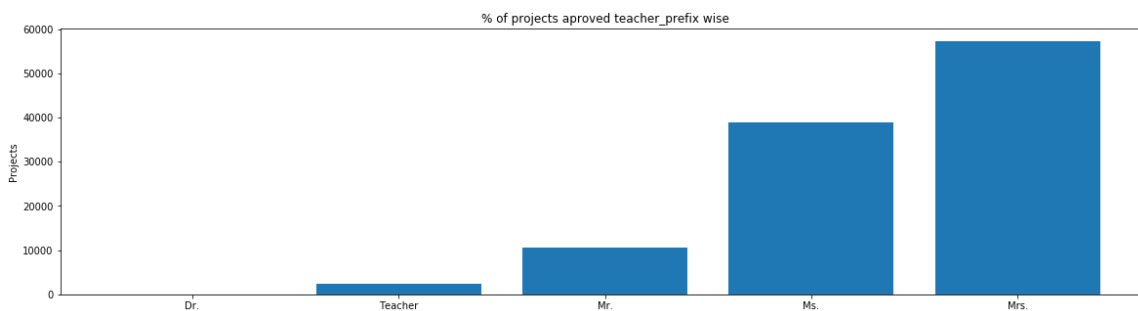
```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
from collections import Counter
my_counter = Counter()
for word in project_data['teacher_prefix'].values:
    my_counter.update(word.split())
```


In [44]:

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

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

plt.ylabel('Projects')
plt.title('% of projects aproved teacher_prefix wise')
plt.xticks(ind, list(sorted_tp_dict.keys()))
plt.show()
```



In [45]:

```
'''
# we use count vectorizer to convert the values into one hot encoded features
vectorizer = CountVectorizer(vocabulary=list(sorted_tp_dict.keys()), lowercase=False, binary=True)
vectorizer.fit(project_data['teacher_prefix'].values)
print(vectorizer.get_feature_names())

sub_categories_one_hot_2 = vectorizer.transform(project_data['teacher_prefix'].values)
print("Shape of matrix after one hot encodig ", sub_categories_one_hot_2.shape)

'''
```

Out[45]:

```
'\n# we use count vectorizer to convert the values into one hot encoded fe
atures\nvectorizer = CountVectorizer(vocabulary=list(sorted_tp_dict.keys
()), lowercase=False, binary=True)\nvectorizer.fit(project_data[\'teacher_
prefix\'].values)\nprint(vectorizer.get_feature_names())\n\n\nsub_categori
es_one_hot_2 = vectorizer.transform(project_data[\'teacher_prefix\'].value
s)\nprint("Shape of matrix after one hot encodig ", sub_categories_one_hot_
2.shape)\n\n'
```

project grade category

In [46]:

```
project_data['project_grade_category'][project_data['project_grade_category'].isnull()=
=True]
```

Out[46]:

Series([], Name: project_grade_category, dtype: object)

In [47]:

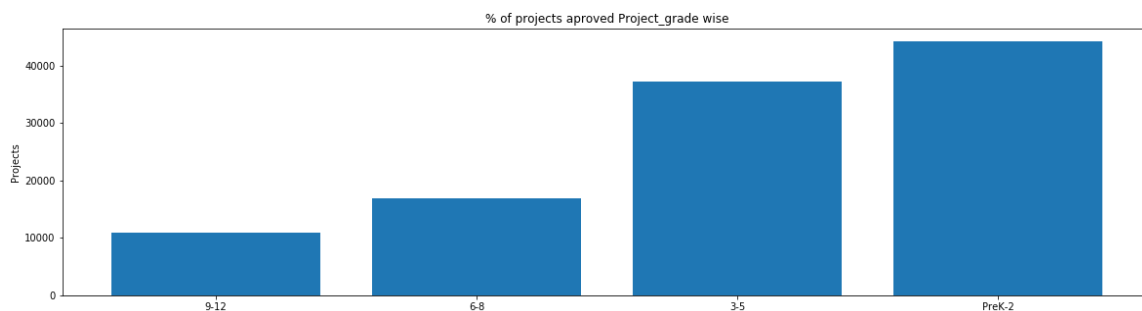
```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
from collections import Counter
my_counter = Counter()
for word in project_data['project_grade_category'].values:
    my_counter.update(word.split())
```

In [48]:

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

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

plt.ylabel('Projects')
plt.title('% of projects aproved Project_grade wise')
plt.xticks(ind, list(sorted_tp_dict.keys()))
plt.show()
```



In [49]:

```
'''
# we use count vectorizer to convert the values into one hot encoded features
vectorizer = CountVectorizer(vocabulary=list(sorted_tp_dict.keys()), lowercase=False, binary=True)
vectorizer.fit(project_data['project_grade_category'].values)
print(vectorizer.get_feature_names())

sub_categories_one_hot_3 = vectorizer.transform(project_data['project_grade_category'].values)
print("Shape of matrix after one hot encoding ", sub_categories_one_hot_3.shape)

'''
```

Out[49]:

```
'\n# we use count vectorizer to convert the values into one hot encoded features\nvectorizer = CountVectorizer(vocabulary=list(sorted_tp_dict.keys()), lowercase=False, binary=True)\nvectorizer.fit(project_data['project_grade_category'].values)\nprint(vectorizer.get_feature_names())\n\nsub_categories_one_hot_3 = vectorizer.transform(project_data['project_grade_category'].values)\nprint("Shape of matrix after one hot encoding ", sub_categories_one_hot_3.shape)\n'
```

In []:

In []:

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [50]:

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

'''
```

Out[50]:

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

In [51]:

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

vectorizer1 = CountVectorizer()
text_bow1 = vectorizer1.fit_transform(preprocessed_titles)
print("Shape of matrix after one hot encoding ",text_bow1.shape)

'''
```

Out[51]:

```
'\n# you can vectorize the title also \n# before you vectorize the title make sure you preprocess it\n\nvectorizer1 = CountVectorizer()\ntext_bow1 = vectorizer1.fit_transform(preprocessed_titles)\nprint("Shape of matrix after one hot encoding ",text_bow1.shape)\n\n'
```

1.5.2.2 TFIDF vectorizer

In [52]:

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

'''
```

Out[52]:

```
'\nfrom sklearn.feature_extraction.text import TfidfVectorizer\nvectorizer = TfidfVectorizer(min_df=10)\ntext_tfidf = vectorizer.fit_transform(preprocessed_essays)\nprint("Shape of matrix after one hot encoding ",text_tfidf.shape)\n\n'
```

In [53]:

```
'''
# Similarly you can vectorize for title also
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer2 = TfidfVectorizer()
text_tfidf2 = vectorizer2.fit_transform(preprocessed_titles)
print("Shape of matrix after one hot encoding ",text_tfidf2.shape)

'''
```

Out[53]:

```
'\n# Similarly you can vectorize for title also\nfrom sklearn.feature_extraction.text import TfidfVectorizer\nvectorizer2 = TfidfVectorizer()\ntext_tfidf2 = vectorizer2.fit_transform(preprocessed_titles)\nprint("Shape of matrix after one hot encoding ",text_tfidf2.shape)\n\n'
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [54]:

```
'''
# 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-save-and-load-variables-in-python/

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

'''
```

Out[54]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/3823034
9/4084039\ndef loadGloveModel(gloveFile):\n    print ("Loading Glove Mode
l")\n    f = open(gloveFile,\r', encoding="utf8")\n    model = {}\n    f
or line in tqdm(f):\n        splitLine = line.split()\n        word = spli
tLine[0]\n        embedding = np.array([float(val) for val in splitLine
[1:]])\n        model[word] = embedding\n    print ("Done.",len(model)," w
ords loaded!")\n    return model\nmodel = loadGloveModel('glove.42B.300d.
txt')\n\n# =====\nOutput:\n    \nLoading Glove Mod
el\n1917495it [06:32, 4879.69it/s]\nDone. 1917495 words loaded!\n\n# ===
=====
\n\nwords = []\nfor i in preproced_texts:\n    wor
ds.extend(i.split(' '))\n\nfor i in preproced_titles:\n    words.extend
(i.split(' '))\n\nprint("all the words in the coupus", len(words))\n\nwords
= set(words)\n\nprint("the unique words in the coupus", len(words))\n\ninter
_words = set(model.keys()).intersection(words)\n\nprint("The number of words
that are present in both glove vectors and our coupus", len(inter_wo
rds), "("np.round(len(inter_words)/len(words)*100,3),"%")\n\nwords_courpu
s = {}\n\nwords_glove = set(model.keys())\n\nfor i in words:\n    if i in word
s_glove:\n        words_courpus[i] = model[i]\n\nprint("word 2 vec length",
len(words_courpus))\n\n\n# stronging variables into pickle files python: h
ttp://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-
python/\n\nimport pickle \n\nwith open('glove_vectors', 'wb') as f:\n
pickle.dump(words_courpus, f)\n\n\n'
```

In [55]:

```
'''
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-p
ickle-to-save-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())
'''
```

Out[55]:

```
"\n# stronging variables into pickle files python: http://www.jessicayung.
com/how-to-use-pickle-to-save-and-load-variables-in-python/\n# make sure y
ou have the glove_vectors file\n\nwith open('../glove_vectors', 'rb') as
f:\n    model = pickle.load(f)\n    glove_words = set(model.keys())\n\n
\n"
```

In [56]:

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

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

'''
```

Out[56]:

```
'\n# average Word2Vec\n# compute average word2vec for each review.\navg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list\nfor sentence in tqdm(preprocessed_essays): # for each review/sentence\n    vector = np.zeros(300) # as word vectors are of zero length\n    cnt_words = 0; # num of words with a valid vector in the sentence/review\n    for word in sentence.split(): # for each word in a review/sentence\n        if word in glove_words:\n            vector += model[word]\n            cnt_words += 1\n    if cnt_words != 0:\n        vector /= cnt_words\n    avg_w2v_vectors.append(vector)\n\nprint(len(avg_w2v_vectors))\nprint(len(avg_w2v_vectors[0]))\n'
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [57]:

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

'''
```

Out[57]:

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

In [58]:

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

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

Out[58]:

```
'\n# average Word2Vec\n# compute average word2vec for each review.\nntfidf_
w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this
list\nfor sentence in tqdm(preprocessed_essays): # for each review/sentenc
e\n    vector = np.zeros(300) # as word vectors are of zero length\n    tf
_idf_weight = 0; # num of words with a valid vector in the sentence/review
\n    for word in sentence.split(): # for each word in a review/sentence\n
if (word in glove_words) and (word in tfidf_words):\n        vec = mod
el[word] # getting the vector for each word\n        # here we are mul
tiplying idf value(dictionary[word]) and the tf value((sentence.count(wor
d)/len(sentence.split())))\n        tf_idf = dictionary[word]*(sentenc
e.count(word)/len(sentence.split())) # getting the tfidf value for each wo
rd\n        vector += (vec * tf_idf) # calculating tfidf weighted w2v
\n        tf_idf_weight += tf_idf\n    if tf_idf_weight != 0:\n
vector /= tf_idf_weight\n    tfidf_w2v_vectors.append(vector)\n\nprint(len
(tfidf_w2v_vectors))\nprint(len(tfidf_w2v_vectors[0]))\n\n'
```

In [59]:

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

1.5.3 Vectorizing Numerical features

In [60]:

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


In [61]:

```
'''
# check this one: https://www.youtube.com/watch?v=0H0qOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler

# price_standardized = standardScaler.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329. ...
399. 287.73 5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

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

'''
```

Out[61]:

```
'\n# check this one: https://www.youtube.com/watch?v=0H0qOcln3Z4&t=530s\n# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html\nfrom sklearn.preprocessing import StandardScaler\n\n# price_standardized = standardScaler.fit(project_data['price'].values)\n# this will rise the error\n# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329. ... 399. 287.73 5.5 ].\n# Reshape your data either using array.reshape(-1, 1)\n\nprice_scalar = StandardScaler()\nprice_scalar.fit(project_data['price'].values.reshape(-1,1)) # finding the mean and standard deviation of this data\n\nprint(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")\n\n# Now standardize the data with above mean and variance.\nprice_standardized = price_scalar.transform(project_data['price'].values.reshape(-1, 1))\n\n'
```

In [62]:

```
# price_standardized
```

1.5.4 Merging all the above features

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

In [63]:

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

Out[63]:

```
'\nprint(categories_one_hot.shape)\nprint(sub_categories_one_hot.shape)\nprint(text_bow.shape)\nprint(price_standardized.shape)\n'
```

In [64]:

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

Out[64]:

```
'\n# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
\nfrom scipy.sparse import hstack\n# with the same hstack function we are concatenating a sparse matrix and a dense matrix :)\nX = hstack((categories_one_hot, sub_categories_one_hot, text_bow, price_standardized))\nX.shape\n'
```

Assignment 3: Apply KNN



1. [Task-1] Apply KNN(brute force version) on these feature sets

- **Set 1**: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)
- **Set 2**: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)
- **Set 3**: categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)
- **Set 4**: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

2. Hyper paramter tuning to find best K

- Find the best hyper parameter which results in the maximum [AUC](https://www.applidaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) (<https://www.applidaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/>) value
- Find the best hyper paramter using k-fold cross validation (or) simple cross validation data
- Use gridsearch-cv or randomsearch-cv or 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, as shown in the figure

- Once you find the best hyper parameter, you need to train your model-M using the best hyper-param. Now, find the AUC on test data and plot the ROC curve on both train and test using model-M.

- Along with plotting ROC curve, you need to print the [confusion matrix](https://www.applidaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tp-r-fpr-fnr-tnr-1/) (<https://www.applidaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tp-r-fpr-fnr-tnr-1/>) with predicted and original labels of test data points



4. [Task-2]

- Select top 2000 features from feature **Set 2** using `'SelectKBest'` (https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html) and then apply KNN on top of these features

- ```
from sklearn.datasets import load_digits
from sklearn.feature_selection import SelectKBest, chi2
X, y = load_digits(return_X_y=True)
X.shape
X_new = SelectKBest(chi2, k=20).fit_transform(X, y)
X_new.shape
=====
output:
(1797, 64)
(1797, 20)
```

- Repeat the steps 2 and 3 on the data matrix after feature selection

## 5. 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](http://zetcode.com/python/prettytable/) (<http://zetcode.com/python/prettytable/>)



### 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. \(https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf\)](https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf)

## 2. K Nearest Neighbor

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

In [65]:

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

In [66]:

```
project_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 109248 entries, 0 to 109247
Data columns (total 20 columns):
 # Column Non-Null Count Dtype
--- -
 0 Unnamed: 0 109248 non-null int64
 1 id 109248 non-null object
 2 teacher_id 109248 non-null object
 3 teacher_prefix 109248 non-null object
 4 school_state 109248 non-null object
 5 Date 109248 non-null datetime64[ns]
 6 project_grade_category 109248 non-null object
 7 project_title 109248 non-null object
 8 project_essay_1 109248 non-null object
 9 project_essay_2 109248 non-null object
10 project_essay_3 3758 non-null object
11 project_essay_4 3758 non-null object
12 project_resource_summary 109248 non-null object
13 teacher_number_of_previously_posted_projects 109248 non-null int64
14 project_is_approved 109248 non-null int64
15 clean_categories 109248 non-null object
16 clean_subcategories 109248 non-null object
17 essay 109248 non-null object
18 price 109248 non-null float64
19 quantity 109248 non-null int64
dtypes: datetime64[ns](1), float64(1), int64(4), object(14)
memory usage: 17.5+ MB
```

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

## Dropping the unnecessary columns

In [67]:

```
data1 = project_data.drop(['Unnamed: 0', 'id', 'Date', 'project_essay_1', 'project_essay_2', 'project_essay_3', 'project_essay_4', 'project_resource_summary', 'teacher_id', 'quantity'], axis = 1)
```

In [68]:

```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 109248 entries, 0 to 109247
Data columns (total 10 columns):
Column Non-Null Count Dtype
--- -
0 teacher_prefix 109248 non-null object
1 school_state 109248 non-null object
2 project_grade_category 109248 non-null object
3 project_title 109248 non-null object
4 teacher_number_of_previously_posted_projects 109248 non-null int64
5 project_is_approved 109248 non-null int64
6 clean_categories 109248 non-null object
7 clean_subcategories 109248 non-null object
8 essay 109248 non-null object
9 price 109248 non-null float64
dtypes: float64(1), int64(2), object(7)
memory usage: 9.2+ MB
```

In [69]:

```
data1 = data1[:50000]
```

In [ ]:

In [70]:

```
y = data1['project_is_approved'].values
X = data1.drop(['project_is_approved'], axis=1)
X.head(1)
```

Out[70]:

|   | teacher_prefix | school_state | project_grade_category | project_title                                | teacher_number_of_previ |
|---|----------------|--------------|------------------------|----------------------------------------------|-------------------------|
| 0 | Mrs.           | CA           | Grades PreK-2          | Engineering STEAM into the Primary Classroom |                         |

In [71]:

```
train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

In [ ]:

## 2.2 Make Data Model Ready: encoding numerical, categorical features

In [72]:

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

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

### 2.2.1 Numerical features

1. teacher\_number\_of\_previously\_posted\_projects
2. price

### 2.2.1.1 Teacher number of previously posted projects

In [73]:

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

X_train_TPPP_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1, -1))
X_cv_TPPP_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(1, -1))
X_test_TPPP_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(1, -1))

print("After vectorizations")
print(X_train_TPPP_norm.shape, y_train.shape)
print(X_cv_TPPP_norm.shape, y_cv.shape)
print(X_test_TPPP_norm.shape, y_test.shape)
print("=*100)
```

After vectorizations

```
(1, 22445) (22445,)
(1, 11055) (11055,)
(1, 16500) (16500,)
```

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



In [74]:

```
print("Transpose of teacher number of previously posted projects")

X_train_TPPP_norm = X_train_TPPP_norm.transpose()
X_cv_TPPP_norm = X_cv_TPPP_norm.transpose()
X_test_TPPP_norm = X_test_TPPP_norm.transpose()

print("After transpose")
print(X_train_TPPP_norm.shape, y_train.shape)
print(X_cv_TPPP_norm.shape, y_cv.shape)
print(X_test_TPPP_norm.shape, y_test.shape)
print("=*100)
```

Transpose of teacher number of previously posted projects

After transpose

(22445, 1) (22445,)

(11055, 1) (11055,)

(16500, 1) (16500,)

=====

### 2.2.1.2 price

In [75]:

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

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(1, -1))
X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1, -1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1, -1))

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
print("=*100)
```

After vectorizations

(1, 22445) (22445,)

(1, 11055) (11055,)

(1, 16500) (16500,)

=====

In [76]:

```
print("Transpose of price")

X_train_price_norm = X_train_price_norm.transpose()
X_cv_price_norm = X_cv_price_norm.transpose()
X_test_price_norm = X_test_price_norm.transpose()

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

Transpose of price

After vectorizations

(22445, 1) (22445,)

(11055, 1) (11055,)

(16500, 1) (16500,)

=====

## 2.2.2 Categorical Data

### Categorical Features for vectorization

1. Clean Categories
2. Clean Sub Categories
3. School State
4. Teacher Prefix
5. Project grade category

#### 2.2.2.1 Clean Categories

In [77]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on train data

we use the fitted CountVectorizer to convert the text to vector
X_train_CC_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_cv_CC_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_CC_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_CC_ohe.shape, y_train.shape)
print(X_cv_CC_ohe.shape, y_cv.shape)
print(X_test_CC_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 7) (22445,)
(11055, 7) (11055,)
(16500, 7) (16500,)
['appliedlearning', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'music_arts', 'specialneeds']
=====
=====
```

### 2.2.2.2 Clean Sub Categories

In [78]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only on train
data

we use the fitted CountVectorizer to convert the text to vector
X_train_CSC_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_CSC_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_CSC_ohe = vectorizer.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_CSC_ohe.shape, y_train.shape)
print(X_cv_CSC_ohe.shape, y_cv.shape)
print(X_test_CSC_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 28) (22445,)
(11055, 28) (11055,)
(16500, 28) (16500,)
['appliedsciences', 'charactereducation', 'civics_government', 'college_ca
reerprep', 'communityservice', 'earlydevelopment', 'economics', 'environme
ntalscience', 'esl', 'extracurricular', 'financialliteracy', 'foreignlangu
ages', 'gym_fitness', 'health_lifescience', 'health_wellness', 'history_ge
ography', 'literacy', 'literature_writing', 'mathematics', 'music', 'nutri
tioneducation', 'other', 'parentinvolvement', 'performingarts', 'socialsci
ences', 'specialneeds', 'teamsports', 'visualarts']
=====
=====
```

### 2.2.2.3 School State

In [79]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train data

we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 51) (22445,)
(11055, 51) (11055,)
(16500, 51) (16500,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi',
'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'm
o', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'o
k', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'w
i', 'wv', 'wy']
=====
=====
```

#### 2.2.2.4 Teacher prefix

In [80]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data

we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

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

```
After vectorizations
(22445, 5) (22445,)
(11055, 5) (11055,)
(16500, 5) (16500,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
=====
=====
```

#### 2.2.2.5 Project Grade category

In [81]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only on train data

we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

After vectorizations

(22445, 3) (22445,)

(11055, 3) (11055,)

(16500, 3) (16500,)

['12', 'grades', 'prek']

=====

In [ ]:

## 2.3 Make Data Model Ready: encoding eassay, and project\_title

In [82]:

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

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

### Ecoding Essay and Project title

2.3.1 BOW

2.3.2 TFIDF

2.3.3 AVG W2V

2.3.4 TFIDF W2V

## 2.3.1 BOW Essays and Title

### 2.3.1.1 BOW Essay

In [83]:

```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)

vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data

we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['essay'].values)
X_cv_essay_bow = vectorizer.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
print("="*100)
```

```
(22445, 9) (22445,)
(11055, 9) (11055,)
(16500, 9) (16500,)
=====
=====
After vectorizations
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
=====
=====
```

### 2.3.1.2 BOW Title

In [84]:

```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)

vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['project_title'].values) # fit has to happen only on train data

we use the fitted CountVectorizer to convert the text to vector
X_train_title_bow = vectorizer.transform(X_train['project_title'].values)
X_cv_title_bow = vectorizer.transform(X_cv['project_title'].values)
X_test_title_bow = vectorizer.transform(X_test['project_title'].values)

print("After vectorizations")
print(X_train_title_bow.shape, y_train.shape)
print(X_cv_title_bow.shape, y_cv.shape)
print(X_test_title_bow.shape, y_test.shape)
print("="*100)
```

```
(22445, 9) (22445,)
(11055, 9) (11055,)
(16500, 9) (16500,)
=====
=====
After vectorizations
(22445, 2666) (22445,)
(11055, 2666) (11055,)
(16500, 2666) (16500,)
=====
=====
```

## 2.3.2 TF IDF Essay and Title

### 2.3.2.1 TF IDF Essay



In [85]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)

vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data

we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer.transform(X_train['essay'].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
```

```
(22445, 9) (22445,)
(11055, 9) (11055,)
(16500, 9) (16500,)
=====
=====
After vectorizations
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
=====
=====
```

### 2.3.2.2 TF IDF Title

In [86]:

```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)

vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['project_title'].values) # fit has to happen only on train data

we use the fitted CountVectorizer to convert the text to vector
X_train_title_tfidf = vectorizer.transform(X_train['project_title'].values)
X_cv_title_tfidf = vectorizer.transform(X_cv['project_title'].values)
X_test_title_tfidf = vectorizer.transform(X_test['project_title'].values)

print("After vectorizations")
print(X_train_title_tfidf.shape, y_train.shape)
print(X_cv_title_tfidf.shape, y_cv.shape)
print(X_test_title_tfidf.shape, y_test.shape)
print("="*100)
```

```
(22445, 9) (22445,)
(11055, 9) (11055,)
(16500, 9) (16500,)
=====
=====
After vectorizations
(22445, 2666) (22445,)
(11055, 2666) (11055,)
(16500, 2666) (16500,)
=====
=====
```

## 2.3.3 AVG W2V Essay and Title

### 2.3.3.1 AVG W2V Essay

In [87]:

```
stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-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 [88]:

```
average Word2Vec
compute average word2vec for each review.
avg_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay'].values): # for each review/sentence
 vector = np.zeros(300) # as word vectors are of zero length
 cnt_words = 0; # num of words with a valid vector in the sentence/review
 for word in sentence.split(): # for each word in a review/sentence
 if word in glove_words:
 vector += model[word]
 cnt_words += 1
 if cnt_words != 0:
 vector /= cnt_words
 avg_w2v_vectors_train.append(vector)

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

100%|██████████| 22445/22445 [00:11<00:00, 1908.00it/s]

22445

300

[ -4.85776307e-02 -2.58230962e-02 -7.53944162e-04 -1.69378661e-01  
7.84706000e-02 -1.69612741e-02 -3.49292150e+00 1.93313288e-01  
-7.14677970e-03 -2.47001681e-01 1.07479571e-01 -3.75582052e-02  
2.22784685e-02 -7.72303411e-02 -6.59297107e-02 -9.01666776e-02  
-1.54333902e-01 -8.64541898e-02 4.71888479e-02 5.13646625e-02  
1.32131472e-04 -3.66112371e-02 -4.25200218e-02 5.74573330e-02  
-2.78185965e-02 -7.17841787e-02 4.54613086e-02 -7.42326426e-02  
-6.31279342e-02 -1.21827438e-01 -1.84022829e-01 -8.35673380e-02  
-5.11453959e-03 -2.84179675e-02 -8.93574416e-02 5.92496904e-03  
-5.44107898e-02 -1.06382611e-01 3.26784325e-02 -7.93905581e-02  
-4.79277437e-02 1.15498693e-01 8.57021188e-03 -2.26968476e-01  
-3.12890545e-02 -4.28288396e-02 -7.12156294e-03 -1.43843081e-01  
-1.03849046e-01 -2.05373426e-02 4.76997320e-02 1.12454574e-02  
-1.94389066e-02 -2.32733660e-02 3.90112843e-02 -4.80976173e-02  
7.27092562e-02 -5.40428234e-02 -2.37792975e-02 8.70125299e-02  
1.79004518e-02 2.54425250e-02 4.63524061e-02 -4.86483670e-02  
-5.59435909e-02 1.07795478e-01 4.56960680e-02 5.74233391e-02  
1.86933998e-01 -9.26403447e-02 -1.92678627e-01 7.06190619e-02  
-7.54193208e-02 -6.44514537e-02 -3.04410553e-02 -2.43747811e-01  
1.23566563e-01 4.30909061e-02 5.92427756e-02 -1.12135081e-01  
3.23310520e-02 -4.04146514e-01 -1.03264947e-01 -1.39415078e-01  
-6.01370411e-02 3.97435416e-02 1.81897970e-03 -7.98860849e-02  
8.55221198e-02 -7.15692188e-02 6.25019746e-02 -6.05862518e-02  
-1.18936858e-02 1.20445297e-01 6.70109168e-02 -2.90348594e-01  
-2.46461315e+00 -6.26860772e-02 1.04223262e-01 4.66156949e-02  
-3.07609838e-02 4.62295234e-02 2.55536221e-01 3.05941015e-02  
-7.02161792e-02 7.98587457e-03 3.58539269e-02 -9.90718766e-02  
-8.03088437e-02 -3.79970751e-03 -2.28234870e-02 5.82914721e-02  
5.03502934e-02 1.64724281e-01 -9.98015223e-02 7.99105901e-02  
3.60073650e-03 -6.33948746e-02 9.61857411e-02 4.19582926e-02  
-9.45173838e-02 2.30400563e-02 7.49727278e-02 -1.73293409e-01  
2.00129691e-02 1.02397954e-02 8.70129442e-04 -3.11325609e-02  
3.36028959e-02 1.41679899e-01 -5.61175170e-02 6.77547096e-02  
-7.11987239e-02 -1.03369019e-01 3.32209533e-02 -3.44355888e-03  
1.26005977e-01 -3.09623350e-03 2.53085007e-01 3.39049175e-01  
1.17488694e-01 6.56853453e-02 6.81034105e-02 -1.11506449e-02  
8.96999543e-03 -4.50130680e-02 8.81892345e-02 -5.18273640e-02  
3.40415821e-01 1.80052697e-01 -1.22893462e-02 -3.09536269e-02  
1.52218721e-03 -3.50229132e-02 1.20119525e-01 -8.44707574e-02  
5.50328650e-02 2.91314079e-02 -8.86302873e-02 1.02525797e-02  
3.11148569e-02 -3.14305482e-02 -4.03037477e-02 -4.30107990e-02  
-5.37034183e-02 3.33924178e-02 -5.67308670e-02 6.21575447e-02  
1.35584370e-01 -5.43543416e-02 -5.36290543e-02 -1.02645563e-02  
-8.94422359e-02 -7.46430645e-02 -8.20295990e-02 1.90865906e-01  
-9.27164147e-02 -6.60849817e-02 -5.33868929e-02 -6.23285807e-02  
1.23888674e-01 9.48572898e-02 -8.60238051e-02 -3.96425208e-02  
6.95507020e-02 -1.92207134e-01 2.64079365e-02 -2.72430853e-02  
1.76447426e-01 -2.62953604e-03 -4.38578442e-02 -2.82051061e-02  
-6.70528152e-02 -1.25916369e-02 1.68471508e-02 -6.41953315e-02  
7.73497878e-02 7.33860543e-02 6.24150958e-02 -5.16268741e-03  
1.56765827e-01 -2.97526626e-02 1.03870787e-02 -1.21017179e-02  
-3.17168594e-02 1.15501717e-01 2.43880107e-02 -1.35462307e-01  
1.65384599e-01 -5.05549772e-02 1.30529007e-01 -1.36632528e-02  
-4.85056569e-02 -6.02856411e-02 -2.98935455e-02 -3.92663870e-02  
-7.20603000e-02 -8.90841521e-02 -8.79443948e-03 3.08010118e-02  
-3.87498010e-02 -1.21611136e-01 -6.63228376e-03 5.89015939e-03  
-2.50753962e+00 6.83948817e-02 -1.72556182e-02 -1.64832543e-02  
1.93990430e-02 -1.15967028e-01 1.24867722e-01 5.90383096e-02  
1.27453939e-02 -5.36535791e-02 -8.65697660e-02 9.66815723e-02

```
-7.35629797e-03 -8.25274025e-02 -9.46618487e-02 -3.16447208e-03
-1.07790948e-01 6.09004690e-02 -1.76005476e-01 5.14061208e-02
-2.26384173e-02 -4.33372452e-02 -7.89397190e-02 -3.92952635e-02
-3.87236255e-02 1.09429041e-02 2.65608579e-03 -6.81088543e-02
4.65151472e-03 2.41531088e-02 5.17310000e-02 -3.14483553e-03
2.37574470e-02 -2.72721954e-02 4.36156041e-02 -2.11985888e-03
2.55388711e-02 3.11481320e-03 -5.29350294e-02 -3.71500456e-02
5.80453419e-02 -9.21470634e-02 -2.17532675e-01 -6.70373442e-02
8.09473340e-02 -1.78640540e-02 -3.55105482e-03 -9.40096000e-02
-2.00124385e-01 3.79565299e-02 6.10459442e-03 1.01081826e-01
6.72770462e-02 7.21390589e-02 -9.08572391e-02 -3.30181371e-03
3.88115040e-01 3.20983812e-02 2.96353421e-02 1.32398990e-01
5.77234619e-04 7.75101108e-02 2.88278020e-02 3.41728514e-02
-5.38777843e-02 -4.00915416e-02 -1.16378797e-03 -1.14462574e-01
-7.51847335e-02 -6.54350959e-02 5.43805056e-02 2.86728203e-02
-8.06954442e-02 -7.72856701e-03 8.21209274e-02 7.02909640e-02]
```

In [89]:

```
avg_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essay'].values): # for each review/sentence
 vector = np.zeros(300) # as word vectors are of zero length
 cnt_words = 0; # num of words with a valid vector in the sentence/review
 for word in sentence.split(): # for each word in a review/sentence
 if word in glove_words:
 vector += model[word]
 cnt_words += 1
 if cnt_words != 0:
 vector /= cnt_words
 avg_w2v_vectors_cv.append(vector)
```

100%|██████████| 11055/11055 [00:05<00:00, 2027.49it/s]

In [90]:

```
avg_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay'].values): # for each review/sentence
 vector = np.zeros(300) # as word vectors are of zero length
 cnt_words = 0; # num of words with a valid vector in the sentence/review
 for word in sentence.split(): # for each word in a review/sentence
 if word in glove_words:
 vector += model[word]
 cnt_words += 1
 if cnt_words != 0:
 vector /= cnt_words
 avg_w2v_vectors_test.append(vector)
```

100%|██████████| 16500/16500 [00:08<00:00, 1930.13it/s]

### 2.3.3.2 AVG W2V Title

In [91]:

```
average Word2Vec
compute average word2vec for each review.
avg_w2v_vectors_train_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['project_title'].values): # for each review/sentence
 vector = np.zeros(300) # as word vectors are of zero length
 cnt_words = 0; # num of words with a valid vector in the sentence/review
 for word in sentence.split(): # for each word in a review/sentence
 if word in glove_words:
 vector += model[word]
 cnt_words += 1
 if cnt_words != 0:
 vector /= cnt_words
 avg_w2v_vectors_train_title.append(vector)

print(len(avg_w2v_vectors_train_title))
print(len(avg_w2v_vectors_train_title[0]))
print(avg_w2v_vectors_train_title[0])
```

100%|██████████| 22445/22445 [00:00<00:00, 109902.36it/s]



22445

300

[ -1.7445000e-02 -4.0600000e-02 -2.3054200e-01 -2.0636000e-01  
4.9566000e-01 1.7526500e-01 -3.2853500e+00 1.0163300e+00  
-4.9315000e-02 -6.7993000e-01 1.7552450e-01 -1.6035900e-01  
3.0383000e-01 -1.0966250e-01 -3.5600500e-01 -3.9602500e-01  
-1.8108000e-01 -9.3101000e-02 -2.4226050e-01 -2.4310600e-01  
2.8295000e-02 2.6846500e-01 1.1690100e-01 -7.9415000e-02  
3.4453000e-01 -1.9362250e-01 -1.2781500e-01 -1.3359000e-01  
-1.5712500e-01 -2.5175500e-01 2.5204000e-01 -5.2374500e-02  
3.8290000e-02 -1.7570050e-01 -2.2849300e-01 5.4228950e-02  
2.2043000e-01 2.5091500e-01 2.4661000e-01 -3.9345000e-02  
1.5187500e-01 4.1410000e-02 -2.2463500e-01 -3.7947500e-01  
2.9406800e-01 1.6366500e-01 -3.4094600e-01 -2.1824650e-01  
8.0560000e-02 -1.7700000e-01 -5.9000000e-02 4.5141700e-01  
-1.9508000e-01 -7.2844000e-02 9.4340000e-02 -9.8705000e-02  
7.3961500e-02 -1.3771000e-01 1.7232000e-01 1.7003650e-01  
-3.4094500e-01 -1.3454050e-01 9.7351000e-02 3.5108500e-01  
-2.5253000e-01 2.9122500e-01 4.6171500e-01 2.3577700e-01  
-1.1679050e-01 -1.5215350e-01 -5.4010000e-01 -1.2629500e-01  
3.7255000e-02 -1.3090000e-01 -3.7768350e-01 -4.6667000e-01  
3.7733000e-01 -4.5778000e-02 3.1217500e-01 -1.3090650e-01  
-9.3800000e-03 -1.6138000e-01 -2.5208450e-01 -1.2786500e-01  
-3.1085000e-02 -7.5780000e-02 -9.9687500e-02 -3.0410000e-01  
2.6051800e-01 -7.1346300e-02 -3.3539000e-02 -8.2350000e-02  
-4.8010000e-02 4.1386000e-01 3.0301000e-01 2.5300500e-01  
-2.5960000e+00 -2.5089950e-01 -2.7455500e-01 -1.0165450e-01  
4.2566000e-01 -9.5815000e-02 5.4935000e-01 3.0035000e-01  
1.1979800e-01 -1.5325000e-02 -2.1788500e-01 4.6037000e-01  
-1.8405000e-02 -8.1531500e-02 5.2134500e-02 7.6700000e-03  
-1.8039650e-01 3.2087000e-02 -3.8784000e-01 3.9807500e-01  
5.3293000e-01 -2.1762000e-01 2.2054000e-01 -5.9621500e-02  
-2.9960000e-02 -2.7159500e-01 1.3092900e-01 -1.0702275e-01  
1.7976600e-01 1.1242800e-01 -2.6995000e-02 -3.3844000e-01  
3.0048950e-01 -2.5390000e-02 1.1585000e-02 1.1868550e-01  
8.3624500e-02 2.2083000e-01 2.0575000e-02 -8.1508500e-02  
-2.1802000e-01 2.2227500e-01 3.6018500e-01 4.1130795e-01  
2.6910500e-02 4.5266000e-01 3.4812500e-01 -2.3872150e-01  
-3.3064900e-01 2.2834000e-01 1.0943000e-01 1.0466000e-01  
2.2261500e-01 -6.8548000e-02 -1.2015450e-01 4.2090000e-03  
-2.6745000e-02 -4.3771000e-01 3.5920000e-01 -2.1449500e-01  
2.7297500e-01 2.9853600e-01 -1.0565600e-01 2.4108550e-02  
2.2193500e-01 -6.1420000e-02 -1.8944000e-01 2.9180000e-03  
4.2698500e-02 -2.3534000e-01 -1.5584500e-01 -8.5770000e-02  
1.5222500e-01 4.5850000e-02 2.7581500e-01 4.1930000e-02  
1.4621350e-01 -3.1667100e-01 -4.5963000e-02 6.1303000e-01  
1.0295000e-01 -4.6940000e-01 -7.0495000e-02 1.7670000e-02  
-1.8351000e-01 -9.1366500e-02 -1.4228600e-01 3.2900500e-01  
2.5305500e-01 -2.5000000e-01 4.7655000e-01 -2.7587500e-01  
2.0777500e-01 1.0931660e-01 -1.5493700e-01 2.0685050e-01  
-4.3450000e-03 3.1720000e-02 4.6505500e-01 3.9665500e-01  
2.3951500e-01 1.5576000e-01 1.3373500e-01 -1.6525950e-01  
1.1872800e-01 9.3122000e-02 4.8105800e-01 -2.0339000e-02  
-2.1336430e-01 2.1088750e-01 -4.8270000e-02 2.0459500e-01  
2.7451000e-01 -9.9313000e-02 2.0332000e-01 -5.7873500e-02  
7.1081500e-02 3.0829800e-01 2.1120900e-01 7.7385000e-02  
4.0087000e-01 -2.3656600e-01 2.6917500e-01 -5.1873500e-01  
-3.8945000e-02 -2.0600000e-02 9.2815000e-02 3.5684000e-01  
-3.0072000e+00 -1.9009500e-01 7.3200000e-02 -2.9109000e-01  
-1.9088000e-01 -2.6200000e-02 2.7658000e-01 1.4905500e-01  
-9.8165500e-02 -6.1060500e-01 -2.5751550e-01 -3.5961000e-01

```

5.3560000e-02 -6.3646000e-02 1.4533300e-01 -3.0519000e-01
8.3037500e-02 -1.3386900e-01 -1.1170700e-01 -2.2059500e-01
2.4886500e-01 -2.9547500e-02 1.1955800e-01 6.7843500e-02
1.9020000e-02 -8.2700000e-02 1.6910500e-01 2.2299985e-01
-8.9961000e-02 1.7951000e-01 9.5965000e-02 2.2952500e-02
-6.3842500e-02 4.0634700e-01 7.7256000e-02 -2.5949500e-01
-6.7210000e-02 2.1812500e-01 -4.8661500e-01 5.6404000e-02
1.9198000e-02 -1.2083500e-01 -6.0969500e-01 -1.9371000e-01
4.9695000e-02 -2.5313100e-01 3.2322000e-01 -8.0949000e-02
-8.7181350e-02 1.5449200e-01 -2.3785500e-01 4.8148000e-02
7.4750500e-02 1.0353950e-01 -5.0043500e-01 -9.4900000e-02
8.1481000e-01 -2.0080100e-01 -1.5463000e-01 -3.1004050e-01
-1.4547500e-01 -1.8080500e-01 -3.2145000e-02 1.2955250e-01
-3.0524000e-01 -1.8871500e-01 -1.6616600e-01 -3.1775850e-01
2.5443500e-01 1.3079900e-01 5.8784500e-01 2.1166150e-01
4.5926500e-02 -2.1505300e-01 8.8585000e-02 -1.1020100e-01]

```

In [92]:

```

avg_w2v_vectors_cv_title = []; # the avg-w2v for each sentence/review is stored in this
list
for sentence in tqdm(X_cv['project_title'].values): # for each review/sentence
 vector = np.zeros(300) # as word vectors are of zero length
 cnt_words = 0; # num of words with a valid vector in the sentence/review
 for word in sentence.split(): # for each word in a review/sentence
 if word in glove_words:
 vector += model[word]
 cnt_words += 1
 if cnt_words != 0:
 vector /= cnt_words
 avg_w2v_vectors_cv_title.append(vector)

```

100%|██████████| 11055/11055 [00:00<00:00, 87279.85it/s]

In [93]:

```

avg_w2v_vectors_test_title = []; # the avg-w2v for each sentence/review is stored in th
is list
for sentence in tqdm(X_test['project_title'].values): # for each review/sentence
 vector = np.zeros(300) # as word vectors are of zero length
 cnt_words = 0; # num of words with a valid vector in the sentence/review
 for word in sentence.split(): # for each word in a review/sentence
 if word in glove_words:
 vector += model[word]
 cnt_words += 1
 if cnt_words != 0:
 vector /= cnt_words
 avg_w2v_vectors_test_title.append(vector)

```

100%|██████████| 16500/16500 [00:00<00:00, 100267.77it/s]

## 2.3.4 TF IDF W2V Essay and Title

### 2.3.4.1 TF IDF W2V Essay

In [94]:

```
S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['essay'].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 [95]:

```
average Word2Vec
compute average word2vec for each review.
tfidf_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this
list
for sentence in tqdm(X_train['essay'].values): # 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((sen
 tence.count(word)/len(sentence.split())))
 tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
 tting the tfidf value for each word
 vector += (vec * tf_idf) # calculating tfidf weighted w2v
 tf_idf_weight += tf_idf
 if tf_idf_weight != 0:
 vector /= tf_idf_weight
 tfidf_w2v_vectors_train.append(vector)

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

100%|██████████| 22445/22445 [02:00<00:00, 186.48it/s]

22445

300

In [96]:

```
tfidf_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essay'].values): # 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 tfidf value for each word
 vector += (vec * tf_idf) # calculating tfidf weighted w2v
 tf_idf_weight += tf_idf
 if tf_idf_weight != 0:
 vector /= tf_idf_weight
 tfidf_w2v_vectors_cv.append(vector)
```

100%|██████████| 11055/11055 [00:58<00:00, 187.65it/s]

In [97]:

```
tfidf_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay'].values): # 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 tfidf value for each word
 vector += (vec * tf_idf) # calculating tfidf weighted w2v
 tf_idf_weight += tf_idf
 if tf_idf_weight != 0:
 vector /= tf_idf_weight
 tfidf_w2v_vectors_test.append(vector)
```

100%|██████████| 16500/16500 [01:27<00:00, 189.23it/s]

### 2.3.4.2 TF IDF W2V Title

In [98]:

```
S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['project_title'].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 [99]:

```
average Word2Vec
compute average word2vec for each review.
tfidf_w2v_vectors_train_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['project_title'].values): # 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 tfidf value for each word
 vector += (vec * tf_idf) # calculating tfidf weighted w2v
 tf_idf_weight += tf_idf
 if tf_idf_weight != 0:
 vector /= tf_idf_weight
 tfidf_w2v_vectors_train_title.append(vector)

print(len(tfidf_w2v_vectors_train_title))
print(len(tfidf_w2v_vectors_train_title[0]))
```

100%|██████████| 22445/22445 [00:00<00:00, 69060.82it/s]

22445

300

In [100]:

```
tfidf_w2v_vectors_cv_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['project_title'].values): # 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 tfidf value for each word
 vector += (vec * tf_idf) # calculating tfidf weighted w2v
 tf_idf_weight += tf_idf
 if tf_idf_weight != 0:
 vector /= tf_idf_weight
 tfidf_w2v_vectors_cv_title.append(vector)
```

100%|██████████| 11055/11055 [00:00<00:00, 67241.56it/s]

In [101]:

```
tfidf_w2v_vectors_test_title = []; # the avg-w2v for each sentence/review is stored in
this list
for sentence in tqdm(X_test['project_title'].values): # 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((sen
tence.count(word)/len(sentence.split())))
 tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting the tfidf value for each word
 vector += (vec * tf_idf) # calculating tfidf weighted w2v
 tf_idf_weight += tf_idf
 if tf_idf_weight != 0:
 vector /= tf_idf_weight
 tfidf_w2v_vectors_test_title.append(vector)
```

100%|██████████| 16500/16500 [00:00<00:00, 64198.23it/s]

In [ ]:

## Concatinating all the features

### 1. SET 1 BOW

In [102]:

```
merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_BOW = hstack((X_train_essay_bow, X_train_title_bow, X_train_state_ohe, X_train_tea
cher_ohe, X_train_grade_ohe, X_train_CSC_ohe, X_train_CC_ohe, X_train_price_norm, X_tra
in_TPPP_norm)).tocsr()
X_cr_BOW = hstack((X_cv_essay_bow, X_cv_title_bow, X_cv_state_ohe, X_cv_teacher_ohe, X_
cv_grade_ohe, X_cv_CSC_ohe, X_cv_CC_ohe, X_cv_price_norm, X_cv_TPPP_norm)).tocsr()
X_te_BOW = hstack((X_test_essay_bow, X_test_title_bow, X_test_state_ohe, X_test_teacher
_ohe, X_test_grade_ohe, X_test_CSC_ohe, X_test_CC_ohe, X_test_price_norm, X_test_TPPP_n
orm)).tocsr()

print("Final Data matrix")
print(X_tr_BOW.shape, y_train.shape)
print(X_cr_BOW.shape, y_cv.shape)
print(X_te_BOW.shape, y_test.shape)
print("=*100)
```

Final Data matrix

```
(22445, 7762) (22445,)
(11055, 7762) (11055,)
(16500, 7762) (16500,)
```

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

## 2. SET 2 TF IDF

In [125]:

```
merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_TFIDF = hstack((X_train_essay_tfidf, X_train_title_tfidf, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_CSC_ohe, X_train_CC_ohe, X_train_price_norm, X_train_TPPP_norm)).tocsr()
X_cr_TFIDF = hstack((X_cv_essay_tfidf, X_cv_title_tfidf, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_CSC_ohe, X_cv_CC_ohe, X_cv_price_norm, X_cv_TPPP_norm)).tocsr()
X_te_TFIDF = hstack((X_test_essay_tfidf, X_test_title_tfidf, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_CSC_ohe, X_test_CC_ohe, X_test_price_norm, X_test_TPPP_norm)).tocsr()

print("Final Data matrix")
print(X_tr_TFIDF.shape, y_train.shape)
print(X_cr_TFIDF.shape, y_cv.shape)
print(X_te_TFIDF.shape, y_test.shape)
print("=="*100)
```

```
Final Data matrix
(22445, 7762) (22445,)
(11055, 7762) (11055,)
(16500, 7762) (16500,)
```

## 3. SET 3 AVG W2V

In [127]:

```
merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_AVG_W2V = hstack((avg_w2v_vectors_train, avg_w2v_vectors_train_title, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_CSC_ohe, X_train_CC_ohe, X_train_price_norm, X_train_TPPP_norm)).tocsr()
X_cr_AVG_W2V = hstack((avg_w2v_vectors_cv, avg_w2v_vectors_cv_title, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_CSC_ohe, X_cv_CC_ohe, X_cv_price_norm, X_cv_TPPP_norm)).tocsr()
X_te_AVG_W2V = hstack((avg_w2v_vectors_test, avg_w2v_vectors_test_title, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_CSC_ohe, X_test_CC_ohe, X_test_price_norm, X_test_TPPP_norm)).tocsr()

print("Final Data matrix")
print(X_tr_AVG_W2V.shape, y_train.shape)
print(X_cr_AVG_W2V.shape, y_cv.shape)
print(X_te_AVG_W2V.shape, y_test.shape)
print("=="*100)
```

```
Final Data matrix
(22445, 696) (22445,)
(11055, 696) (11055,)
(16500, 696) (16500,)
```

## 4 SET 1 TFIDF W2V

In [129]:

```
merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_TFIDF_W2V = hstack((tfidf_w2v_vectors_train, tfidf_w2v_vectors_train_title, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_CSC_ohe, X_train_CC_ohe, X_train_price_norm, X_train_TPPP_norm)).tocsr()
X_cr_TFIDF_W2V = hstack((tfidf_w2v_vectors_cv, tfidf_w2v_vectors_cv_title, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_CSC_ohe, X_cv_CC_ohe, X_cv_price_norm, X_cv_TPPP_norm)).tocsr()
X_te_TFIDF_W2V = hstack((tfidf_w2v_vectors_test, tfidf_w2v_vectors_test_title, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_CSC_ohe, X_test_CC_ohe, X_test_price_norm, X_test_TPPP_norm)).tocsr()

print("Final Data matrix")
print(X_tr_TFIDF_W2V.shape, y_train.shape)
print(X_cr_TFIDF_W2V.shape, y_cv.shape)
print(X_te_TFIDF_W2V.shape, y_test.shape)
print("=*100)
```

Final Data matrix

(22445, 696) (22445,)

(11055, 696) (11055,)

(16500, 696) (16500,)

=====

In [ ]:

## 2.4 Applying KNN on different kind of featurization as mentioned in the instructions

Apply KNN on different kind of featurization as mentioned in the instructions

For Every model that you work on make sure you do the step 2 and step 3 of instructions

In [103]:

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

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

### 2.4.1 Applying KNN brute force on BOW, SET 1



In [104]:

```
Please write all the code with proper documentation
```

In [105]:

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

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

 return y_data_pred
```

In [140]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence va
lues, or non-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [5, 15, 29, 37, 49, 57]
for i in tqdm(K):
 neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
 neigh.fit(X_tr_BOW, y_train)

 y_train_pred = batch_predict(neigh, X_tr_BOW)
 y_cv_pred = batch_predict(neigh, X_cr_BOW)

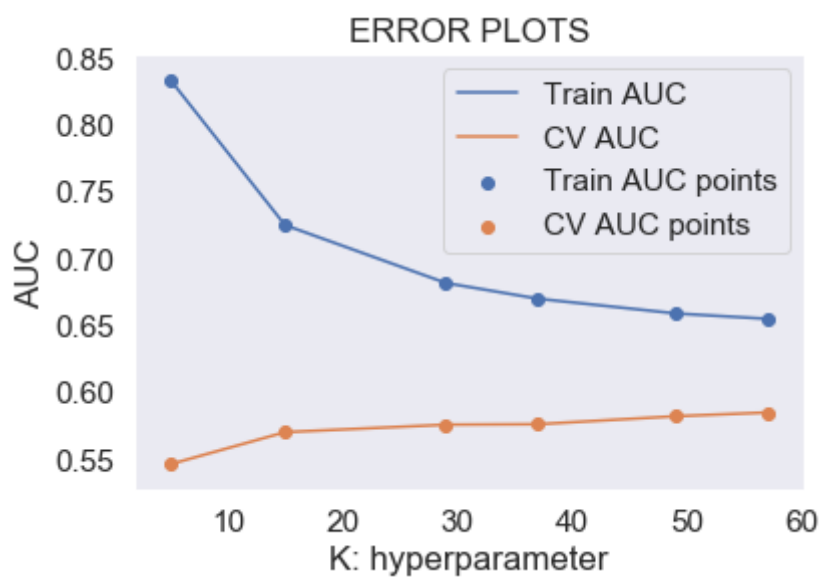
 # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
 of the positive class
 # not the predicted outputs
 train_auc.append(roc_auc_score(y_train, y_train_pred))
 cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

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

|      |  |     |                           |
|------|--|-----|---------------------------|
| 0%   |  | 0/6 | [00:00<?, ?it/s]          |
| 17%  |  | 1/6 | [01:52<09:21, 112.34s/it] |
| 33%  |  | 2/6 | [03:25<07:06, 106.54s/it] |
| 50%  |  | 3/6 | [05:01<05:10, 103.44s/it] |
| 67%  |  | 4/6 | [06:37<03:22, 101.27s/it] |
| 83%  |  | 5/6 | [08:07<01:37, 97.89s/it]  |
| 100% |  | 6/6 | [10:00<00:00, 100.06s/it] |



In [141]:

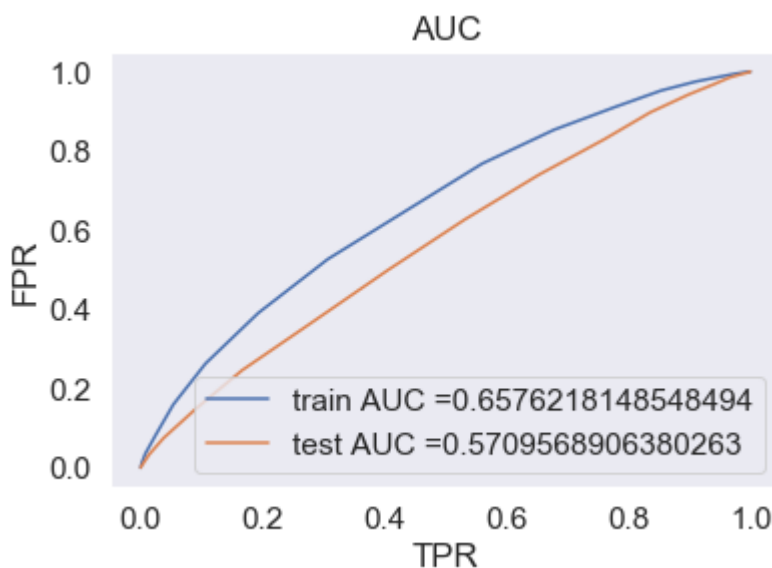
```
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier(n_neighbors=51, n_jobs=-1)
neigh.fit(X_tr_BOW, y_train)
roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr_BOW)
y_test_pred = batch_predict(neigh, X_te_BOW)

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

plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("TPR")
plt.ylabel("FPR")
plt.title("AUC")
plt.grid()
plt.show()
```



In [142]:

```
we are writing our own function for predict, with defined threshold
we will pick a threshold that will give the least fpr
def predict(proba, 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))
 predictions = []
 for i in proba:
 if i>=t:
 predictions.append(1)
 else:
 predictions.append(0)
 return predictions
```

In [143]:

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

```
=====
=====
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.36626947978883134 for threshold 0.824
[[2026 1569]
 [6599 12251]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2964714342259638 for threshold 0.843
[[1572 1070]
 [6953 6905]]
```

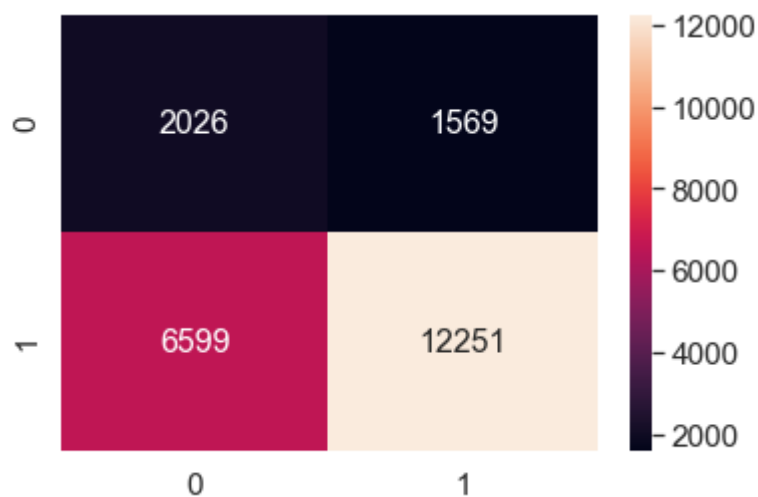
In [144]:

```
conf_matr_df_train = pd.DataFrame(confusion_matrix(y_train[:], predict(y_train_pred, t
r_thresholds, train_fpr, train_tpr)))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

the maximum value of  $tpr \cdot (1 - fpr)$  0.36626947978883134 for threshold 0.824

Out[144]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x204ef53ea08>



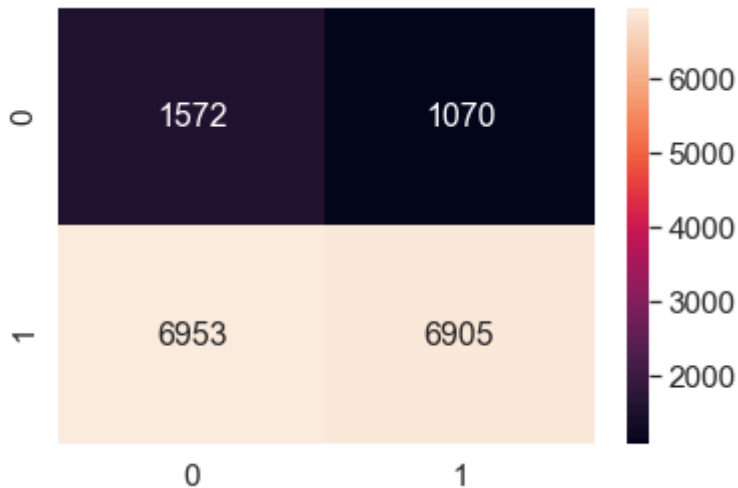
In [145]:

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

the maximum value of  $tpr \cdot (1 - fpr)$  0.2964714342259638 for threshold 0.843

Out[145]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x2049550a908>



In [146]:

```
print(train_fpr.shape)
print(train_tpr.shape)
print(len(y_train_pred))
```

```
(24,)
(24,)
22445
```

In [ ]:

## 2.4.2 Applying KNN brute force on TFIDF, SET 2

In [64]:

```
Please write all the code with proper documentation
```

In [147]:

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

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

 return y_data_pred
```



In [148]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence va
lues, or non-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [15,37,49,57,69]
for i in tqdm(K):
 neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
 neigh.fit(X_tr_TFIDF, y_train)

 y_train_pred = batch_predict(neigh, X_tr_TFIDF)
 y_cv_pred = batch_predict(neigh, X_cr_TFIDF)

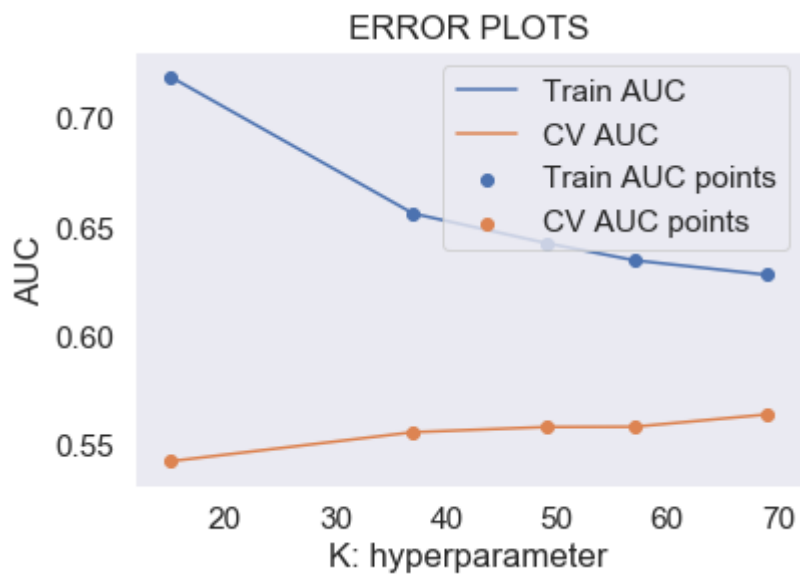
 # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
 of the positive class
 # not the predicted outputs
 train_auc.append(roc_auc_score(y_train,y_train_pred))
 cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

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

|      |       |                               |
|------|-------|-------------------------------|
| 0%   |       | 0/5 [00:00<?, ?it/s]          |
| 20%  | █     | 1/5 [02:14<08:56, 134.15s/it] |
| 40%  | ██    | 2/5 [04:10<06:26, 128.69s/it] |
| 60%  | ███   | 3/5 [06:02<04:07, 123.76s/it] |
| 80%  | ████  | 4/5 [07:54<02:00, 120.14s/it] |
| 100% | █████ | 5/5 [09:46<00:00, 117.37s/it] |



In [149]:

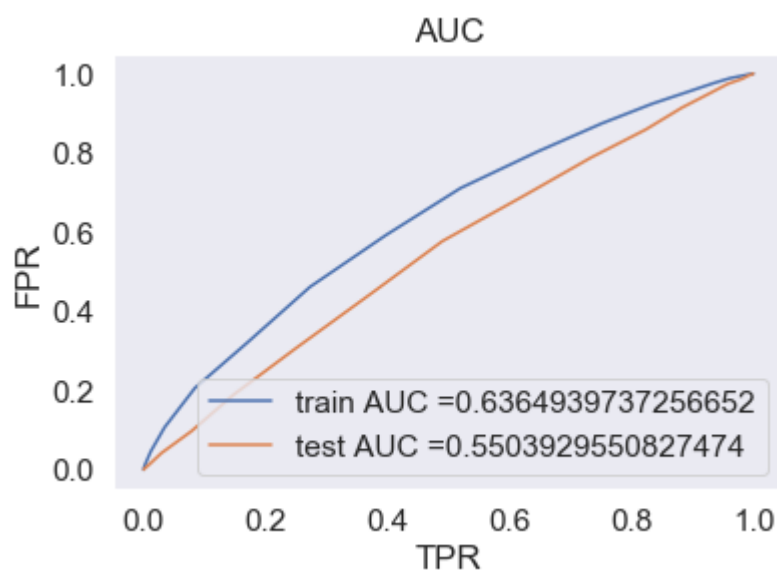
```
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier(n_neighbors=55, n_jobs=-1)
neigh.fit(X_tr_TFIDF, y_train)
roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr_TFIDF)
y_test_pred = batch_predict(neigh, X_te_TFIDF)

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

plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("TPR")
plt.ylabel("FPR")
plt.title("AUC")
plt.grid()
plt.show()
```



In [150]:

```
we are writing our own function for predict, with defined threshold
we will pick a threshold that will give the least fpr
def predict(proba, 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))
 predictions = []
 for i in proba:
 if i>=t:
 predictions.append(1)
 else:
 predictions.append(0)
 return predictions
```

In [151]:

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

```
=====
=====
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.3568674440997111 for threshold 0.855
[[2166 1429]
 [7685 11165]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2942837861563087 for threshold 0.855
[[1348 1294]
 [5865 7993]]
```

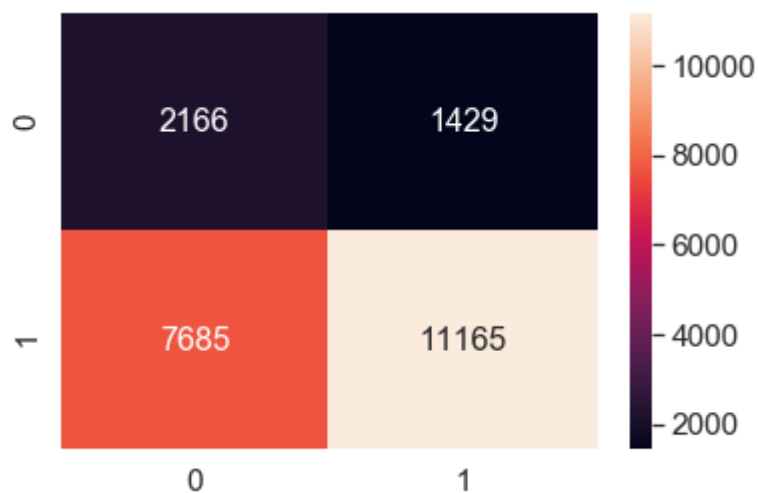
In [152]:

```
conf_matr_df_train = pd.DataFrame(confusion_matrix(y_train[:], predict(y_train_pred, t
r_thresholds, train_fpr, train_tpr)))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

the maximum value of  $tpr \cdot (1 - fpr)$  0.3568674440997111 for threshold 0.855

Out[152]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x204f21c8748>



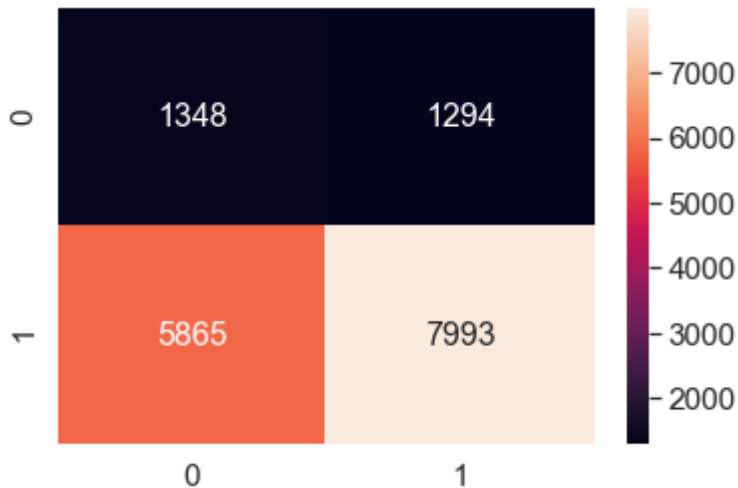
In [153]:

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

the maximum value of  $tpr \cdot (1 - fpr)$  0.2942837861563087 for threshold 0.855

Out[153]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x204ec1d5e08>



In [ ]:

### 2.4.3 Applying KNN brute force on AVG W2V, SET 3

In [154]:

```
Please write all the code with proper documentation
```

In [ ]:

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

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

 return y_data_pred
```

In [156]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence va
lues, or non-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.
"""

train_auc = []
cv_auc = []
K = [15,37,49,57,69]
for i in tqdm(K):
 neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
 neigh.fit(X_tr_AVG_W2V, y_train)

 y_train_pred = batch_predict(neigh, X_tr_AVG_W2V)
 y_cv_pred = batch_predict(neigh, X_cr_AVG_W2V)

 # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
 of the positive class
 # not the predicted outputs
 train_auc.append(roc_auc_score(y_train,y_train_pred))
 cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

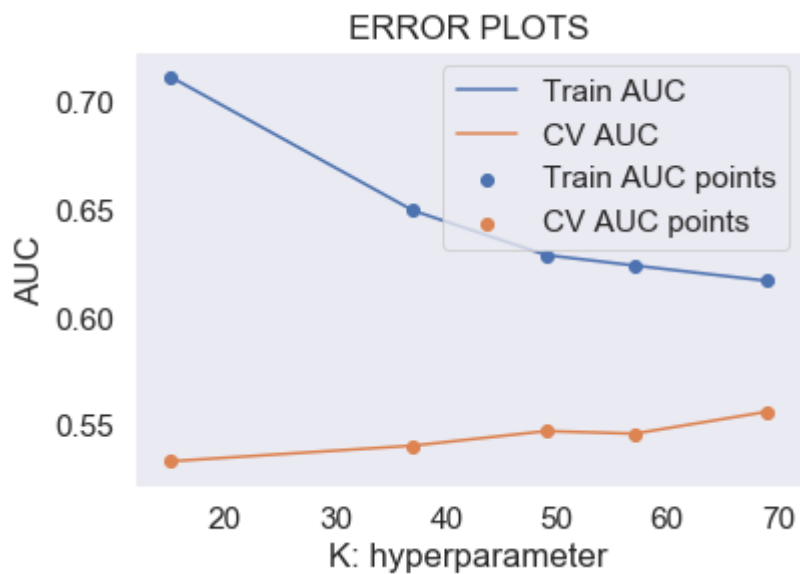
plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

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



|      |       |                               |
|------|-------|-------------------------------|
| 0%   |       | 0/5 [00:00<?, ?it/s]          |
| 20%  | █     | 1/5 [07:20<29:22, 440.60s/it] |
| 40%  | ██    | 2/5 [13:58<21:23, 427.82s/it] |
| 60%  | ███   | 3/5 [19:53<13:31, 405.82s/it] |
| 80%  | ████  | 4/5 [25:46<06:29, 389.98s/it] |
| 100% | █████ | 5/5 [31:38<00:00, 379.71s/it] |



In [157]:

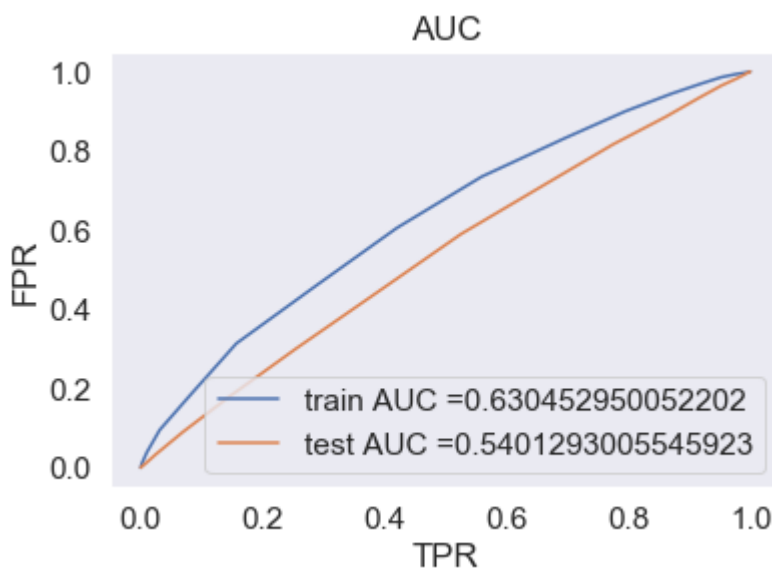
```
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier(n_neighbors=47, n_jobs=-1)
neigh.fit(X_tr_AVG_W2V, y_train)
roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr_AVG_W2V)
y_test_pred = batch_predict(neigh, X_te_AVG_W2V)

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

plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("TPR")
plt.ylabel("FPR")
plt.title("AUC")
plt.grid()
plt.show()
```



In [158]:

```
we are writing our own function for predict, with defined threshold
we will pick a threshold that will give the least fpr
def predict(proba, 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))
 predictions = []
 for i in proba:
 if i>=t:
 predictions.append(1)
 else:
 predictions.append(0)
 return predictions
```

In [159]:

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

```
=====
=====
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.35093701464235255 for threshold 0.851
[[2085 1510]
 [7444 11406]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2798933139186487 for threshold 0.851
[[1254 1388]
 [5686 8172]]
```

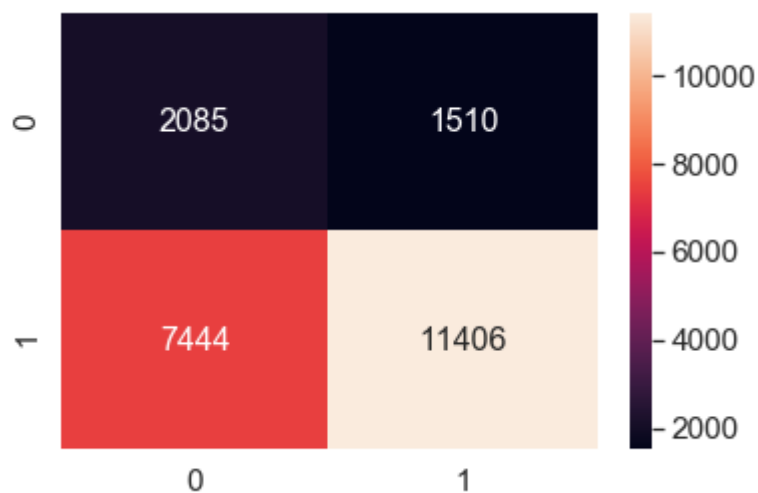
In [160]:

```
conf_matr_df_train = pd.DataFrame(confusion_matrix(y_train[:], predict(y_train_pred, t
r_thresholds, train_fpr, train_tpr)))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

the maximum value of  $tpr \cdot (1 - fpr)$  0.35093701464235255 for threshold 0.851

Out[160]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x204eb6901c8>



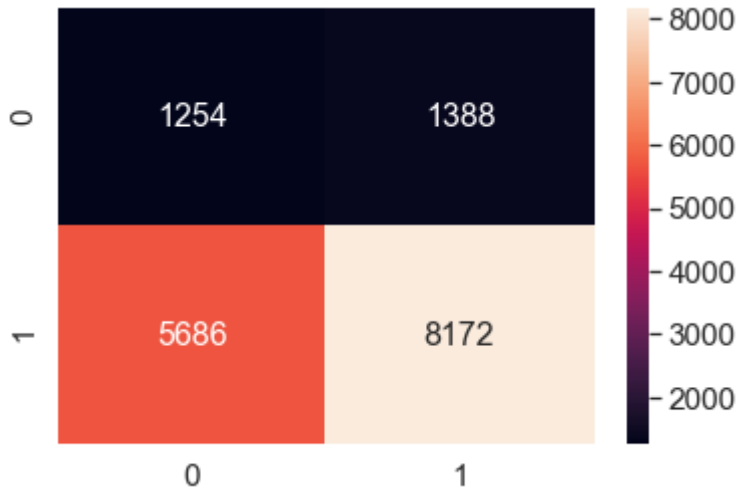
In [161]:

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

the maximum value of  $tpr \cdot (1 - fpr)$  0.2798933139186487 for threshold 0.851

Out[161]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x204ef705288>



In [ ]:

## 2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

In [162]:

```
Please write all the code with proper documentation
```

In [163]:

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

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

 return y_data_pred
```

In [164]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence va
lues, or non-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.
"""

train_auc = []
cv_auc = []
K = [15,37,49,57,69]
for i in tqdm(K):
 neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
 neigh.fit(X_tr_TFIDF_W2V, y_train)

 y_train_pred = batch_predict(neigh, X_tr_TFIDF_W2V)
 y_cv_pred = batch_predict(neigh, X_cr_TFIDF_W2V)

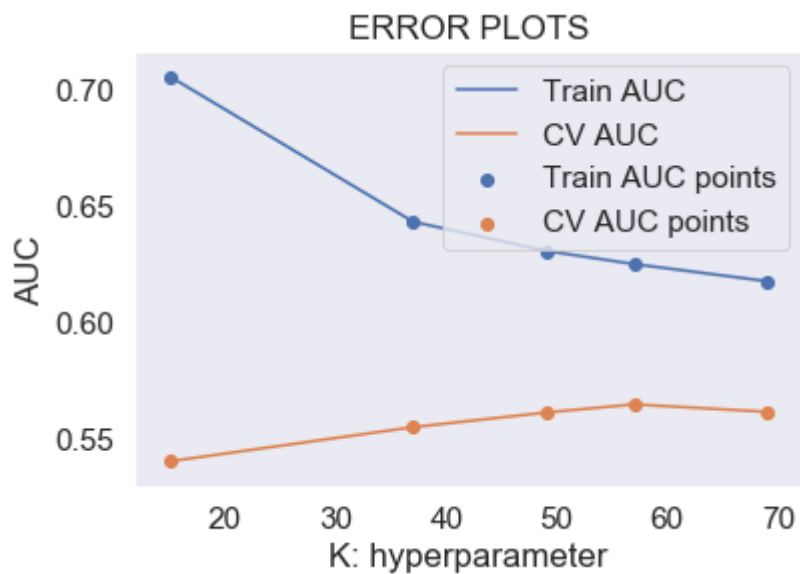
 # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
 of the positive class
 # not the predicted outputs
 train_auc.append(roc_auc_score(y_train,y_train_pred))
 cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

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

|      |       |                               |
|------|-------|-------------------------------|
| 0%   |       | 0/5 [00:00<?, ?it/s]          |
| 20%  | █     | 1/5 [05:44<22:56, 344.17s/it] |
| 40%  | ██    | 2/5 [11:24<17:09, 343.07s/it] |
| 60%  | ███   | 3/5 [17:05<11:24, 342.30s/it] |
| 80%  | ████  | 4/5 [22:46<05:41, 341.94s/it] |
| 100% | █████ | 5/5 [28:20<00:00, 340.11s/it] |





In [171]:

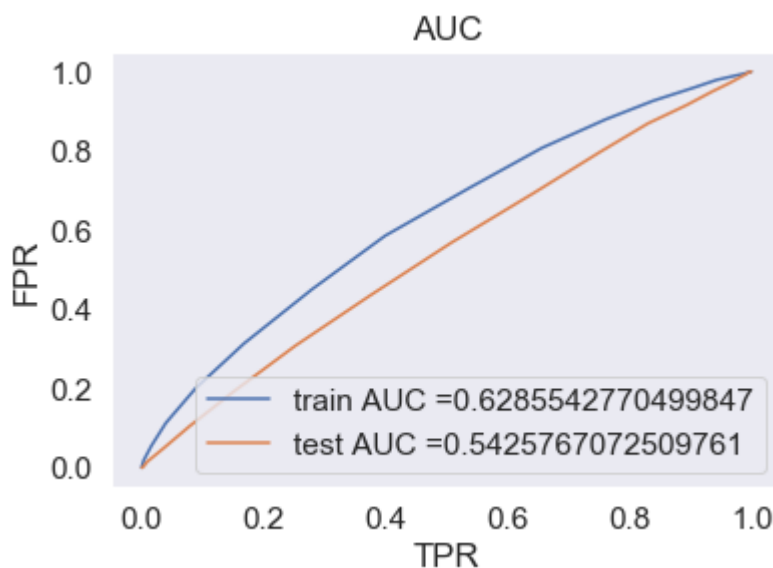
```
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier(n_neighbors=53, n_jobs=-1)
neigh.fit(X_tr_TFIDF_W2V, y_train)
roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr_TFIDF_W2V)
y_test_pred = batch_predict(neigh, X_te_TFIDF_W2V)

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

plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("TPR")
plt.ylabel("FPR")
plt.title("AUC")
plt.grid()
plt.show()
```



In [176]:

```
we are writing our own function for predict, with defined threshold
we will pick a threshold that will give the least fpr
def predict(proba, 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))
 predictions = []
 for i in proba:
 if i>=t:
 predictions.append(1)
 else:
 predictions.append(0)
 return predictions
```

In [177]:

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

```
=====
=====
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.35169980410458085 for threshold 0.849
[[2159 1436]
 [7811 11039]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2795085308332848 for threshold 0.849
[[1300 1342]
 [5986 7872]]
```

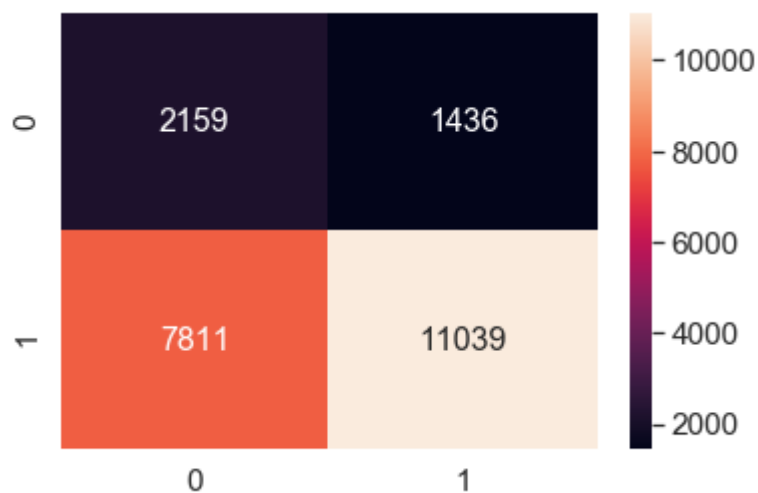
In [178]:

```
conf_matr_df_train = pd.DataFrame(confusion_matrix(y_train[:], predict(y_train_pred, t
r_thresholds, train_fpr, train_tpr)))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

the maximum value of  $tpr \cdot (1 - fpr)$  0.35169980410458085 for threshold 0.849

Out[178]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x20495362308>



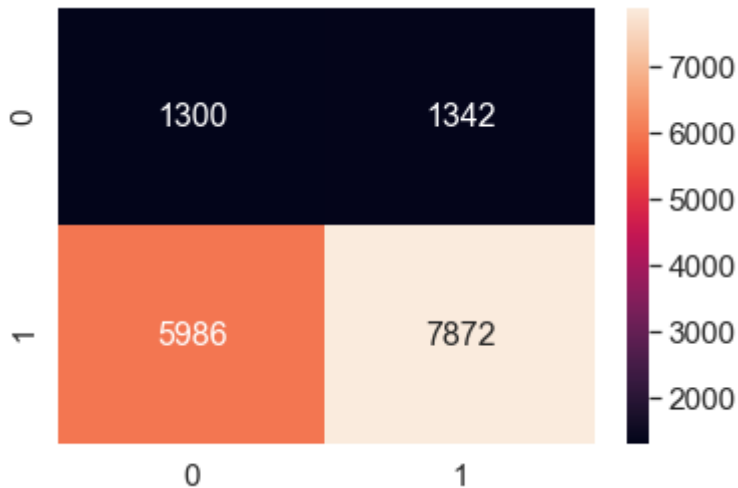
In [179]:

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

the maximum value of  $tpr \cdot (1 - fpr)$  0.2795085308332848 for threshold 0.849

Out[179]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x204f2f48548>



In [ ]:

## 2.5 Feature selection with `SelectKBest`

In [180]:

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

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

In [181]:

```
from sklearn.feature_selection import SelectKBest, chi2
t = SelectKBest(chi2,k=2000).fit(X_tr_TFIDF, y_train)
X_tr_KBEST = t.transform(X_tr_TFIDF)
X_te_KBEST = t.transform(X_te_TFIDF)
X_cr_KBEST = t.transform(X_cr_TFIDF)

print("Final Data matrix on TFIDF")
print(X_tr_KBEST.shape, y_train.shape)
print(X_cr_KBEST.shape, y_cv.shape)
print(X_te_KBEST.shape, y_test.shape)
print("="*100)
```

Final Data matrix on TFIDF

```
(22445, 2000) (22445,)
(11055, 2000) (11055,)
(16500, 2000) (16500,)
```

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

In [182]:

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

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

 return y_data_pred
```

In [183]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence va
lues, or non-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.
"""

train_auc = []
cv_auc = []
K = [15,37,49,57,69]
for i in tqdm(K):
 neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
 neigh.fit(X_tr_KBEST, y_train)

 y_train_pred = batch_predict(neigh, X_tr_KBEST)
 y_cv_pred = batch_predict(neigh, X_cr_KBEST)

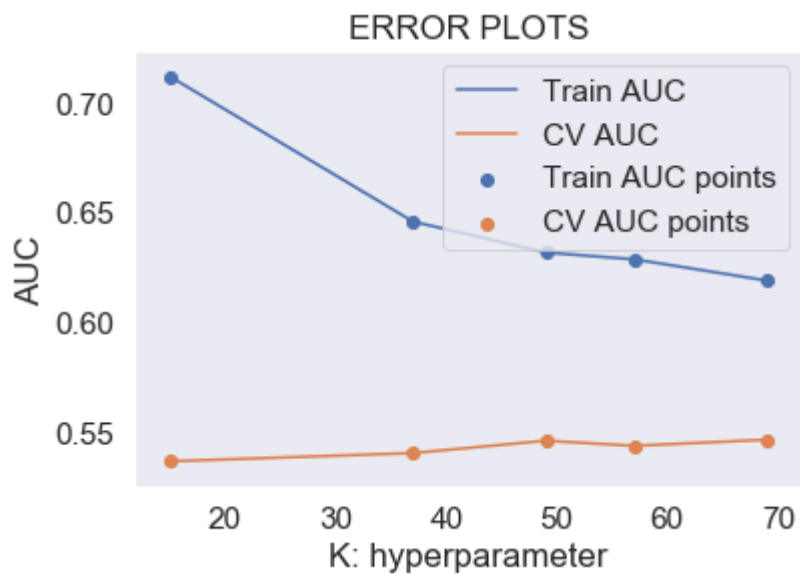
 # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
 of the positive class
 # not the predicted outputs
 train_auc.append(roc_auc_score(y_train,y_train_pred))
 cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

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

|      |       |                              |
|------|-------|------------------------------|
| 0%   |       | 0/5 [00:00<?, ?it/s]         |
| 20%  | █     | 1/5 [00:52<03:28, 52.10s/it] |
| 40%  | ██    | 2/5 [01:40<02:33, 51.12s/it] |
| 60%  | ███   | 3/5 [02:29<01:40, 50.47s/it] |
| 80%  | ████  | 4/5 [03:18<00:49, 49.79s/it] |
| 100% | █████ | 5/5 [04:06<00:00, 49.34s/it] |



In [191]:

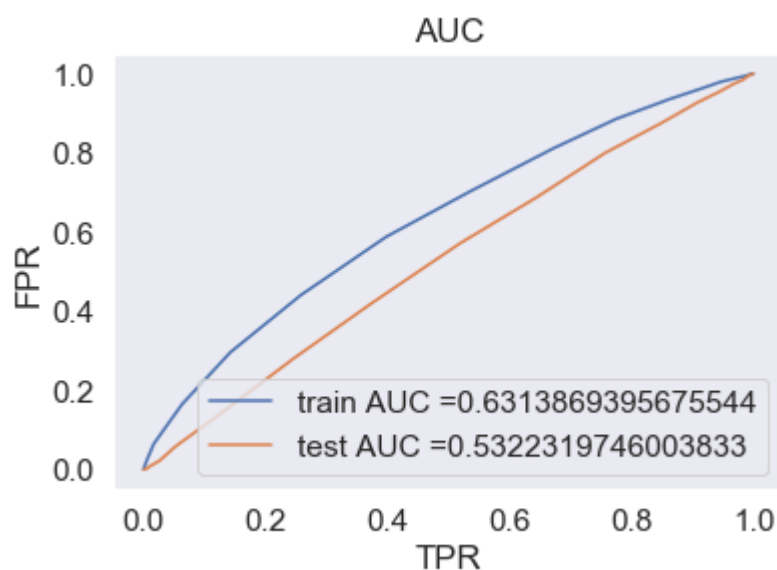
```
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier(n_neighbors=50, n_jobs=-1)
neigh.fit(X_tr_KBEST, y_train)
roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr_KBEST)
y_test_pred = batch_predict(neigh, X_te_KBEST)

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

plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("TPR")
plt.ylabel("FPR")
plt.title("AUC")
plt.grid()
plt.show()
```





In [192]:

```
we are writing our own function for predict, with defined threshold
we will pick a threshold that will give the least fpr
def predict(proba, 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))
 predictions = []
 for i in proba:
 if i>=t:
 predictions.append(1)
 else:
 predictions.append(0)
 return predictions
```

In [193]:

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

```
=====
=====
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.35405469577183163 for threshold 0.84
[[2166 1429]
 [7773 11077]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.27490492678578626 for threshold 0.84
[[1271 1371]
 [5939 7919]]
```

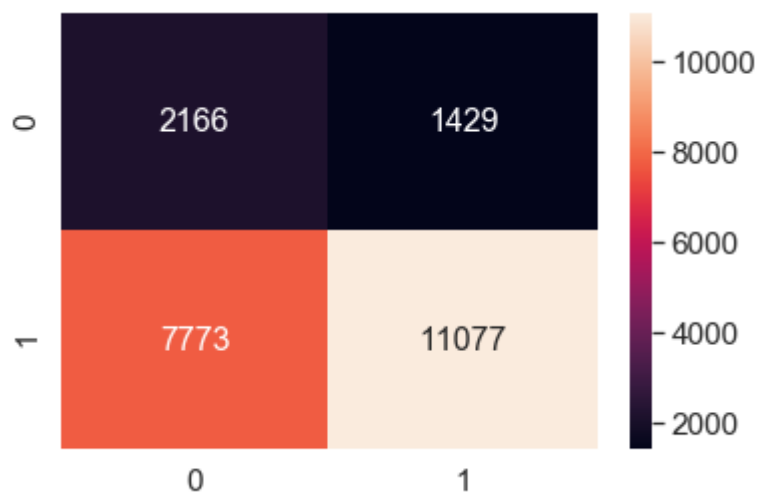
In [194]:

```
conf_matr_df_train = pd.DataFrame(confusion_matrix(y_train[:], predict(y_train_pred, t
r_thresholds, train_fpr, train_tpr)))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

the maximum value of  $tpr \cdot (1 - fpr)$  0.35405469577183163 for threshold 0.84

Out[194]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x204f21c8a48>



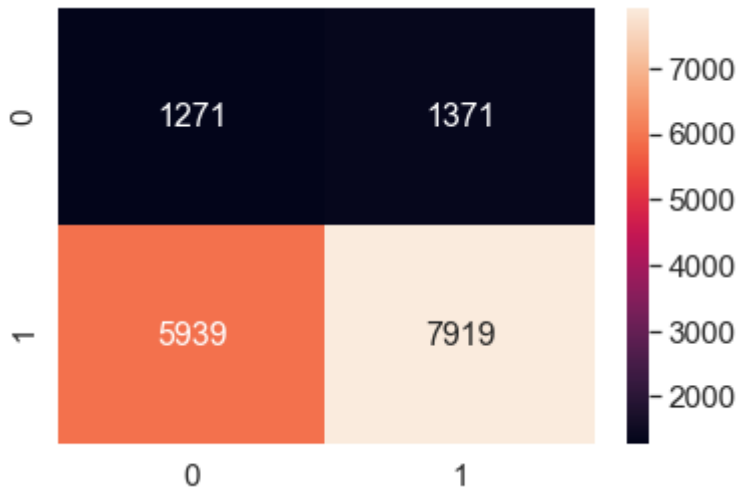
In [195]:

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

the maximum value of  $tpr \cdot (1 - fpr)$  0.27490492678578626 for threshold 0.84

Out[195]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x204a64ca288>



In [ ]:

### 3. Conclusions

In [0]:

```
Please compare all your models using Prettytable Library
```

In [197]:

```
from prettytable import PrettyTable
#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable
x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyper Parameter", "AUC"]
x.add_row(["BOW", "Brute", 51, 0.57])
x.add_row(["TFIDF", "Brute", 55, 0.55])
x.add_row(["AVG W2V", "Brute", 47, 0.54])
x.add_row(["TFIDF W2V", "Brute", 53, 0.54])
x.add_row(["TFIDF", "Top 2000", 50, 0.53])
print(x)
```

| Vectorizer | Model    | Hyper Parameter | AUC  |
|------------|----------|-----------------|------|
| BOW        | Brute    | 51              | 0.57 |
| TFIDF      | Brute    | 55              | 0.55 |
| AVG W2V    | Brute    | 47              | 0.54 |
| TFIDF W2V  | Brute    | 53              | 0.54 |
| TFIDF      | Top 2000 | 50              | 0.53 |

### Observation

As there is a minor difference in AUC for normal brute force models and top 2000 features, considering the time complexity we can select the top 2000 features.

In [ ]: