Document Created Date:14-02-2019

Grasping the data and datasource - 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

Feature	
project_id	A unique identifier for the proposed project. **Example**
project_title	Title of the projec • Art Will Make Y • First
project_grade_category	Grade level of students for which the project is targeted. One of t enumera • Grad • G • G • Gr
project_subject_categories	One or more (comma-separated) subject categories for the pro following enumerated li • Applied • Care • Health • History • Literacy & • Math • Music & • Spec •
	• Music & • Literacy & Language, Math
school_state	State where school is located (Two-letter U.S. (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Pos **Ex**
project_subject_subcategories	One or more (comma-separated) subject subcategories fo • • Literature & Writing, Social
project_resource_summary	An explanation of the resources needed for the projec • My students need hands on literacy materials f sensory nee
project_essay_1	First applic
project_essay_2	Second applic
project_essay_3	Third applic
project_essay_4	Fourth applic
project_submitted_datetime	Datetime when project application was submitted. **Example:** 2(12:
teacher_id	A unique identifier for the teacher of the proposed proje bdf8baa8fedef6bfeec7ae4

Feature	
`teacher_prefix`	Teacher's title. One of the following enumera • • • • • •
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the sa I

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

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

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

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

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

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

Notes on the Essay Data

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

- **project_essay_1:** "Introduce us to your classroom"
- **project_essay_2:** "Tell us more about your students"
- **project_essay_3:** "Describe how your students will use the materials you're requesting"
- **project_essay_3:** "Close by sharing why your project will make a difference"

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

- **project_essay_1:** "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."
- **project_essay_2:** "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

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

KNN

Step by Step Procedure

- Understanding the Businessreal world problem
- Loading the data
- Preprocessing the data(based on the type of data = categorical , text, Numarical)
- Preprocessing data includes (removing outliers, impute missung values, cleaning data, remove spacial character, etc..)
- Split the data into train, cv, test(random splitting)
- Vectorization data (one hot encoding)
- Vectorizing text data(bow, tfidf, avgw2v, tfidf weighted w2v)
- vectorizing numarical - Normalizer
- Computing Sentiment Scores
- Applying K Nearest Neighbors
- Contactinating all the type of features(cat + text + num)
- Hyper parameter Tuning to find alpha:: Simple cross Validation (applied two techniques - this is one)
- Hyperparameter tuning to find th best estimator(GridSearchCV- 2nd technique)
- Train the KNN model using best hyperparameter and ploting auc roc-curve
- Ploting confusion matrix(heatmaps)
- Feature selection with SelectKBest
- Hyperparameter tuning to find th best estimator(GridSearchCV)
- Train the KNN model using best hyperparameter and ploting auc roc-curve
- Ploting confusion matrix(heatmaps)
- Observation on overall model performences(Conclusion)
- Ploting the performences by table format.

```
C:\Users\Ramesh Battu> import required libraries
```

In [1]:

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

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

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

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

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

from tqdm import tqdm
import os

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

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

#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/40840
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	0(
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	0(

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

1.1.1 preprocessing of project_subject_categories

In [6]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47

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

1.1.2 preprocessing of project_subject_subcategories

In [7]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47

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

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

1.1.3 preprocessing of `school_state`

In [8]:

```
school_state_catogories = list(project_data['school_state'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47

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

project_data['school_state'] = cat_list


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

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

1.1.4 preprocessing of `teacher_prefix`

In [9]:

```
# citation code :https://www.datacamp.com/community/tutorials/categorical-data
project_data = project_data.fillna(project_data['teacher_prefix'].value_counts().index[0
teacher_prefix_catogories = list(project_data['teacher_prefix'].values)
# Citation code : https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit
# To convert the data type object to unicode string : used """astype('U')""" code from
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
# remove special characters from list of strings python: https://stackoverflow.com/a/47
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-stri
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-pyth
cat_list = []
for i in teacher_prefix_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth
        if 'The' in j.split(): # this will split each of the catogory based on space "M
            j=j.replace('The','') # if we have the words "The" we are going to replace
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"M
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spa
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

project_data['teacher_prefix'] = cat_list


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

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
teacher_prefix_dict = dict(my_counter)
sorted_teacher_prefix_dict = dict(sorted(teacher_prefix_dict.items(), key=lambda kv: kv
```

1.1.5 Preprocessing of `project_grade_category`

In [10]:

```
# Feature encoding with 'project_grade_category'
project_grade_catogories = list(project_data['project_grade_category'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-stri
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-pyth
cat_list = []
for i in project_grade_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth
        if 'The' in j.split(): # this will split each of the catogory based on space "M
            j=j.replace('The','') # if we have the words "The" we are going to replace
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"M
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spa
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

project_data['project_grade_category'] = cat_list

#link : https://www.datacamp.com/community/tutorials/categorical-data
project_data = project_data.fillna(project_data['project_grade_category'].value_counts(

# Citation code : https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-sciki
# To convert the data type object to unicode string : used """astype('U')""" code from
# 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:
    word = str(word)
    my_counter.update(word.split())



# dict sort by value python: https://stackoverflow.com/a/613218/4084039
project_grade_category_dict = dict(my_counter)
sorted_project_grade_category_dict = dict(sorted(project_grade_category_dict.items(), k
```

1.2. Text Preprocessing

1.2.1 Text Preprocessing of essay

In [11]:

```
# 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 [12]:

```
project_data.head(1)
```

Out[12]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	00

In [13]:

```
# 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 class
room 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 les
sons.My students come from a variety of backgrounds, including language
and socioeconomic status.  Many of them don't have a lot of experience i
n science and engineering and these kits give me the materials to provid
e these exciting opportunities for my students.Each month I try to do se
veral science or STEM/STEAM projects.  I would use the kits and robot to
help guide my science instruction in engaging and meaningful ways.  I ca
n adapt the kits to my current language arts pacing guide where we alrea
dy 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 sch
ool year where I will implement these kits: magnets, motion, sink vs. fl
oat, 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 add
itional ideas, strategies, and lessons to prepare my students in scienc
e.It is challenging to develop high quality science activities.  These k
its give me the materials I need to provide my students with science act
```

In [14]:

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

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

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

In [15]:

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

```
\"A person is a person, no matter how small.\" (Dr.Seuss) I teach the smal
lest 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. \r\nStud
ents in my class come from a variety of different backgrounds which makes
for wonderful sharing of experiences and cultures, including Native Americ
ans.\r\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 mate
rials and have many different opportunities to practice a skill before it
is mastered. Having the social skills to work cooperatively with friends i
s a crucial aspect of the kindergarten curriculum.Montana is the perfect p
lace to learn about agriculture and nutrition. My students love to role pl
ay in our pretend kitchen in the early childhood classroom. I have had sev
eral 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 m
ath and writing concepts while cooking delicious healthy food for snack ti
me. My students will have a grounded appreciation for the work that went i
nto making the food and knowledge of where the ingredients came from as we
ll as how it is healthy for their bodies. This project would expand our le
arning of nutrition and agricultural cooking recipes by having us peel our
own apples to make homemade applesauce, make our own bread, and mix up hea
lthy plants from our classroom garden in the spring. We will also create o
ur own cookbooks to be printed and shared with families. \r\nStudents will
gain math and literature skills as well as a life long enjoyment for healt
hy cooking.Mrs.Mrs.
==================================================
```

In [16]:

```
# \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks-py
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 smalle
st 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 w
onderful 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 cla
ssroom. Kindergarteners in my class love to work with hands-on materials a
nd have many different opportunities to practice a skill before it is mast
ered. Having the social skills to work cooperatively with friends is a cru
cial aspect of the kindergarten curriculum.Montana is the perfect place to
learn about agriculture and nutrition. My students love to role play in ou
r pretend kitchen in the early childhood classroom. I have had several kid
s 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 wri
ting concepts while cooking delicious healthy food for snack time. My stud
ents 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 i
t is healthy for their bodies. This project would expand our learning of n
utrition 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 cookb
ooks to be printed and shared with families.  Students will gain math and
literature skills as well as a life long enjoyment for healthy cooking.Mr
s.Mrs.
```

In [17]:

```
#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 st
udents 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 cl
ass 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 col
laborative student project based learning in and out of the classroom Kind
ergarteners in my class love to work with hands on materials and have many
different opportunities to practice a skill before it is mastered Having t
he social skills to work cooperatively with friends is a crucial aspect of
the kindergarten curriculum Montana is the perfect place to learn about ag
riculture and nutrition My students love to role play in our pretend kitch
en 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 C
ooking Lessons where we learn important math and writing concepts while co
oking delicious healthy food for snack time My students will have a ground
ed 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 agricultura
l cooking recipes by having us peel our own apples to make homemade apples
auce make our own bread and mix up healthy plants from our classroom garde
n in the spring We will also create our own cookbooks to be printed and sh
ared with families Students will gain math and literature skills as well a
s a life long enjoyment for healthy cooking Mrs Mrs
```

In [18]:

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

In [19]:

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

```
  0%|          | 0/109248 [00:00<?, ?it/s]

CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 6.91 µs

100%|██████████| 109248/109248 [01:18<00:00, 1397.58it/s]
```

In [20]:

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

Out[20]:

```
'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 mrs mrs'
```

1.2.2 Text Preprocessing of `project_title`

In [21]:

```
# similarly you can preprocess the titles also
print(project_data['project_title'].head(1))
```

```
55660    Engineering STEAM into the Primary Classroom
Name: project_title, dtype: object
```

In [22]:

```
# printing some random title texts
print(project_data['project_title'].values[20000])
print('--'*19)
print(project_data['project_title'].values[1959])
print('--'*19)
print(project_data['project_title'].values[1969])
print('--'*19)
print(project_data['project_title'].values[1989])
print('--'*19)
```

```
Health Nutritional Cooking in Kindergarten
--------------------------------------
Learning is FUN!!!!!!
--------------------------------------
Divide and Color!
--------------------------------------
Kindergarten Wants to Be Zookeepers with Letters Alive!
--------------------------------------
```

In [23]:

```
# 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 [24]:

```
sent = decontracted(project_data['project_title'].values[15000])
print(sent)
```

```
Building Independence One Piece at a Time.
```

In [25]:

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

```
Building Independence One Piece at a Time.
```

In [26]:

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

```
Building Independence One Piece at a Time
```

In [27]:

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

In [28]:

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

```
100%|██████████| 109248/109248 [00:02<00:00, 48085.22it/s]
```

In [29]:

```
# after preprocesing
preprocessed_project_title[15000]
```

Out[29]:

```
'building independence one piece time'
```

In [30]:

```
# merge data frames
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_i
project_data = pd.merge(project_data, price_data, on='id', how='left')
project_data.shape
project_data.head(1)
```

Out[30]:

Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Dat
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016 04-2 00:27:3

1.3. Numerical standardization

1.3.1 standardization of `price`

In [31]:

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

# price_standardized = standardScalar.fit(project_data_train['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... .
# Reshape your data either using array.reshape(-1, 1)
# standardize the data with maen and variance.
price_scalar = StandardScaler(with_mean = False)
price_standardized = price_scalar.fit_transform(project_data['price'].values.reshape(-1
print("Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[(
print("------------------------------------------------------------")
print("shape of price_standardized:",  price_standardized.shape)
```

```
Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scala
r.var_[0])}
------------------------------------------------------------
shape of price_standardized: (109248, 1)
```

In [32]:

```
price_standardized
```

Out[32]:

```
array([[1.97294477],
       [0.57967923],
       [0.89524699],
       ...,
       [1.08572507],
       [0.78294656],
       [0.01496614]])
```

1.3.2 standardization of teacher_number_of_previously_posted_projects

In [33]:

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

# price_standardized = standardScalar.fit(project_data_train['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... .
# Reshape your data either using array.reshape(-1, 1)
# standardize the data with maen and variance.
teacher_number_of_previously_posted_projects_scalar = StandardScaler(with_mean = False)
teacher_number_of_previously_posted_projects_standardized = teacher_number_of_previousl
print("Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[(
print("------------------------------------------------------------")
print("shape of price_standardized:",  teacher_number_of_previously_posted_projects_sta
```

```
Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scala
r.var_[0])}
------------------------------------------------------------
shape of price_standardized: (109248, 1)
```

In [34]:

```
teacher_number_of_previously_posted_projects_standardized
```

Out[34]:

```
array([[1.90805161],
       [0.1440039 ],
       [0.36000974],
       ...,
       [0.        ],
       [0.03600097],
       [0.07200195]])
```

1.3.3 spilt the data into train ,CV and test

In [35]:

```
# class label
label = project_data['project_is_approved']
project_data.drop(['project_is_approved'], axis=1, inplace=True)
```

In [36]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_
from sklearn.model_selection import train_test_split

# X_train, X_test, y_train, y_test = train_test_split(project_data, label, test_size=0.
X_train, X_test, y_train, y_test = train_test_split(project_data, label, test_size=0.33

X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33) # thi
```

In [37]:

```
print("Shape of X_train and y_train :", X_train.shape, y_train.shape)
print("Shape of X_test and y_test   :", X_test.shape, y_test.shape)
print("Shape of X_cv and y_cv       :", X_cv.shape, y_cv.shape)
```

```
Shape of X_train and y_train : (49041, 19) (49041,)
Shape of X_test and y_test   : (36052, 19) (36052,)
Shape of X_cv and y_cv       : (24155, 19) (24155,)
```

1.4. Vectorizing Categorical data

1.4.1 Vectorization of `project_subject_categories`

- 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 [38]:

```
# 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, b
clean_categories_one_hot_train = vectorizer.fit_transform(X_train['clean_categories'].va
clean_categories_one_hot_test  = vectorizer.transform(X_test['clean_categories'].values
clean_categories_one_hot_cv    = vectorizer.transform(X_cv['clean_categories'].values)
print("vectorizer feature names :", vectorizer.get_feature_names())
print("-------------------------------------------------------")
print("Shape of matrix after one hot encodig train : ",clean_categories_one_hot_train.sh
print("Shape of matrix after one hot encodig test  : ",clean_categories_one_hot_test.sha
print("Shape of matrix after one hot encodig cv    : ",clean_categories_one_hot_cv.shap
```

```
vectorizer feature names : ['History_Civics', 'Care_Hunger', 'Literacy_Lan
guage', 'SpecialNeeds', 'Music_Arts', 'Math_Science', 'Warmth', 'AppliedLe
arning', 'Health_Sports']
-------------------------------------------------------
Shape of matrix after one hot encodig train :  (49041, 9)
Shape of matrix after one hot encodig test  :  (36052, 9)
Shape of matrix after one hot encodig cv    :  (24155, 9)
```

1.4.2 Vectorization of `project_subject_subcategories`

In [39]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=Fals
clean_subcategories_one_hot_train = vectorizer.fit_transform(X_train['clean_subcategori
clean_subcategories_one_hot_test  = vectorizer.transform(X_test['clean_subcategories'].
clean_subcategories_one_hot_cv    = vectorizer.transform(X_cv['clean_subcategories'].val
print("vectorizer feature names :", vectorizer.get_feature_names())
print("-------------------------------------------------------")
print("Shape of matrix after one hot encodig train : ",clean_subcategories_one_hot_trai
print("Shape of matrix after one hot encodig test  : ",clean_subcategories_one_hot_test
print("Shape of matrix after one hot encodig cv    : ",clean_subcategories_one_hot_cv.sh
```

```
vectorizer feature names : ['Economics', 'ParentInvolvement', 'VisualArt
s', 'Warmth', 'ESL', 'CommunityService', 'History_Geography', 'Literacy',
'Literature_Writing', 'Health_Wellness', 'Health_LifeScience', 'Music', 'F
inancialLiteracy', 'ForeignLanguages', 'Mathematics', 'NutritionEducatio
n', 'College_CareerPrep', 'Civics_Government', 'PerformingArts', 'Gym_Fitn
ess', 'TeamSports', 'Care_Hunger', 'Other', 'Extracurricular', 'SocialScie
nces', 'CharacterEducation', 'SpecialNeeds', 'EnvironmentalScience', 'Appl
iedSciences', 'EarlyDevelopment']
-------------------------------------------------------
Shape of matrix after one hot encodig train :  (49041, 30)
Shape of matrix after one hot encodig test  :  (36052, 30)
Shape of matrix after one hot encodig cv    :  (24155, 30)
```

1.4.3 Vectorization of `school_state`

In [40]:

```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(sorted_school_state_dict.keys()), lowercas
school_state_one_hot_train = vectorizer.fit_transform(X_train['school_state'].values)
school_state_one_hot_test  = vectorizer.transform(X_test['school_state'].values)
school_state_one_hot_cv    = vectorizer.transform(X_cv['school_state'].values)
print(vectorizer.get_feature_names())
print("-------------------------------------------------------")
print("Shape of matrix after one hot encodig train : ",school_state_one_hot_train.shape
print("Shape of matrix after one hot encodig test  : ",school_state_one_hot_test.shape)
print("Shape of matrix after one hot encodig cv    : ",school_state_one_hot_cv.shape)
```

```
['AL', 'MT', 'KS', 'CA', 'VT', 'LA', 'DE', 'MO', 'ID', 'NE', 'MI', 'AR',
'WV', 'AK', 'SD', 'WA', 'MN', 'CT', 'MA', 'NY', 'NV', 'RI', 'ND', 'OR', 'H
I', 'UT', 'NJ', 'OH', 'CO', 'TN', 'MD', 'AZ', 'NM', 'VA', 'WY', 'NC', 'P
A', 'IA', 'IL', 'KY', 'NH', 'TX', 'GA', 'SC', 'OK', 'WI', 'IN', 'MS', 'D
C', 'ME', 'FL']
-------------------------------------------------------
Shape of matrix after one hot encodig train :  (49041, 51)
Shape of matrix after one hot encodig test  :  (36052, 51)
Shape of matrix after one hot encodig cv    :  (24155, 51)
```

1.4.4 Vectorization of `teacher_prefix`

In [41]:

```
# we use count vectorizer to convert the values into one hot encoded features
vectorizer = CountVectorizer(vocabulary=list(sorted_teacher_prefix_dict.keys()), lowerc


teacher_prefix_one_hot_train = vectorizer.fit_transform(X_train['teacher_prefix'].value
teacher_prefix_one_hot_test  = vectorizer.transform(X_test['teacher_prefix'].values.asty
teacher_prefix_one_hot_cv    = vectorizer.transform(X_cv['teacher_prefix'].values.astyp
print(vectorizer.get_feature_names())
print("-------------------------------------------------------")
print("Shape of matrix after one hot encodig train : ",teacher_prefix_one_hot_train.sha
print("Shape of matrix after one hot encodig test  : ",teacher_prefix_one_hot_test.shap
print("Shape of matrix after one hot encodig cv    : ",teacher_prefix_one_hot_cv.shape)
```

```
['Ms.', 'Dr.', 'Mrs.', 'Mr.', 'Teacher']
-------------------------------------------------------
Shape of matrix after one hot encodig train :  (49041, 5)
Shape of matrix after one hot encodig test  :  (36052, 5)
Shape of matrix after one hot encodig cv    :  (24155, 5)
```

1.4.5 Vectorization of `project_grade_category`

In [42]:

```
# we use count vectorizer to convert the values into one hot encoded features
vectorizer = CountVectorizer(vocabulary=list(sorted_project_grade_category_dict.keys())

project_grade_category_one_hot_train = vectorizer.fit_transform(X_train['project_grade_
project_grade_category_one_hot_test  = vectorizer.transform(X_test['project_grade_categ
project_grade_category_one_hot_cv    = vectorizer.transform(X_cv['project_grade_categor
print(vectorizer.get_feature_names())
print("-------------------------------------------------------")
print("Shape of matrix after one hot encodig train  : ",project_grade_category_one_hot_
print("Shape of matrix after one hot encodig test   : ",project_grade_category_one_hot_
print("Shape of matrix after one hot encodig cv     : ",project_grade_category_one_hot_
```

```
['GradesPreK-2', 'Grades9-12', 'Grades3-5', 'Grades6-8']
-------------------------------------------------------
Shape of matrix after one hot encodig train  :  (49041, 4)
Shape of matrix after one hot encodig test   :  (36052, 4)
Shape of matrix after one hot encodig cv     :  (24155, 4)
```

1.5. Vectorizing Text

1.5.1 Vectorization of `essays bow`

In [43]:

```
X_train['essay'].tail(1)
```

Out[43]:

```
30825    My students live in Jackson Heights, Queens, o...
Name: essay, dtype: object
```

In [44]:

```
# we are considering only the words which appeared in at least 10 documents (rows or pr
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(preprocessed_essays, min_df=10, )

essays_bow_train = vectorizer.fit_transform(X_train['essay'].values)
essays_bow_test  = vectorizer.transform(X_test['essay'].values)
essays_bow_cv    = vectorizer.transform(X_cv['essay'].values)

print("-------------------------------------------------------")
print("Shape of matrix after one hot encodig train : ",essays_bow_train.shape)
print("Shape of matrix after one hot encodig test  : ",essays_bow_test.shape)
print("Shape of matrix after one hot encodig cv    : ",essays_bow_cv.shape)
```

```
-------------------------------------------------------
Shape of matrix after one hot encodig train :  (49041, 12544)
Shape of matrix after one hot encodig test  :  (36052, 12544)
Shape of matrix after one hot encodig cv    :  (24155, 12544)
```

1.5.1.1 Vectorization of `essays tfidf`

In [45]:

```
# we are considering only the words which appeared in at least 10 documents (rows or pr
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(preprocessed_essays, min_df=10)

essays_tfidf_train = vectorizer.fit_transform(X_train['essay'].values)
essays_tfidf_test  = vectorizer.transform(X_test['essay'].values)
essays_tfidf_cv    = vectorizer.transform(X_cv['essay'].values)

print("Shape of matrix after one hot encodig of train : ",essays_tfidf_train.shape)
print("Shape of matrix after one hot encodig test     : ",essays_tfidf_test.shape)
print("Shape of matrix after one hot encodig cv       : ",essays_tfidf_cv.shape)
```

```
Shape of matrix after one hot encodig of train :  (49041, 12544)
Shape of matrix after one hot encodig test     :  (36052, 12544)
Shape of matrix after one hot encodig cv       :  (24155, 12544)
```

1.5.1.2 Using Pretrained Models: essays Avg W2V

In [46]:

```
'''
# 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_essays:
    words.extend(i.split(' '))

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

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

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


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

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


'''
```

Out[46]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/382303
```

```
49/4084039\ndef (https://stackoverflow.com/a/38230349/4084039\ndef) load
GloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open
(gloveFile,\'r\', encoding="utf8")\n    model = {}\n    for line in tqdm
(f):\n        splitLine = line.split()\n        word = splitLine[0]\n
embedding = np.array([float(val) for val in splitLine[1:]])\n        mod
el[word] = embedding\n    print ("Done.",len(model)," words loaded!")\n
return model\nmodel = loadGloveModel(\'glove.42B.300d.txt\')\n\n# ======
======================\nOutput:\n    \nLoading Glove Model\n1917495it [0
6:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n# ================
==========\n\nwords = []\nfor i in preproced_essays:\n    words.extend
(i.split(\' \'))\n\nfor i in preprocessed_project_title:\n    words.exte
nd(i.split(\' \'))\nprint("all the words in the coupus", len(words))\nwo
rds = set(words)\nprint("the unique words in the coupus", len(words))\n
\ninter_words = set(model.keys()).intersection(words)\nprint("The number
of words that are present in both glove vectors and our coupus",       l
en(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")\n
\nwords_courpus = {}\nwords_glove = set(model.keys())\nfor i in words:\n
if i in words_glove:\n        words_courpus[i] = model[i]\nprint("word 2
vec length", len(words_courpus))\n\n\n# stronging variables into pickle
 files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-
load-variables-in-python/\n\nimport (http://www.jessicayung.com/how-to-u
se-pickle-to-save-and-load-variables-in-python/\n\nimport) pickle\nwith
 open(\'glove_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus,
 f)\n\n\n'
```

In [47]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-p
# 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 [48]:

```
# average Word2Vec X_train
# compute average word2vec for each review.
essays_avg_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in
for sentence in tqdm(X_train['essay']): # 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
    essays_avg_w2v_vectors_train.append(vector)

print(len(essays_avg_w2v_vectors_train))
print(len(essays_avg_w2v_vectors_train[0]))
```

```
100%|██████████| 49041/49041 [00:28<00:00, 1708.72it/s]

49041
300
```

Average Word2Vec X_test_essay

In [49]:

```
# average Word2Vec X_test_essay
# compute average word2vec for each review.
essays_avg_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in t
for sentence in tqdm(X_test['essay']): # 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
    essays_avg_w2v_vectors_test.append(vector)

print(len(essays_avg_w2v_vectors_test))
print(len(essays_avg_w2v_vectors_test[0]))
```

```
100%|██████████| 36052/36052 [00:21<00:00, 1710.17it/s]

36052
300
```

Average Word2Vec X_cv_essay

In [50]:

```
# average Word2Vec X_cv
# compute average word2vec for each review.
essays_avg_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in thi
for sentence in tqdm(X_cv['essay']): # 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
    essays_avg_w2v_vectors_cv.append(vector)

print(len(essays_avg_w2v_vectors_cv))
print(len(essays_avg_w2v_vectors_cv[0]))
```

```
100%|██████████| 24155/24155 [00:14<00:00, 1725.22it/s]

24155
300
```

1.5.1.3 essays TFIDF weighted W2V train

In [51]:

```
tfidf_model_preprocessed_essays_train = TfidfVectorizer()
tfidf_model_preprocessed_essays_train.fit(preprocessed_essays)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_preprocessed_essays_train.get_feature_names(), list(t
tfidf_words = set(tfidf_model_preprocessed_essays_train.get_feature_names())
```

In [52]:

```
# essays TFIDF weighted W2V_train
# compute average word2vec for each review.
preprocessed_essays_train_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/revie
for sentence in tqdm(X_train['essay']): # 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
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    preprocessed_essays_train_tfidf_w2v_vectors.append(vector)

print(len(preprocessed_essays_train_tfidf_w2v_vectors))
print(len(preprocessed_essays_train_tfidf_w2v_vectors[0]))
```

```
100%|██████████| 49041/49041 [02:03<00:00, 396.69it/s]

49041
300
```

essays TFIDF weighted W2V test

In [53]:

```
# tfidf_model_preprocessed_essays_test
tfidf_model_preprocessed_essays_test = TfidfVectorizer()
tfidf_model_preprocessed_essays_test.fit(preprocessed_essays)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_preprocessed_essays_test.get_feature_names(), list(tf
tfidf_words = set(tfidf_model_preprocessed_essays_test.get_feature_names())
```

In [54]:

```
# tfidf_model_preprocessed_essays_test
# compute average word2vec for each review.
preprocessed_essays_test_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review
for sentence in tqdm(X_test['essay']): # 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
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    preprocessed_essays_test_tfidf_w2v_vectors.append(vector)

print(len(preprocessed_essays_test_tfidf_w2v_vectors))
print(len(preprocessed_essays_test_tfidf_w2v_vectors[0]))
```

```
100%|██████████| 36052/36052 [01:26<00:00, 415.39it/s]

36052
300
```

essays TFIDF weighted W2V cv

In [55]:

```
# tfidf_model_preprocessed_essays_cv
tfidf_model_preprocessed_essays_cv = TfidfVectorizer()
tfidf_model_preprocessed_essays_cv.fit(preprocessed_essays)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_preprocessed_essays_cv.get_feature_names(), list(tfid
tfidf_words = set(tfidf_model_preprocessed_essays_cv.get_feature_names())
```

In [56]:

```
# tfidf_model_preprocessed_essays_cv
# compute average word2vec for each review.
preprocessed_essays_cv_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review i
for sentence in tqdm(X_cv['essay']): # 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
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    preprocessed_essays_cv_tfidf_w2v_vectors.append(vector)

print(len(preprocessed_essays_cv_tfidf_w2v_vectors))
print(len(preprocessed_essays_cv_tfidf_w2v_vectors[0]))
```

```
100%|██████████| 24155/24155 [00:55<00:00, 431.49it/s]

24155
300
```

1.5.2 Vectorization of `project_title` bow train, test, cv

In [57]:

```
X_train['project_title'].head(1)
```

Out[57]:

```
15473    The Key to Success is Order!!
Name: project_title, dtype: object
```

In [58]:

```
# we are considering only the words which appeared in at least 10 documents (rows or pr
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(preprocessed_project_title, min_df=10)

project_title_bow_train = vectorizer.fit_transform(X_train['project_title'].values)
project_title_bow_test  = vectorizer.transform(X_test['project_title'].values)
project_title_bow_cv    = vectorizer.transform(X_cv['project_title'].values)

print("------------------------------------------------------------")
print("Shape of matrix after one hot encodig train : ",project_title_bow_train.shape)
print("Shape of matrix after one hot encodig test  : ",project_title_bow_test.shape)
print("Shape of matrix after one hot encodig cv    : ",project_title_bow_cv.shape)
```

```
------------------------------------------------------------
Shape of matrix after one hot encodig train :  (49041, 2087)
Shape of matrix after one hot encodig test  :  (36052, 2087)
Shape of matrix after one hot encodig cv    :  (24155, 2087)
```

1.5.2.1 Vectorization of `project_title tfidf` train, test. cv

In [59]:

```
# we are considering only the words which appeared in at least 10 documents (rows or pr
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(preprocessed_project_title, min_df=10)

project_title_tfidf_train = vectorizer.fit_transform(X_train['project_title'].values)
project_title_tfidf_test  = vectorizer.transform(X_test['project_title'].values)
project_title_tfidf_cv    = vectorizer.transform(X_cv['project_title'].values)

print("Shape of matrix after one hot encodig of train : ",project_title_tfidf_train.sha
print("Shape of matrix after one hot encodig test     : ",project_title_tfidf_test.shap
print("Shape of matrix after one hot encodig cv       : ",project_title_tfidf_cv.shape)
```

```
Shape of matrix after one hot encodig of train :  (49041, 2087)
Shape of matrix after one hot encodig test     :  (36052, 2087)
Shape of matrix after one hot encodig cv       :  (24155, 2087)
```

1.5.2.2 Using Pretrained Models: `project_title Avg W2V` train

In [60]:

```
'''
# 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_essays:
    words.extend(i.split(' '))

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

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

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


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

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


'''
```

Out[60]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/3823034
```

```
9/4084039\ndef (https://stackoverflow.com/a/38230349/4084039\ndef) loadGlo
veModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(glove
File,\'r\', encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n
splitLine = line.split()\n        word = splitLine[0]\n        embedding =
np.array([float(val) for val in splitLine[1:]])\n        model[word] = emb
edding\n    print ("Done.",len(model)," words loaded!")\n    return model
\nmodel = loadGloveModel(\'glove.42B.300d.txt\')\n\n# ==================
========\nOutput:\n    \nLoading Glove Model\n1917495it [06:32, 4879.69it/
s]\nDone. 1917495  words loaded!\n\n# ============================\n\nword
s = []\nfor i in preproced_essays:\n    words.extend(i.split(\' \'))\n\nfo
r i in preprocessed_project_title:\n    words.extend(i.split(\' \'))\nprin
t("all the words in the coupus", len(words))\nwords = set(words)\nprint("t
he unique words in the coupus", len(words))\n\ninter_words = set(model.key
s()).intersection(words)\nprint("The number of words that are present in b
oth glove vectors and our coupus",       len(inter_words),"(",np.round(len
(inter_words)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove =
set(model.keys())\nfor i in words:\n    if i in words_glove:\n        word
s_courpus[i] = model[i]\nprint("word 2 vec length", len(words_courpus))\n
\n\n# stronging variables into pickle files python: http://www.jessicayun
g.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport (h
ttp://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-
python/\n\nimport) pickle\nwith open(\'glove_vectors\', \'wb\') as f:\n
pickle.dump(words_courpus, f)\n\n\n'
```

In [61]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-p
# 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 [62]:

```
# average Word2Vec  project_title_train
# compute average word2vec for each review.
project_title_avg_w2v_vectors_train = []; # the avg-w2v for each sentence/review is sto
for sentence in tqdm(X_train['project_title']): # 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
    project_title_avg_w2v_vectors_train.append(vector)

print(len(project_title_avg_w2v_vectors_train))
print(len(project_title_avg_w2v_vectors_train[0]))
```

```
100%|██████████| 49041/49041 [00:00<00:00, 98232.50it/s]

49041
300
```

Average Word2Vec project_title_test

In [63]:

```
# average Word2Vec  project_title_test
# compute average word2vec for each review.
project_title_avg_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stor
for sentence in tqdm(X_test['project_title']): # 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
    project_title_avg_w2v_vectors_test.append(vector)

print(len(project_title_avg_w2v_vectors_test))
print(len(project_title_avg_w2v_vectors_test[0]))
```

```
100%|██████████| 36052/36052 [00:00<00:00, 85190.40it/s]

36052
300
```

Average Word2Vec project_title_cv

In [64]:

```
# average Word2Vec  project_title_ cv
# compute average word2vec for each review.
project_title_avg_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored
for sentence in tqdm(X_cv['project_title']): # 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
    project_title_avg_w2v_vectors_cv.append(vector)

print(len(project_title_avg_w2v_vectors_cv))
print(len(project_title_avg_w2v_vectors_cv[0]))
```

```
100%|██████████| 24155/24155 [00:00<00:00, 85693.61it/s]

24155
300
```

1.5.2.3 project_title TFIDF weighted W2V train

In [65]:

```
tfidf_model_preprocessed_project_title_train = TfidfVectorizer()
tfidf_model_preprocessed_project_title_train.fit(preprocessed_project_title)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_preprocessed_project_title_train.get_feature_names(),
tfidf_words = set(tfidf_model_preprocessed_project_title_train.get_feature_names())
```

In [66]:

```
# project_title TFIDF weighted W2V train
# compute average word2vec for each review.
preprocessed_project_title_train_tfidf_w2v_vectors = []; # the avg-w2v for each sentenc
for sentence in tqdm(X_train['project_title']): # 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
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    preprocessed_project_title_train_tfidf_w2v_vectors.append(vector)

print(len(preprocessed_project_title_train_tfidf_w2v_vectors))
print(len(preprocessed_project_title_train_tfidf_w2v_vectors[0]))
```

```
100%|██████████| 49041/49041 [00:00<00:00, 61626.31it/s]

49041
300
```

project_title TFIDF weighted W2V_ test

In [67]:

```
# tfidf_model_preprocessed_project_title_test
tfidf_model_preprocessed_project_title_test = TfidfVectorizer()
tfidf_model_preprocessed_project_title_test.fit(preprocessed_project_title)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_preprocessed_project_title_test.get_feature_names(),
tfidf_words = set(tfidf_model_preprocessed_project_title_test.get_feature_names())
```

In [68]:

```
# project_title TFIDF weighted W2V_ test
# compute average word2vec for each review.
preprocessed_project_title_test_tfidf_w2v_vectors = []; # the avg-w2v for each sentence,
for sentence in tqdm(X_test['project_title']): # 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
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    preprocessed_project_title_test_tfidf_w2v_vectors.append(vector)

print(len(preprocessed_project_title_test_tfidf_w2v_vectors))
print(len(preprocessed_project_title_test_tfidf_w2v_vectors[0]))
```

```
100%|██████████| 36052/36052 [00:00<00:00, 57048.72it/s]

36052
300
```

project_title TFIDF weighted W2V_cv

In [69]:

```
# tfidf_model_preprocessed_project_title_cv
tfidf_model_preprocessed_project_title_cv = TfidfVectorizer()
tfidf_model_preprocessed_project_title_cv.fit(preprocessed_project_title)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_preprocessed_essays_cv.get_feature_names(), list(tfid
tfidf_words = set(tfidf_model_preprocessed_essays_cv.get_feature_names())
```

In [70]:

```
# project_title TFIDF weighted W2V_cv
# compute average word2vec for each review.
preprocessed_project_title_cv_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/r
for sentence in tqdm(X_cv['project_title']): # 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
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    preprocessed_project_title_cv_tfidf_w2v_vectors.append(vector)

print(len(preprocessed_project_title_cv_tfidf_w2v_vectors))
print(len(preprocessed_project_title_cv_tfidf_w2v_vectors[0]))
```

```
100%|██████████| 24155/24155 [00:00<00:00, 76658.62it/s]

24155
300
```

1.6. Vectorizing Numerical features

1.6.1 standardization of price_train

In [71]:

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

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

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

# Now standardize the data with above maen and variance.
price_standardized_train = price_scalar.transform(X_train['price'].values.reshape(-1, 1
print("Shape of price_standardized_train: ", price_standardized_train.shape)
```

```
Mean: 299.52416263942416 Standard deviation: 368.5676343977104
Shape of price_standardized_train:  (49041, 1)
```

1.6.2 standardization of price_test

In [72]:

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

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

price_scalar = StandardScaler()
# Now standardize the data with  maen and variance.
price_scalar.fit(X_train['price'].values.reshape(-1,1))

price_standardized_test = price_scalar.transform(X_test['price'].values.reshape(-1, 1))
print("Mean:",price_scalar.mean_[0],"Standard deviation:",np.sqrt(price_scalar.var_[0]))
price_standardized_test.shape
```

```
Mean: 299.52416263942416 Standard deviation: 368.5676343977104
```

Out[72]:

```
(36052, 1)
```

1.6.3 standardization of price_cv

In [73]:

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

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

price_scalar = StandardScaler()
# Now standardize the data with  maen and variance.
price_scalar.fit(X_train['price'].values.reshape(-1,1))
price_standardized_cv = price_scalar.fit_transform(X_cv['price'].values.reshape(-1, 1))
print("Mean:",price_scalar.mean_[0],"Standard deviation:",np.sqrt(price_scalar.var_[0]))
print("Shape of price_standardized_cv: ", price_standardized_cv.shape)
```

```
Mean: 294.53515628234317 Standard deviation: 367.9790662060577
Shape of price_standardized_cv:  (24155, 1)
```

1.6.4 Teacher_number_of_previously_posted_projects_train : Numerical / Standardize

In [74]:

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

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

teacher_number_of_previously_posted_projects_scalar = StandardScaler()
# Now standardize the data with  maen and variance.
teacher_number_of_previously_posted_projects_standardized_train = teacher_number_of_prev
print("Mean :",teacher_number_of_previously_posted_projects_scalar.mean_[0], "Standard
teacher_number_of_previously_posted_projects_standardized_train = teacher_number_of_prev
print("Shape of teacher_number_of_previously_posted_projects_standardized_train: ", tea
```

```
Mean : 11.201015476845905 Standard deviation : 27.846829997968037
Shape of teacher_number_of_previously_posted_projects_standardized_train:
(49041, 1)
```

1.6.5 Teacher_number_of_previously_posted_projects_test : Numerical / Standardize

In [75]:

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

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

teacher_number_of_previously_posted_projects_scalar = StandardScaler()
# Now standardize the data with  maen and variance.
teacher_number_of_previously_posted_projects_scalar.fit(X_train['teacher_number_of_prev
teacher_number_of_previously_posted_projects_standardized_test = teacher_number_of_prev
print("Mean :", teacher_number_of_previously_posted_projects_scalar.mean_[0], "Standard

print("Shape of teacher_number_of_previously_posted_projects_standardized_test: ", teac
```

```
Mean : 11.201015476845905 Standard deviation :  27.846829997968037
Shape of teacher_number_of_previously_posted_projects_standardized_test:
(36052, 1)
```

1.6.6 Teacher_number_of_previously_posted_projects_cv : Numerical / Standardize

In [76]:

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

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

teacher_number_of_previously_posted_projects_scalar = StandardScaler()
# Now standardize the data with  maen and variance.
teacher_number_of_previously_posted_projects_scalar.fit(X_train['teacher_number_of_prev
teacher_number_of_previously_posted_projects_standardized_cv = teacher_number_of_previo
print("Mean :", teacher_number_of_previously_posted_projects_scalar.mean_[0], "Standard
print("Shape of teacher_number_of_previously_posted_projects_standardized_cv: ", teache
```

```
Mean : 11.201015476845905 Standard deviation : 27.846829997968037
Shape of teacher_number_of_previously_posted_projects_standardized_cv:  (2
4155, 1)
```

In [77]:

```
project_data.columns
```

Out[77]:

```
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', 'clean_categories',
       'clean_subcategories', 'essay', 'quantity', 'price'],
      dtype='object')
```

we are going to consider

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

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

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

Assignment : Apply KNN

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

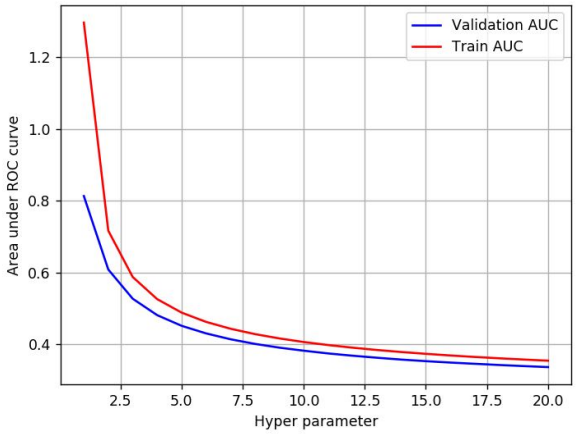
- Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)
- Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)
- Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)
- Set 4: categorical, numerical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

2. **Hyper paramter tuning to find best K**

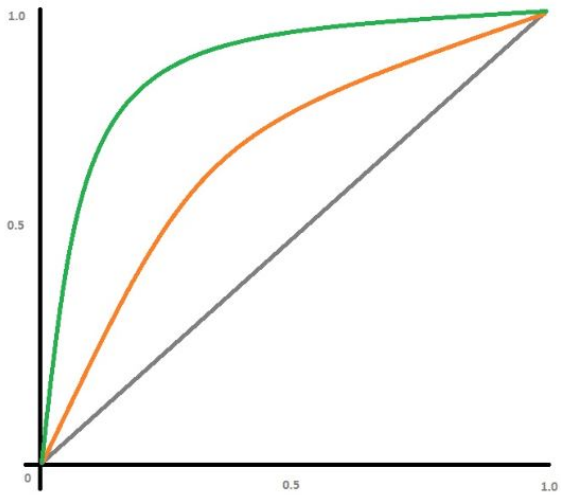
- Find the best hyper parameter which results in the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
- Find the best hyper paramter using k-fold cross validation (or) simple cross validation data
- 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.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points

	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

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/)

Vectorizer	Model	Hyper parameter	AUC
BOW	Brute	7	0.78
TFIDF	Brute	12	0.79
W2V	Brute	10	0.78
TFIDFW2V	Brute	6	0.78

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)
  ```

2. K Nearest Neighbor

2.1 Applying KNN brute force on BOW, SET 1

Merging features encoding numerical + categorical features BOW, SET 1

In [78]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
# set1_ = all categorical features + numarical features + essays_bow + project_title_bo

set1_train = hstack((clean_categories_one_hot_train, clean_subcategories_one_hot_train,
                     teacher_prefix_one_hot_train,project_grade_category_one_hot_train,
                     teacher_number_of_previously_posted_projects_standardized_train,
                    essays_bow_train,price_standardized_train)).tocsr()

set1_test = hstack((clean_categories_one_hot_test, clean_subcategories_one_hot_test, sc
                     teacher_prefix_one_hot_test,project_grade_category_one_hot_test, p
                     teacher_number_of_previously_posted_projects_standardized_test,
                    essays_bow_test,price_standardized_test)).tocsr()

set1_cv = hstack((clean_categories_one_hot_cv, clean_subcategories_one_hot_cv, school_s
                     teacher_prefix_one_hot_cv,project_grade_category_one_hot_cv, proje
                     teacher_number_of_previously_posted_projects_standardized_cv,
                    essays_bow_cv,price_standardized_cv)).tocsr()
```

In [79]:

```
print("Final Data Matrix of set1 :")
print("shape of set1_train and y_train :", set1_train.shape , y_train.shape)
print("shape of set1_test and y_test   :", set1_test.shape , y_test.shape)
print("shape of set1_cv and y_cv       :", set1_cv.shape , y_cv.shape)
```

```
Final Data Matrix of set1 :
shape of set1_train and y_train : (49041, 14732) (49041,)
shape of set1_test and y_test   : (36052, 14732) (36052,)
shape of set1_cv and y_cv       : (24155, 14732) (24155,)
```

2.1.1 A. Hyper parameter Tuning to find bestK: Simple cross Validation set1

In [80]:

```
%%time
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

for i in range(1,96,5):
    # instantiate learning model (k = 96)
    knn = KNeighborsClassifier(n_neighbors=i, algorithm='brute')

    # fitting the model on crossvalidation train
    knn.fit(set1_train, y_train)

    # predict the response on the crossvalidation train
    pred = knn.predict(set1_cv)

    # evaluate CV accuracy
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    print('\nCV accuracy for k = %d is %d%%' % (i, acc))
```

```
CV accuracy for k = 1 is 74%

CV accuracy for k = 6 is 80%

CV accuracy for k = 11 is 84%

CV accuracy for k = 16 is 84%

CV accuracy for k = 21 is 85%

CV accuracy for k = 26 is 85%

CV accuracy for k = 31 is 85%

CV accuracy for k = 36 is 85%

CV accuracy for k = 41 is 85%

CV accuracy for k = 46 is 85%

CV accuracy for k = 51 is 85%

CV accuracy for k = 56 is 85%

CV accuracy for k = 61 is 85%

CV accuracy for k = 66 is 85%

CV accuracy for k = 71 is 85%

CV accuracy for k = 76 is 85%

CV accuracy for k = 81 is 85%

CV accuracy for k = 86 is 85%

CV accuracy for k = 91 is 85%
```

```
CPU times: user 1h 11min 29s, sys: 6min 58s, total: 1h 18min 28s
Wall time: 1h 18min 29s
```

In [81]:

```
# Acc in test model

knn = KNeighborsClassifier(51)
knn.fit(set1_train,y_train)
pred = knn.predict(set1_test)
acc = accuracy_score(y_test, pred, normalize=True) * float(100)
print('\n****Test accuracy for k = 51 is %d%%' % (acc))
```

```
****Test accuracy for k = 51 is 84%
```

2.1.1 B. Hyper parameter Tuning to find bestK:: Simple for loop

In [82]:

```
%%time

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

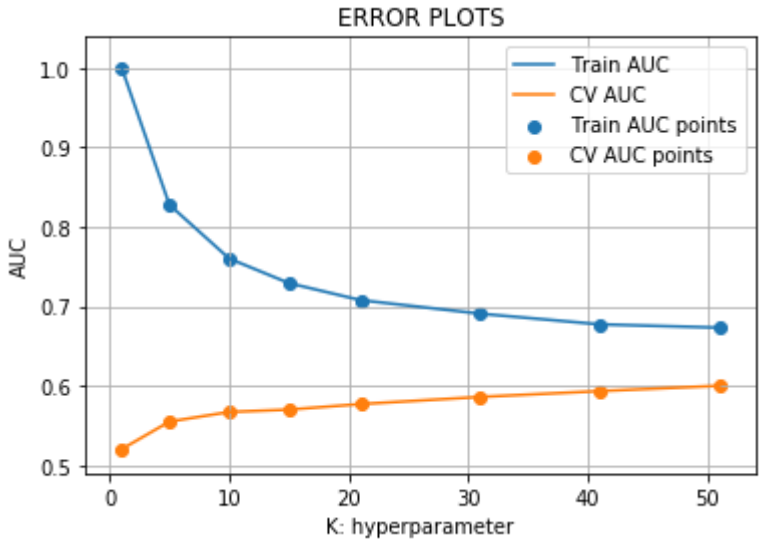
y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence va
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute')
    neigh.fit(set1_train, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(set1_train)[:,1]
    y_cv_pred =  neigh.predict_proba(set1_cv)[:,1]

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')
plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```




```
CPU times: user 1h 29min 55s, sys: 8min 53s, total: 1h 38min 49s
```

```
Wall time: 1h 38min 50s
```

In [83]:

```
print(neigh.get_params)
```

```
<bound method BaseEstimator.get_params of KNeighborsClassifier(algorithm
='brute', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=51, p=2,
                     weights='uniform')>
```

In [84]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and ga
# besed on the GridsearchCV method i have choosen best k is 51.

best_k = 51
```

2.1.2Train model using the best hyper-parameter value set1

In [85]:

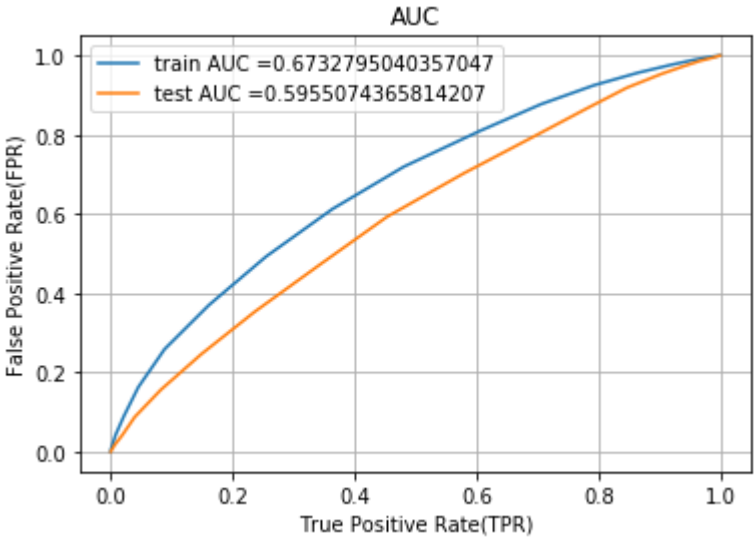
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklea
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k,algorithm='brute')
neigh.fit(set1_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of tr
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(set1_train)[:
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(set1_test)[:,1])

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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



2.1.3 Confustion Matrix set1_train and set1_test

In [87]:

```
def predict(proba, threshould, fpr, tpr):
    t = threshould[np.argmax(fpr*(1-tpr))]

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

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.rou

    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [88]:

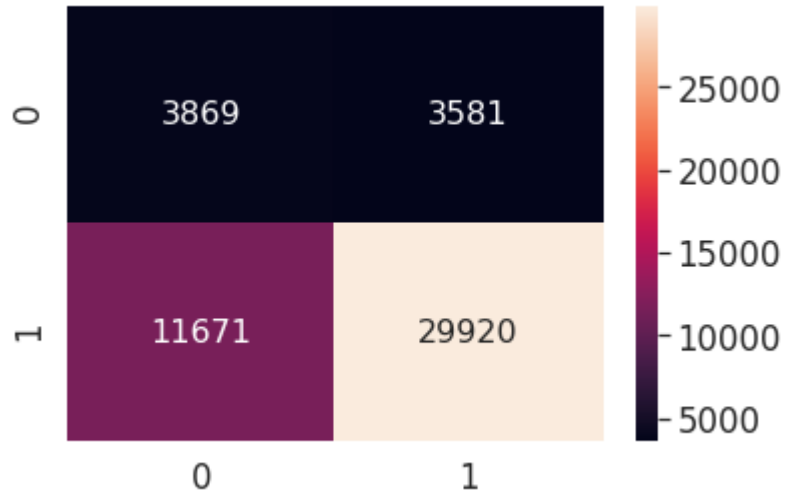
```
#Confustion Matrix Train Set1
conf_matr_df_train = pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred,thresho
train_fpr, train_fpr)), range(2),range(2))

sns.set(font_scale=1.5)#for label size
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24962639520742305 for threshold 0.804
```

Out[88]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f57229a1860>
```



In [119]:

```
# confustion matrix test set1

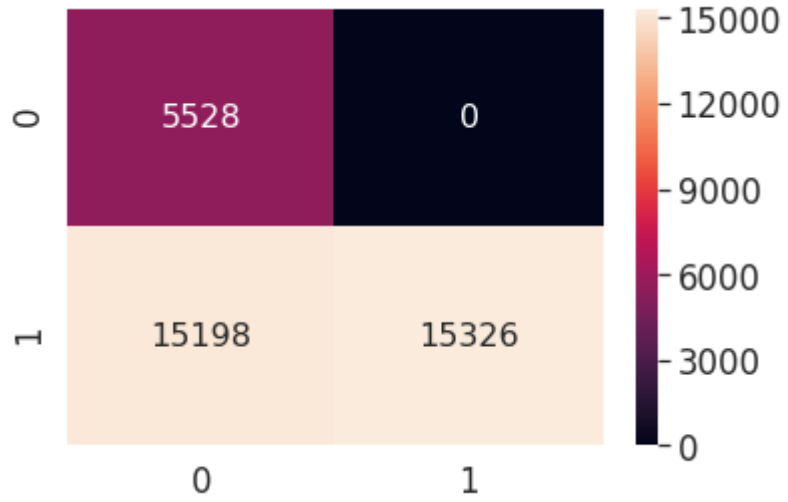
neigh.fit(set1_test,y_test)
y_test_pred =  neigh.predict_proba(set1_test)[:,1]
conf_matr_df_test = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred,thresholds


sns.set(font_scale=1.5)
sns.heatmap(conf_matr_df_test,annot=True,annot_kws={"size":16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24933083070321124 for threshold 0.835
```

Out[119]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5714570908>
```



2.2 Applying KNN brute force on TFIDF, SET 2

Merging features encoding numerical + categorical features tfidf, SET 2

In [89]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
# set2_ = all categorical features + numarical features + essays_tfidf_train + project_

set2_train = hstack((clean_categories_one_hot_train, clean_subcategories_one_hot_train,
                     teacher_prefix_one_hot_train,project_grade_category_one_hot_train,
                     teacher_number_of_previously_posted_projects_standardized_train,
                    essays_tfidf_train,price_standardized_train)).tocsr()

set2_test = hstack((clean_categories_one_hot_test, clean_subcategories_one_hot_test, scl
                     teacher_prefix_one_hot_test,project_grade_category_one_hot_test, pr
                     teacher_number_of_previously_posted_projects_standardized_test,
                    essays_tfidf_test,price_standardized_test)).tocsr()

set2_cv = hstack((clean_categories_one_hot_cv, clean_subcategories_one_hot_cv, school_s
                     teacher_prefix_one_hot_cv,project_grade_category_one_hot_cv, proje
                     teacher_number_of_previously_posted_projects_standardized_cv,
                    essays_tfidf_cv,price_standardized_cv)).tocsr()
```

In [90]:

```
print("Final Data Matrix of set2 :")
print("shape of set2_train and y_train :", set2_train.shape , y_train.shape)
print("shape of set2_test and y_test   :", set2_test.shape , y_test.shape)
print("shape of set2_cv and y_cv       :", set2_cv.shape , y_cv.shape)
```

```
Final Data Matrix of set2 :
shape of set2_train and y_train : (49041, 14732) (49041,)
shape of set2_test and y_test   : (36052, 14732) (36052,)
shape of set2_cv and y_cv       : (24155, 14732) (24155,)
```

2.2.1 Hyper parameter Tuning to find bestK:: GridSearchcv

In [91]:

```
%%time
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchC
from sklearn.model_selection import GridSearchCV

neigh = KNeighborsClassifier( algorithm='brute')
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51, 65, 71, 85, 91]}
clf = GridSearchCV(neigh, parameters, cv=3,scoring='roc_auc', return_train_score=True)
clf.fit(set2_train, y_train)

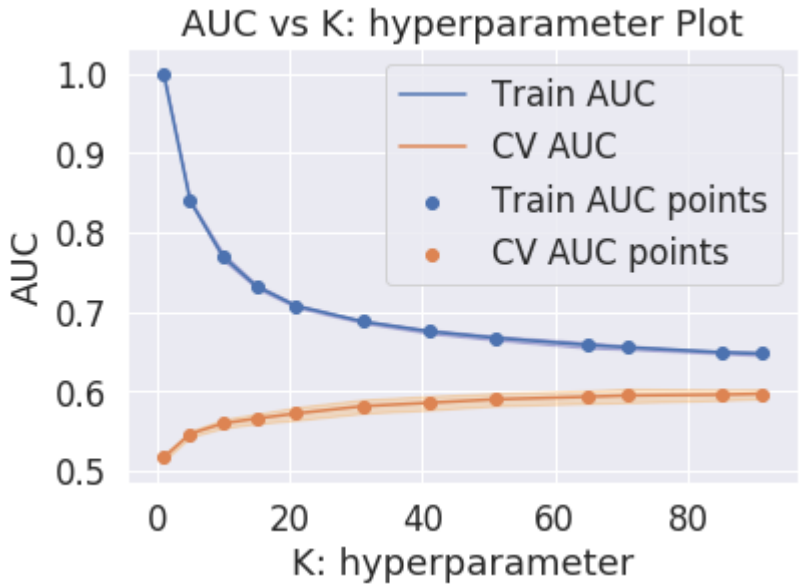
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_auc +

plt.plot(parameters['n_neighbors'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],cv_auc - cv_auc_std,cv_auc + cv_auc_st

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs K: hyperparameter Plot")
plt.show()
```




```
CPU times: user 2h 57min 23s, sys: 17min 22s, total: 3h 14min 45s
Wall time: 3h 14min 48s
```

In [92]:

```
print(clf.get_params)
```

```
<bound method BaseEstimator.get_params of GridSearchCV(cv=3, error_score
='raise-deprecating',
             estimator=KNeighborsClassifier(algorithm='brute', leaf_size=3
0,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=Non
e,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='warn', n_jobs=None,
             param_grid={'n_neighbors': [1, 5, 10, 15, 21, 31, 41, 51, 65,
71,
                                         85, 91]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)>
```

In [176]:

```
print(neigh.get_params)
```

```
<bound method BaseEstimator.get_params of KNeighborsClassifier(algorithm
='brute', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=85, p=2,
                     weights='uniform')>
```

In [177]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and ga
# besed on the GridsearchCV method i have choosen best k is 85.

best_k_set2 = 85
```

2.2.2 Train model using the best hyper-parameter value set2

In [95]:

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

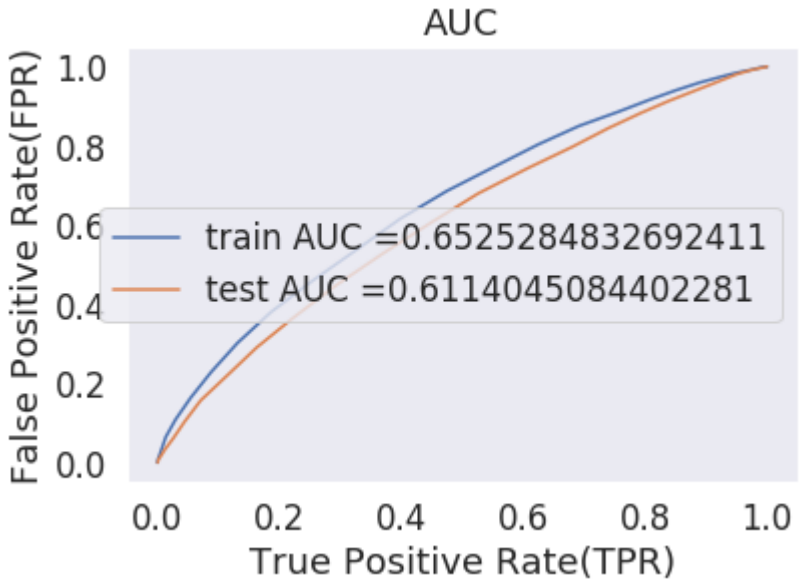
neigh = KNeighborsClassifier(n_neighbors=best_k_set2,algorithm='brute')
neigh.fit(set2_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(set2_train)[:
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(set2_test)[:,1])



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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



2.2.3 Confustion Matrix set2_train and set2_test

In [96]:

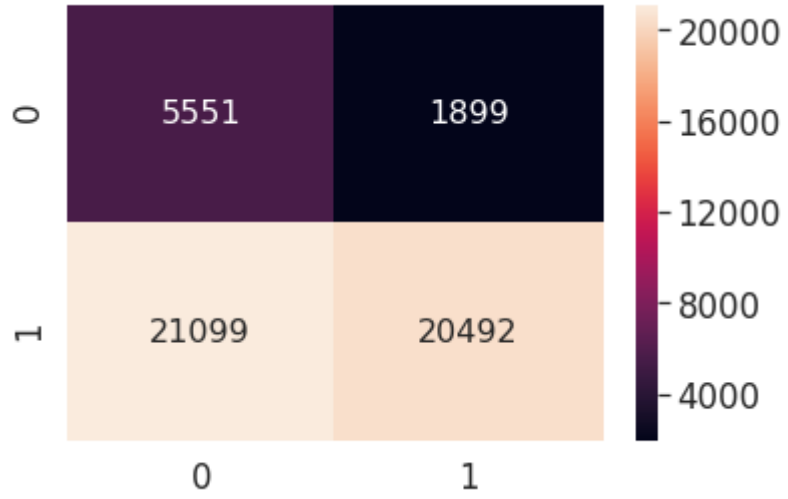
```
#Confustion Matrix Train Set2_train
conf_matr_df_train_2 = pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred,thresl
train_fpr, train_fpr)), range(2),range(2))

sns.set(font_scale=1.5)#for label size
sns.heatmap(conf_matr_df_train_2, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24934271429214902 for threshold 0.835
```

Out[96]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f571c5f1ba8>
```



In [121]:

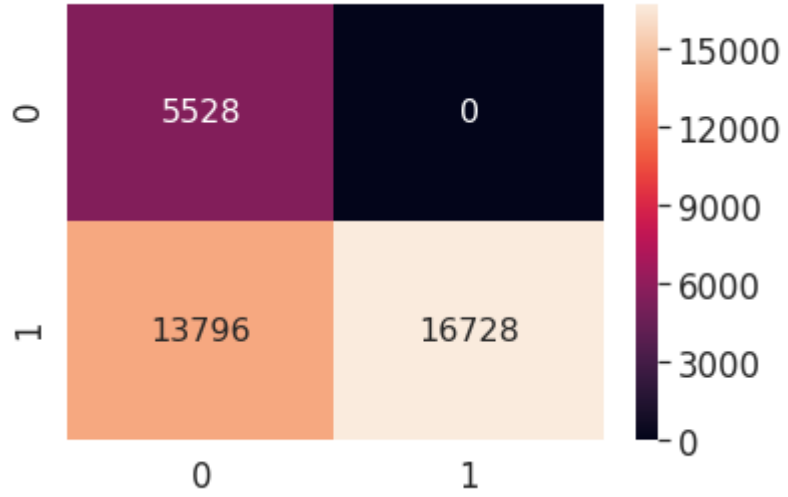
```
#Confustion Matrix Test Set2_test
neigh.fit(set2_test,y_test)
y_test_pred_2 =  neigh.predict_proba(set2_test)[:,1]
conf_matr_df_test_2 = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred_2,thresh


sns.set(font_scale=1.5)
sns.heatmap(conf_matr_df_test_2,annot=True,annot_kws={"size":16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24933083070321124 for threshold 0.835
```

Out[121]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5714543278>
```



2.3 Applying KNN brute on AVG W2V, SET 3

In [122]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
# set3_ = all categorical features + numarical features + essays_avg_w2v_vectors projec


set3_train = hstack((clean_categories_one_hot_train, clean_subcategories_one_hot_train,
                     teacher_prefix_one_hot_train,project_grade_category_one_hot_train,
                     teacher_number_of_previously_posted_projects_standardized_train,es
                     project_title_avg_w2v_vectors_train,price_standardized_train)).toc

set3_test = hstack((clean_categories_one_hot_test, clean_subcategories_one_hot_test, scl
                     teacher_prefix_one_hot_test,project_grade_category_one_hot_test, e
                    teacher_number_of_previously_posted_projects_standardized_test,
                    project_title_avg_w2v_vectors_test, price_standardized_test)).tocsr

set3_cv = hstack((clean_categories_one_hot_cv, clean_subcategories_one_hot_cv, school_s
                     teacher_prefix_one_hot_cv,project_grade_category_one_hot_cv,
                     teacher_number_of_previously_posted_projects_standardized_cv,
                    essays_avg_w2v_vectors_cv, project_title_avg_w2v_vectors_cv,price_s
```

In [123]:

```
print("Final Data Matrix of set4 :")
print("shape of set3_train and y_train :", set3_train.shape , y_train.shape)
print("shape of set3_test and y_test   :", set3_test.shape , y_test.shape)
print("shape of set3_cv and y_cv       :", set3_cv.shape , y_cv.shape)
```

```
Final Data Matrix of set4 :
shape of set3_train and y_train : (49041, 701) (49041,)
shape of set3_test and y_test   : (36052, 701) (36052,)
shape of set3_cv and y_cv       : (24155, 701) (24155,)
```

In [124]:

```
type(set3_train.dtype)
```

Out[124]:

```
numpy.dtype
```

2.3.1 Hyper parameter Tuning to find bestK:: Simple for loop set3

In [126]:

```
%%time

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

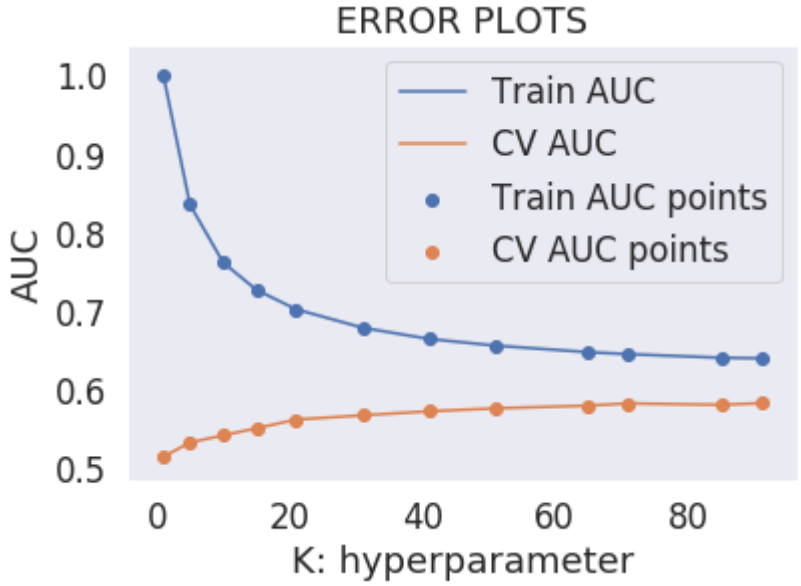
y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence va
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51, 65, 71, 85, 91]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute')
    neigh.fit(set3_train, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(set3_train)[:,1]
    y_cv_pred    =  neigh.predict_proba(set3_cv)[:,1]

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')
plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



```
CPU times: user 18h 29min 33s, sys: 16min 25s, total: 18h 45min 59s
Wall time: 18h 46min 30s
```

In [178]:

```
print(clf.get_params)
```

```
<bound method BaseEstimator.get_params of GridSearchCV(cv=3, error_score
='raise-deprecating',
             estimator=KNeighborsClassifier(algorithm='brute', leaf_size=3
0,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=Non
e,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='warn', n_jobs=None,
             param_grid={'n_neighbors': [1, 5, 10, 15, 21, 31, 41, 51, 65,
71,
                                         85, 91]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)>
```

In [130]:

```
print(neigh.get_params)
```

```
<bound method BaseEstimator.get_params of KNeighborsClassifier(algorithm
='brute', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=91, p=2,
                     weights='uniform')>
```

In [131]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and ga
# besed on the simple for loop method i have choosen best k is 91.

best_k_set3 = 91
```

2.3.2 Train model using the best hyper-parameter value set3

In [132]:

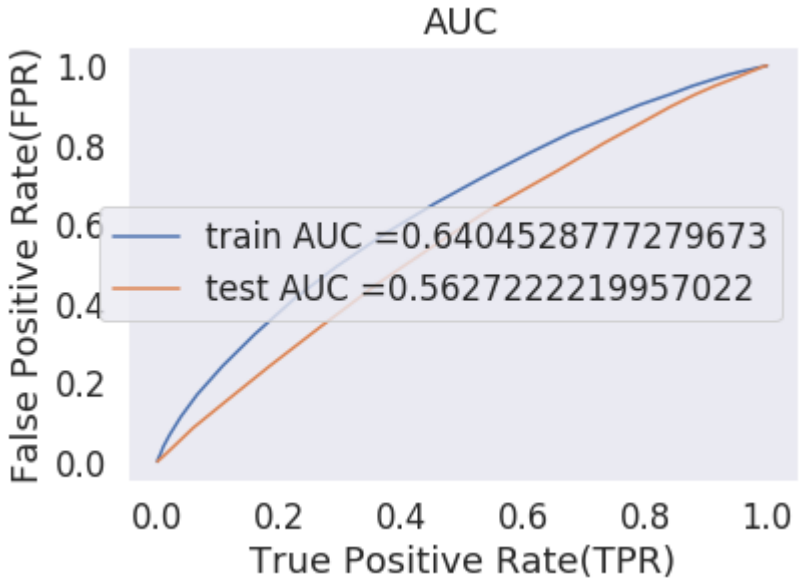
```
%%time
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklea
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier(n_neighbors=best_k_set3,algorithm='brute')
neigh.fit(set3_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(set3_train)[:
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(set3_test)[:,1])

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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```




```
CPU times: user 1h 27min 17s, sys: 1min 26s, total: 1h 28min 43s
Wall time: 1h 28min 45s
```

2.3.3 confusion_matrix : set3_train and set3_test

In [133]:

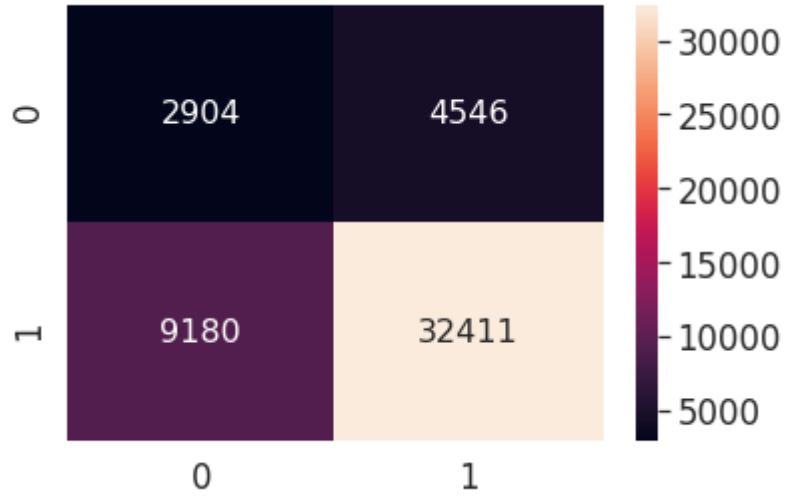
```
#Confustion Matrix Train Set3_train
conf_matr_df_train_3 = pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred,thresl
train_fpr, train_fpr)), range(2),range(2))

sns.set(font_scale=1.5)#for label size
sns.heatmap(conf_matr_df_train_3, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24878203684518715 for threshold 0.824
```

Out[133]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5713e9d780>
```



In [134]:

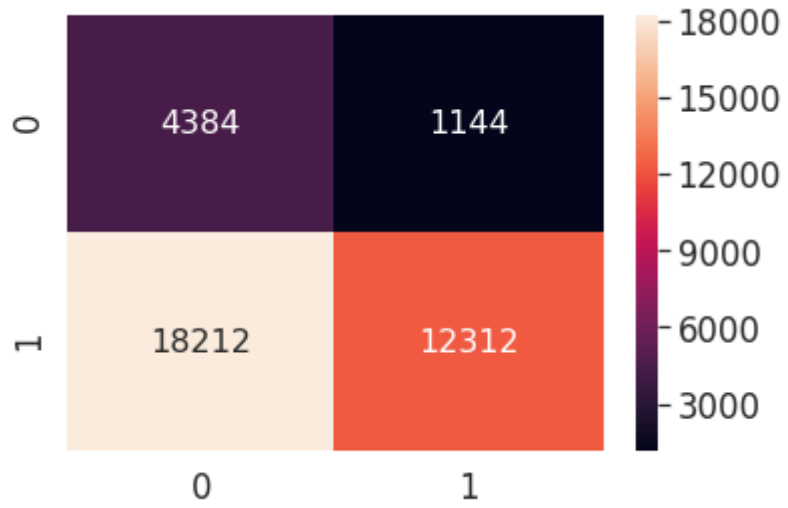
```
#Confustion Matrix Test Set3_test
neigh.fit(set3_test,y_test)
y_test_pred_3 =  neigh.predict_proba(set3_test)[:,1]
conf_matr_df_test_3 = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred_3,thresh

sns.set(font_scale=1.5)
sns.heatmap(conf_matr_df_test_3,annot=True,annot_kws={"size":16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24820817477554083 for threshold 0.879
```

Out[134]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f572b671470>
```



2.4 Applying KNN brute on TFIDF W2V, SET 4

In [135]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
# set1_ = all categorical features + numarical features + essays_bow + project_title_bo

set4_train = hstack((clean_categories_one_hot_train, clean_subcategories_one_hot_train,
                     teacher_prefix_one_hot_train,project_grade_category_one_hot_train,
                     teacher_number_of_previously_posted_projects_standardized_train,
                     preprocessed_project_title_train_tfidf_w2v_vectors,preprocessed_es
                     price_standardized_train)).tocsr()

set4_test = hstack((clean_categories_one_hot_test, clean_subcategories_one_hot_test, scl
                    teacher_prefix_one_hot_test,project_grade_category_one_hot_test,
                    teacher_number_of_previously_posted_projects_standardized_test,
                    preprocessed_project_title_test_tfidf_w2v_vectors,preprocessed_essa
                    price_standardized_test)).tocsr()

set4_cv = hstack((clean_categories_one_hot_cv, clean_subcategories_one_hot_cv, school_s
                     teacher_prefix_one_hot_cv,project_grade_category_one_hot_cv,
                     teacher_number_of_previously_posted_projects_standardized_cv,
                     preprocessed_project_title_cv_tfidf_w2v_vectors,preprocessed_essay
                     price_standardized_cv)).tocsr()
```

In [136]:

```
print("Final Data Matrix of set4 :")
print("shape of set4_train and y_train :", set4_train.shape , y_train.shape)
print("shape of set4_test and y_test   :", set4_test.shape , y_test.shape)
print("shape of set4_cv and y_cv       :", set4_cv.shape , y_cv.shape)
```

```
Final Data Matrix of set4 :
shape of set4_train and y_train : (49041, 701) (49041,)
shape of set4_test and y_test   : (36052, 701) (36052,)
shape of set4_cv and y_cv       : (24155, 701) (24155,)
```

2.4.1 Hyper parameter Tuning to find bestK: Gridsearch CV

In [137]:

```
%%time
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchC
from sklearn.model_selection import GridSearchCV

neigh = KNeighborsClassifier( algorithm='brute')
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51]}
clf = GridSearchCV(neigh, parameters, cv=3,scoring='roc_auc', return_train_score=True)
clf.fit(set4_train, y_train)

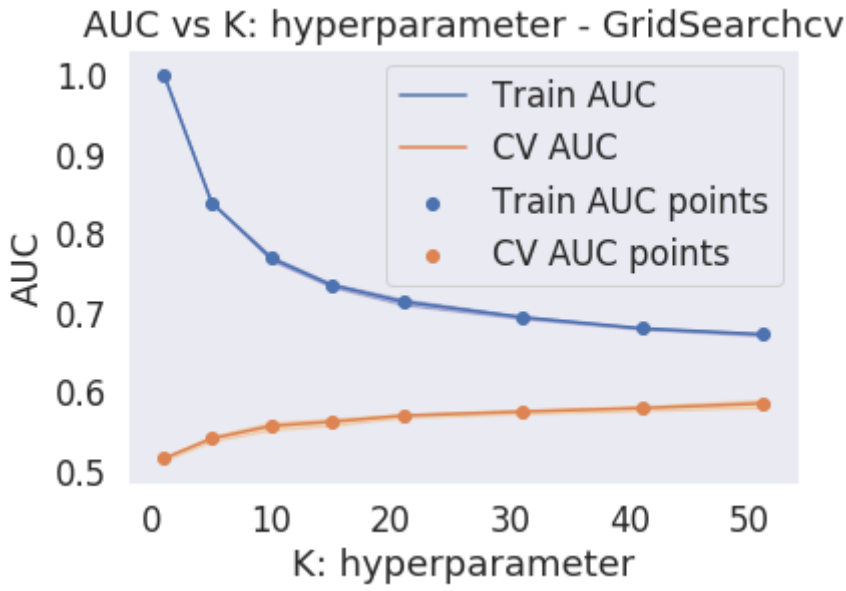
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_auc +

plt.plot(parameters['n_neighbors'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],cv_auc - cv_auc_std,cv_auc + cv_auc_st

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs K: hyperparameter - GridSearchcv")
plt.grid()
plt.show()
```




```
CPU times: user 12h 51min 1s, sys: 12min 22s, total: 13h 3min 24s
Wall time: 13h 3min 44s
```

In [179]:

```
print(neigh.get_params)
```

```
<bound method BaseEstimator.get_params of KNeighborsClassifier(algorithm
='brute', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=85, p=2,
                     weights='uniform')>
```

In [138]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and ga
# besed on the GridsearchCV method i have choosen best k is 51.

best_k_set4 = 51
```

2.4.2Train model using best hyperparameter value set4

In [147]:

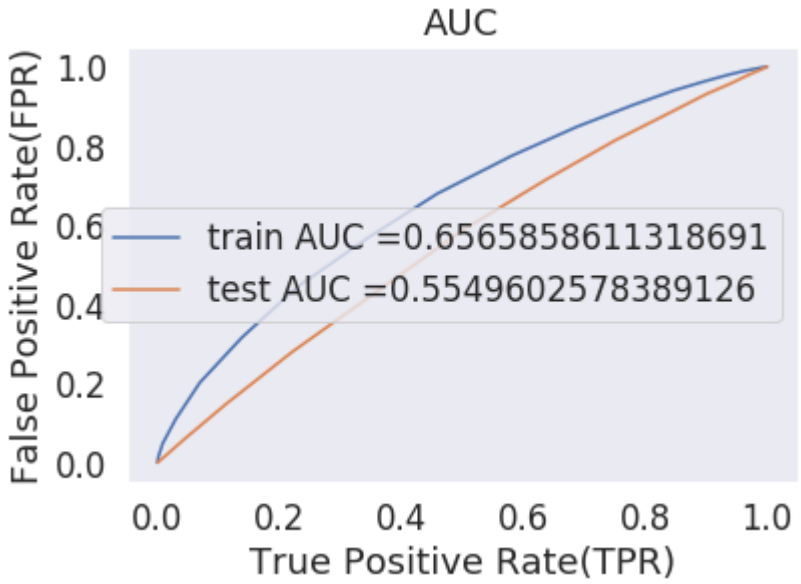
```
%%time
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklea
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier(n_neighbors=best_k_set4,algorithm='brute')
neigh.fit(set3_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of th
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(set3_train)[:
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(set3_test)[:,1])

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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```




```
CPU times: user 1h 27min 37s, sys: 1min 22s, total: 1h 29min
Wall time: 1h 29min 2s
```

Experminent changing the best k value to 85

In [182]:

```
best_k_exe = 85
```

In [183]:

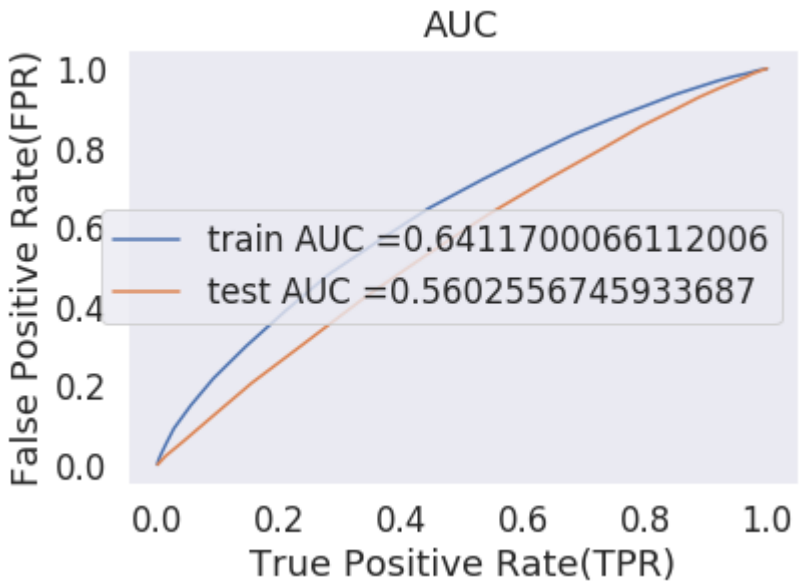
```
%%time
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklea
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier(n_neighbors=best_k_exe,algorithm='brute')
neigh.fit(set3_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of th
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(set3_train)[:
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(set3_test)[:,1])

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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```




```
CPU times: user 1h 28min 58s, sys: 1min 26s, total: 1h 30min 24s
Wall time: 1h 30min 27s
```

2.4.3 Confustion Matrix Set4_train and set4_test

In [148]:

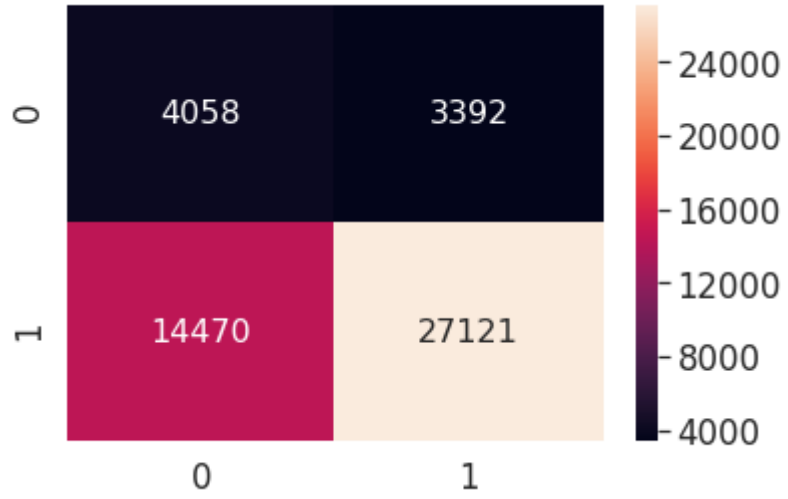
```
#Confustion Matrix Train Set4_train
conf_matr_df_train_4 = pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred,thresl
train_fpr, train_fpr)), range(2),range(2))

sns.set(font_scale=1.5)#for label size
sns.heatmap(conf_matr_df_train_4, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24832394937164995 for threshold 0.843
```

Out[148]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5723f86470>
```



In [149]:

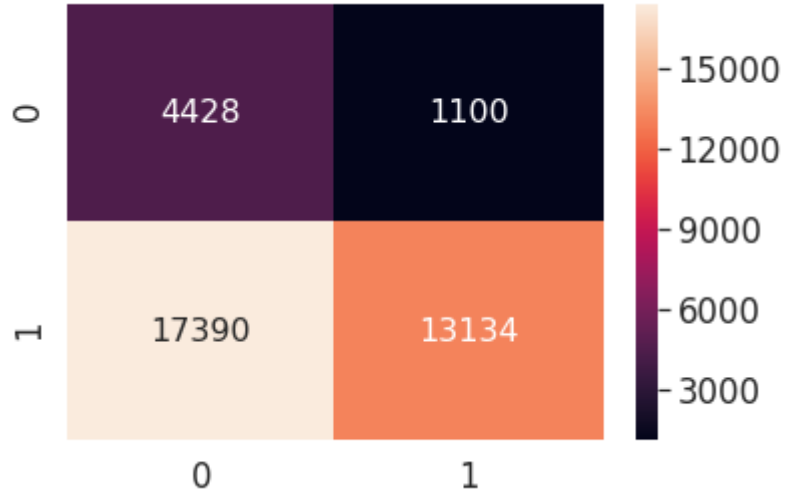
```
#Confustion Matrix Test Set4_test
neigh.fit(set4_test,y_test)
y_test_pred_4 =  neigh.predict_proba(set4_test)[:,1]
conf_matr_df_test_4 = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred_4,thresh


sns.set(font_scale=1.5)
sns.heatmap(conf_matr_df_test_4,annot=True,annot_kws={"size":16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24999947641895698 for threshold 0.882
```

Out[149]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f57229570f0>
```



2.5 Feature selection with `SelectKBest`

In [198]:

```
%%time
# Link : https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mu
# best features selection on set2
from sklearn.datasets import load_digits
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.feature_selection import SelectKBest , mutual_info_classif

print("shape of set2_train and y_train :", set2_train.shape , y_train.shape)
print("shape of set2_test and y_test   :", set2_test.shape , y_test.shape)
print("shape of set2_cv and y_cv       :", set2_cv.shape , y_cv.shape)
print("="*60)

select_bestK_top = SelectKBest(mutual_info_classif, k=2000)
set2_train_bestK_top = select_bestK_top.fit_transform(set2_train, y_train)
set2_test_bestK_top  = select_bestK_top.transform(set2_test)
set2_cv_bestK_top    = select_bestK_top.transform(set2_cv)

print("Final Data Matrix of SelectKBest")
print("shape of set2_train_bestK_top and y_train :", set2_train_bestK_top.shape , y_tra
print("shape of set2_test_bestK_top and y_test   :", set2_test_bestK_top.shape , y_test
print("shape of set2_cv_bestK_top and y_cv       :", set2_cv_bestK_top.shape , y_cv.sha
```

```
shape of set2_train and y_train : (49041, 14732) (49041,)
shape of set2_test and y_test   : (36052, 14732) (36052,)
shape of set2_cv and y_cv       : (24155, 14732) (24155,)
============================================================
Final Data Matrix of SelectKBest
shape of set2_train_bestK_top and y_train : (49041, 2000) (49041,)
shape of set2_test_bestK_top and y_test   : (36052, 2000) (36052,)
shape of set2_cv_bestK_top and y_cv       : (24155, 2000) (24155,)
CPU times: user 1min 28s, sys: 120 ms, total: 1min 28s
Wall time: 1min 28s
```

2.5.1 Hyper parameter Tuning to find bestK: Gridsearch CV - SelectKBest on set2

In [199]:

```
%%time
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchC
from sklearn.model_selection import GridSearchCV

neigh = KNeighborsClassifier( algorithm='brute')
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51, 65, 71, 85, 91]}
clf = GridSearchCV(neigh, parameters, cv=3,scoring='roc_auc', return_train_score=True)
clf.fit(set2_train_bestK_top, y_train)

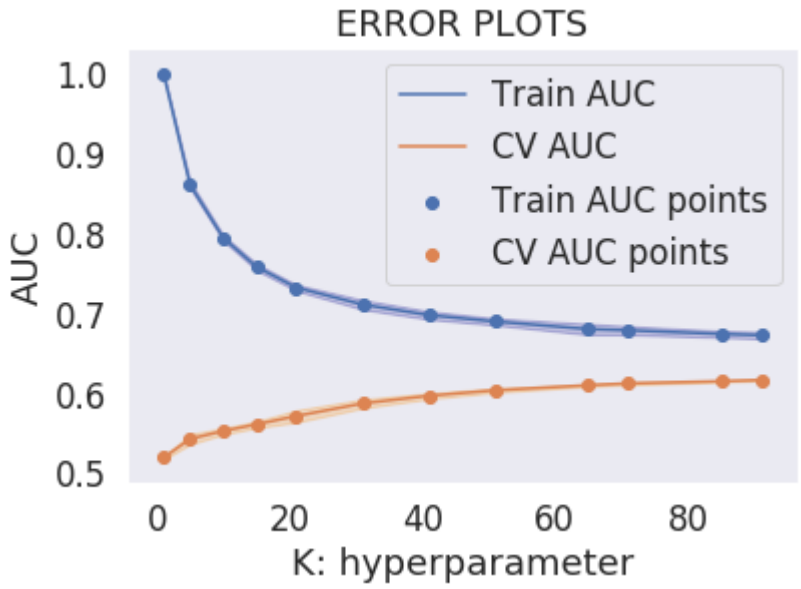
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_auc +

plt.plot(parameters['n_neighbors'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],cv_auc - cv_auc_std,cv_auc + cv_auc_st

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')

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




```
CPU times: user 2h 30min 22s, sys: 18min 18s, total: 2h 48min 41s
Wall time: 2h 48min 49s
```

In [200]:

```
print(neigh.get_params)
print("*"*100)
print(clf.get_params)
```

```
<bound method BaseEstimator.get_params of KNeighborsClassifier(algorithm
='brute', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')>
****************************************************************************
************************
<bound method BaseEstimator.get_params of GridSearchCV(cv=3, error_score
='raise-deprecating',
             estimator=KNeighborsClassifier(algorithm='brute', leaf_size=3
0,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=Non
e,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='warn', n_jobs=None,
             param_grid={'n_neighbors': [1, 5, 10, 15, 21, 31, 41, 51, 65,
71,
                                         85, 91]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)>
```

In [201]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and ga
# besed on the GridsearchCV method i have choosen best k is 85.
best_k_top = 85
```

2.5.2 Train model using the best hyper-parameter value set2_new

In [202]:

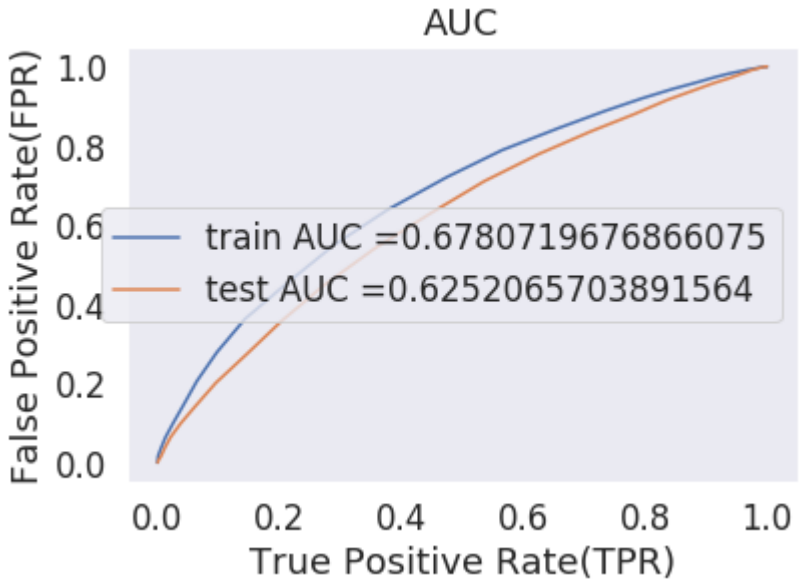
```
%%time
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklea
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier(n_neighbors=best_k_top,algorithm='brute')
neigh.fit(set2_train_bestK_top, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of th
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(set2_train_bes
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(set2_test_bestK_t

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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```




```
CPU times: user 11min 30s, sys: 1min 19s, total: 12min 50s
Wall time: 12min 50s
```

2.5.3 Confustion Matrix set2_bestK_top train & test

In [203]:

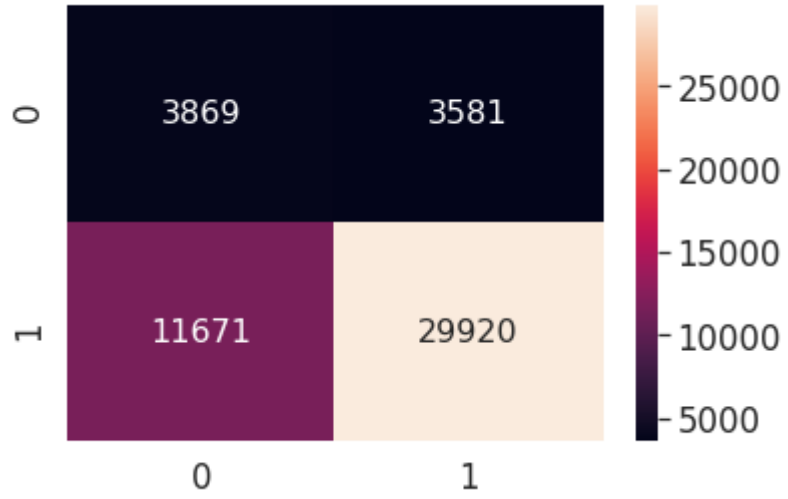
```
#Confustion Matrix Train Set2_train_bestK_top
conf_matr_df_train_bestK_top = pd.DataFrame(confusion_matrix(y_train,predict(y_train_pr
train_fpr, train_fpr)), range(2),range(2))

sns.set(font_scale=1.5)#for label size
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24934957884779965 for threshold 0.847
```

Out[203]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5723b00fd0>
```



In [204]:

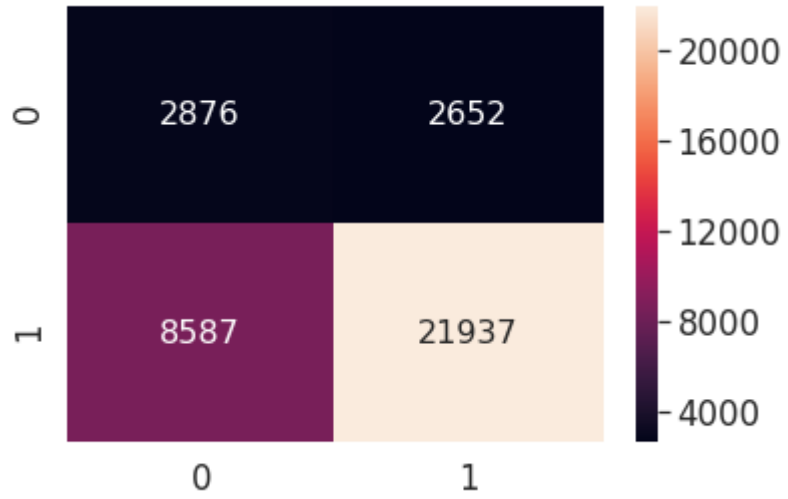
```
#Confustion Matrix Test Set2_test_bestK_top
neigh.fit(set2_test_bestK_top,y_test)
y_test_pred_bestK_top =  neigh.predict_proba(set2_test_bestK_top)[:,1]
conf_matr_df_test_bestK_top = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred_
test_fpr, test_fpr)), range(2),range(2))

sns.set(font_scale=1.5)#for label size
sns.heatmap(conf_matr_df_test_bestK_top, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24873002145635115 for threshold 0.847
```

Out[204]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5723a975f8>
```



3. Conclusions

In [1]:

```
# Link : http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
p = PrettyTable()

p.field_names = ["Vectorizer", "Model", "Hyper parameter", "AUC"]

p.add_row(["BOW","Brute",51,0.59])
p.add_row(["TFIDF","Brute", 85, 0.61])
p.add_row(["W2V", "Brute", 91, 0.56])
p.add_row(["TFIDFw2V", "Brute", 51, 0.55])
p.add_row(["TFIDF", "Top2000 on Set2", 85, 0.62])

print(p)
```

```
+------------+-----------------+-----------------+------+
| Vectorizer |      Model      | Hyper parameter | AUC  |
+------------+-----------------+-----------------+------+
|    BOW     |      Brute      |        51       | 0.59 |
|   TFIDF    |      Brute      |        85       | 0.61 |
|    W2V     |      Brute      |        91       | 0.56 |
|  TFIDFw2V  |      Brute      |        51       | 0.55 |
|   TFIDF    | Top2000 on Set2 |        85       | 0.62 |
+------------+-----------------+-----------------+------+
```

Thank You.

Sign Off RAMESH BATTU