K-Nearest Neighbor on DonorsChoose

Refrence : https://www.kaggle.com/shashank49/donors-choose-knn

About DonorsChoose Dataset

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.

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

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

Feature	Description
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*
`project_essay_4`	Fourth application essay*
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teacher. **Example:** `2`

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [1]:

```
# Importing required Packages

%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

from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

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

from tqdm import tqdm
import os

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

import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
```

1.1 Reading Data

In [148]:

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

In [149]:

```
print("Number of data points in project data : ", project_data.shape)
print("*"*75)
print("The Column names of the data : ",project_data.columns.values)
print("\n", "*"*75, "\n")
print("Number of data points in resources data : ", resource_data.shape)
print("*"*75)
print("The Column names of the data : ", resource_data.columns.values)
```

```
Number of data points in project data :  (109248, 17)
***************************************************************************
The Column names of the 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']

 ***************************************************************************

Number of data points in resources data :  (1541272, 4)
***************************************************************************
The Column names of the data :  ['id' 'description' 'quantity' 'price']
```

In [150]:

```
# 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.columns)]

#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
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[150]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	

In [151]:

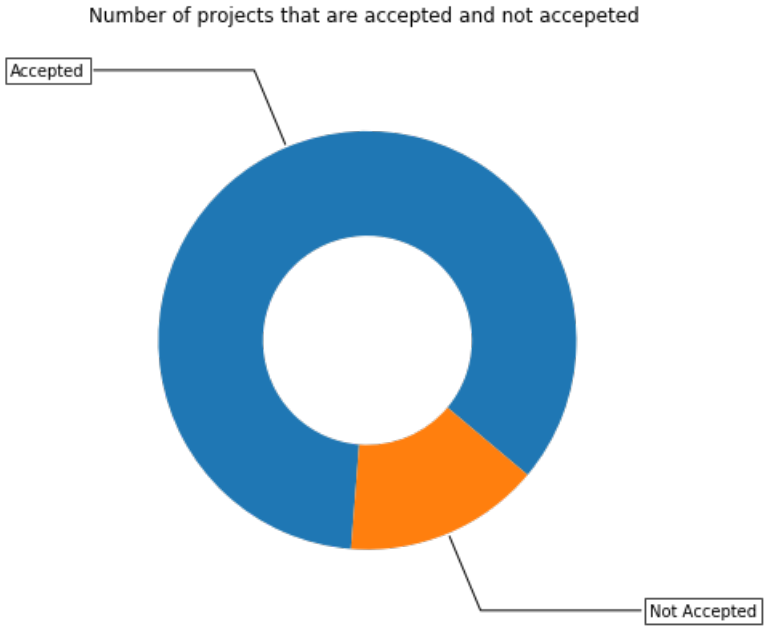
```
# Combining Project data and resource data together
price_data = resource_data.groupby("id").agg({"price" : "sum" , "quantity" : "sum"}).reset_index()
project_data = pd.merge(project_data, price_data, on = "id", how = "left")
```

Project Accepted and Not Accepted graph

In [152]:

```
y_value_counts = project_data["project_is_approved"].value_counts()
print("The number of projects that are approved for funding ", y_value_counts[1], ", (" ,
(y_value_counts[1]/(y_value_counts[1] + y_value_counts[0])) *100, "%)")
print("The Number of projects that are not approved for funding ", y_value_counts[0], ", (", (y_va
lue_counts[0]/(y_value_counts[1] + y_value_counts[0]))*100, "%)")

fig, ax = plt.subplots(figsize = (6,6), subplot_kw = dict(aspect = "equal"))
recipe = ["Accepted", "Not Accepted"]

data = [y_value_counts[1], y_value_counts[0]]

wedges, texts = ax.pie(data, wedgeprops = dict(width = 0.5), startangle = -40)

bbox_props = dict(boxstyle = "square, pad=0.3", fc = "w", ec = "k" , lw = 0.72)
kw = dict(xycoords = "data", textcoords = "data", arrowprops = dict(arrowstyle = "-"), bbox = bbox_
props, zorder = 0, va = "center")

for i, p in enumerate(wedges):
    ang = (p.theta2 - p.theta1)/2. + p.theta1
    y = np.sin(np.deg2rad(ang))
    x = np.cos(np.deg2rad(ang))
    horizontalalignment = {-1: "right", 1: "left"}[int(np.sign(x))]
    connectionstyle = "angle,angleA=0,angleB={}".format(ang)
    kw["arrowprops"].update({"connectionstyle": connectionstyle})
    ax.annotate(recipe[i], xy=(x, y), xytext=(1.35*np.sign(x), 1.4*y),
                horizontalalignment=horizontalalignment, **kw)

ax.set_title("Number of projects that are accepted and not accepeted \n\n")

plt.show()
```

```
The number of projects that are approved for funding  92706 , ( 84.85830404217927 %)
The Number of projects that are not approved for funding  16542 , ( 15.141695957820739 %)
```



1.2 (a) Text Preprocessing of Project_Subject_Categories

In [153]:

```
categories = list(project_data['project_subject_categories'].values)

# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039

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

cat_list = []
for i in categories:
    temp = ""
    # Considering for the text "Maths & Science, Warmth,Care & Hunger"

    for j in i.split(','):  # it will split in three parts ["Maths & Science", "Warmth", "Care & Hu
nger"]
        if 'The' in j.split(): # This will split each of the category based on space  "Math & Scien
ce" to "Math" ,"&" ,"Science"
            j = j.replace('The' , '') # If we have words "The" we are going to replace it with '' (
i.e. removing the)
        j = j.replace(' ','') #we placing all the ' '(space) with ''(empty) ex: "Math & Scioence" w
ith "Math&Science"
        temp += j.strip()+" " # " abc ".strip() will return "abc", removing of the trailing spaces
        temp = temp.replace('&', '_') # we are replacing the "&" into "_"
    cat_list.append(temp.strip())
```

In [154]:

```
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]))
```

In [155]:

```
sorted_cat_dict
```

Out[155]:

```
{'Warmth': 1388,
 'Care_Hunger': 1388,
 'History_Civics': 5914,
 'Music_Arts': 10293,
 'AppliedLearning': 12135,
 'SpecialNeeds': 13642,
 'Health_Sports': 14223,
 'Math_Science': 41421,
 'Literacy_Language': 52239}
```

1.2 (b) Text preprocessing : project_subject_subcategories

In [156]:

```
sub_categories = list(project_data['project_subject_subcategories'].values)

# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

sub_cat_list = []

for i in sub_categories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Maths & Science" , "Warmth" , "Care
& Hunger"]
        if 'The' in j.split(): # this will split each of the category based on space "Math & scienc
e" => "Math" , "&" , "Science"
            j = j.replace('The', '') # if we have the words "The" we are going to replace it with '
' (i.e. removing of the)
        j = j.replace(' ', '') # we are placing all the space ' ' with empty space '' eg. "math & s
cience" to math&science
        temp += j.strip() + ' ' # " abc ".strip() to "abc", remove all the trailing spaces
        temp = temp.replace("&", "_")
    sub_cat_list.append(temp.strip())


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

#count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039

my_counter = Counter()
for word in project_data["clean_subcategories"].values:
    my_counter.update(word.split())

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

In [157]:

```
len(sorted_sub_cat_dict)
sorted_sub_cat_dict
```

Out[157]:

```
{'Economics': 269,
 'CommunityService': 441,
 'FinancialLiteracy': 568,
 'ParentInvolvement': 677,
 'Extracurricular': 810,
 'Civics_Government': 815,
 'ForeignLanguages': 890,
 'NutritionEducation': 1355,
 'Warmth': 1388,
 'Care_Hunger': 1388,
 'SocialSciences': 1920,
```

```
 'PerformingArts': 1961,
 'CharacterEducation': 2065,
 'TeamSports': 2192,
 'Other': 2372,
 'College_CareerPrep': 2568,
 'Music': 3145,
 'History_Geography': 3171,
 'Health_LifeScience': 4235,
 'EarlyDevelopment': 4254,
 'ESL': 4367,
 'Gym_Fitness': 4509,
 'EnvironmentalScience': 5591,
 'VisualArts': 6278,
 'Health_Wellness': 10234,
 'AppliedSciences': 10816,
 'SpecialNeeds': 13642,
 'Literature_Writing': 22179,
 'Mathematics': 28074,
 'Literacy': 33700}
```

1.2 (b) Text preprocessing : project_subject_subcategories

In [158]:

```
project_grade_categories = list(project_data['project_grade_category'].values)

# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

grade_cat_list = []

for i in project_grade_categories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Maths & Science" , "Warmth" , "Care
& Hunger"]
        if 'The' in j.split(): # this will split each of the category based on space "Math & scienc
e" => "Math" , "&" , "Science"
            j = j.replace('The', '') # if we have the words "The" we are going to replace it with '
' (i.e. removing of the)
        j = j.replace(' ', '_') # we are placing all the space ' ' with empty space '' eg. "math &
science" to math&science
        temp += j.strip() + ' ' # " abc ".strip() to "abc", remove all the trailing spaces
        temp = temp.replace("-", "_")
    grade_cat_list.append(temp.strip())

project_data.drop(["project_grade_category"], axis = 1, inplace = True)
project_data['project_grade_category'] = grade_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["project_grade_category"].values:
    my_counter.update(word.split())

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

In [159]:

```
print(len(sorted_grade_cat_dict))
print(sorted_grade_cat_dict)
```

```
4
{'Grades_9_12': 10963, 'Grades_6_8': 16923, 'Grades_3_5': 37137, 'Grades_PreK_2': 44225}
```

1.3 Text Preprocessing

In [160]:

```
## Project Essay
## 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 [161]:

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

Out[161]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_title	project_essay_1	project
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Engineering STEAM into the Primary Classroom	I have been fortunate enough to use the Fairy ...	M co back
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Sensory Tools for Focus	Imagine being 8-9 years old. You're in your th...	I stud autis

In [162]:

```
# 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 [163]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
```

```
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "d
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

1.3 Text preprocessing : Project Essays

In [164]:

```
# Combining all the stundents
from tqdm import tqdm # tqdm is for printing status bar for knowing time to execute
preprocessed_essays = []

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

```
100%|██████████████████████████████████| 109248/109248 [02:11<00:00, 830.75it/s]
```

In [165]:

```
print(preprocessed_essays[2000])
print("\n","*"*100)
print(preprocessed_essays[1000])
print("\n","*"*100)
```

```
creativity intelligence fun albert einstein elementary library greenville elementary anything quie
t hushed space place collaboration research place incorporating technology place innovation place c
reating school serves 350 third fourth graders primarily live rural poverty stricken areas
community title school approximately 85 receive free reduced lunch inquisitive creative eager
learn love visiting library check books hear stories create digital stories use computer lab learn
ing fun want build library makerspace activities revolving around art literacy provide engaging ha
nds activities want begin makerspace fridays school recently received 1000 grant books arts
integrated makerspace received titles origami everyone make stuff ducktape cool engineering activi
ties girls need supplies correlate new informational texts adding art craft supplies students able
design create masterpieces related coursework example studying native americans students use looms
yarn recreate navajo pueblo weaving weaving also integrated literacy greek mythology story arachne
creating art perler beads many possibilities students design animals studying characteristics use
symmetry patterning create one kind originals origami reinforces geometry thinking skills
fractions problem solving fun science students need able apply read learn read book apply reading
hands art activity actually create product crucial skill real world creating designing
masterpieces using many critical thinking skills students become analytical thinkers


**********

life moves pretty fast not stop look around awhile could miss movie ferris bueller day think back
remember grandparents amazing would able flip book see day lives second graders voracious readers
love read fiction nonfiction books favorite characters include pete cat fly guy piggie elephant me
rcy watson also love read insects space plants students hungry bookworms students eager learn read
world around kids love school like little sponges absorbing everything around parents work long ho
urs usually not see children students usually cared grandparents family friend students not someon
e speaks english home thus difficult students acquire language think forward would not mean lot ki
ds nieces nephews grandchildren able see day life today 30 years memories precious us able share m
emories future generations rewarding experience part social studies curriculum students learning c
hanges time students studying photos learn community changed time particular look photos study lan
d buildings clothing schools changed time culminating activity students capture slice history pres
erve scrap booking key important events young lives documented date location names students using
```

```
photos home school create second grade memories scrap books preserve unique stories future
generations enjoy donation project provide second graders opportunity learn social studies fun cre
ative manner scrapbooks children share story others historical document rest lives


**********************************************************************************************************
```

1.4 Text Preprocessing : Project titles

In [166]:

```
preprocessed_titles = []

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

```
100%|████████████████████████████████| 109248/109248 [00:09<00:00, 11165.36it/s]
```

In [167]:

```
print(preprocessed_titles[200])
print("\n","*"*50)
print(preprocessed_titles[1000])
print("\n","*"*50)
print(preprocessed_titles[1500])
print("\n","*"*50)
print(preprocessed_titles[2000])
print("\n","*"*50)
```

```
leveled reading classroom library

 **************************************************
empowering students art learning

 **************************************************
technology 8th grade reading class

 **************************************************
empowering students art makerspace

 **************************************************
```

1.4.1 Text Preprocessing : project resource summary

In [168]:

```
preprocessed_resource_summary = []

for sentence in tqdm(project_data['project_resource_summary'].values):
    sent = decontracted(sentence)
    sent = sent.replace('\\r',' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed_resource_summary.append(sent.lower().strip())
```

```
100%|████████████████████████████████| 109248/109248 [00:13<00:00, 8388.40it/s]
```

In [169]:

```
print(preprocessed_resource_summary[200])
print("\n","*"*50)
print(preprocessed_resource_summary[1000])
print("\n","*"*50)
print(preprocessed_resource_summary[1500])
print("\n","*"*50)
print(preprocessed_resource_summary[2000])
print("\n","*"*50)
```

```
students need leveled reading library class support individual reading needs

 **************************************************
students need scrapbooks self adhesive glitter tape photo corners frames hole punchers preserve me
mories second grade

 **************************************************
students need access able view multimedia content group discussion desktop would allow us desktop
designated

 **************************************************
students need arts craft supplies looms needles yarn perler beads origami paper sharpies library s
team makerspace

 **************************************************
```

In [170]:

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

Out[170]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_title	project_essay_1	projec
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Engineering STEAM into the Primary Classroom	I have been fortunate enough to use the Fairy ...	M co back
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Sensory Tools for Focus	Imagine being 8-9 years old. You're in your th...	N stuc autisı

In [171]:

```
## Reframing the column names

project_data['preprocessed_essays'] = preprocessed_essays
project_data['preprocessed_titles'] = preprocessed_titles
project_data["preprocessed_resource_summary"] = preprocessed_resource_summary

project_data.drop(['Unnamed: 0'], axis = 1, inplace = True)
project_data.drop(['id'], axis = 1, inplace = True)
project_data.drop(['teacher_id'], axis = 1, inplace = True)
project_data.drop(['project_title'], axis = 1, inplace = True)
project_data.drop(['essay'], axis = 1, inplace = True)
project_data.drop(['project_resource_summary'], axis = 1, inplace = True)


project_data.drop(['project_essay_1'], axis = 1, inplace = True)
project_data.drop(['project_essay_2'], axis = 1, inplace = True)
project_data.drop(['project_essay_3'], axis = 1, inplace = True)
project_data.drop(['project_essay_4'], axis = 1, inplace = True)
```

In [172]:

In [172]:

```
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('null')
project_data['school_state'] = project_data["school_state"].fillna("null")
project_data["Date"] = project_data["Date"].fillna("null")
project_data["project_grade_category"] = project_data["project_grade_category"].fillna("null")
project_data["teacher_number_of_previously_posted_projects"] =
project_data["teacher_number_of_previously_posted_projects"].fillna("null")


project_data['project_is_approved'] = project_data['project_is_approved'].fillna('null')
project_data['price'] = project_data['price'].fillna('null')
project_data['quantity'] = project_data['quantity'].fillna('null')
project_data['clean_categories'] = project_data['clean_categories'].fillna('null')

project_data['clean_subcategories'] = project_data['clean_subcategories'].fillna('null')
project_data['preprocessed_essays'] = project_data['preprocessed_essays'].fillna('null')
project_data['preprocessed_titles'] = project_data["preprocessed_titles"].fillna("null")
project_data["preprocessed_resource_summary"] =
project_data["preprocessed_resource_summary"].fillna("null")
```

In [173]:

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

Out[173]:

	teacher_prefix	school_state	Date	teacher_number_of_previously_posted_projects	project_is_approved	price	quantity	clean_c
0	Mrs.	CA	2016-04-27 00:27:36	53	1	725.05	4	Math
1	Ms.	UT	2016-04-27 00:31:25	4	1	213.03	8	Spe

In [174]:

```
project_data.to_csv(r'project_data1.csv', index = False)
```

Assignment 3: Apply KNN

1. **[Task-1] Apply KNN(brute force version) on these feature sets**
   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)
   - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)
   - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)
2. **Hyper paramter tuning to find best K**
   - Find the best hyper parameter which results in the maximum AUC 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 with predicted and original labels of test data points
4. **[Task-2]**
   - Select top 2000 features from feature Set 2 using `SelectKBest` and then apply KNN on top of these features
     - `from sklearn.datasets import load_digits`
     - `from sklearn.feature_selection import SelectKBest, chi2`

```
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

---

2. K Nearest Neighbor

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

In [175]:

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

In [45]:

```
## For BOW and TFIDF nrows = 50000, for Avg W2v and TFIDF W2v nrows = 10,000

project_data1 = pd.read_csv("project_data1.csv", nrows = 10000)

project_data1.head(2)
```

Out[45]:

	teacher_prefix	school_state	Date	teacher_number_of_previously_posted_projects	project_is_approved	price	quantity	clean_c
0	Mrs.	CA	2016-04-27 00:27:36	53	1	725.05	4	Matl
1	Ms.	UT	2016-04-27 00:31:25	4	1	213.03	8	Spe

In [46]:

```
project_data1.head(2)

project_data1['teacher_prefix'] = project_data1['teacher_prefix'].fillna('null')
project_data1['school_state'] = project_data1["school_state"].fillna("null")
project_data1["Date"] = project_data1["Date"].fillna("null")
project_data1["project_grade_category"] = project_data1["project_grade_category"].fillna("null")
project_data1["teacher_number_of_previously_posted_projects"] =
project_data1["teacher_number_of_previously_posted_projects"].fillna("null")
```

```
project_data1['project_is_approved'] = project_data1['project_is_approved'].fillna('null')
project_data1['price'] = project_data1['price'].fillna('null')
project_data1['quantity'] = project_data1['quantity'].fillna('null')
project_data1['clean_categories'] = project_data1['clean_categories'].fillna('null')

project_data1['clean_subcategories'] = project_data1['clean_subcategories'].fillna('null')
project_data1['preprocessed_essays'] = project_data1['preprocessed_essays'].fillna('null')
project_data1['preprocessed_titles'] = project_data1["preprocessed_titles"].fillna("null")
project_data1["preprocessed_resource_summary"] =
project_data1["preprocessed_resource_summary"].fillna("null")
```

In [47]:

```
y = project_data1["project_is_approved"].values
X = project_data1.drop(["project_is_approved"], axis = 1)
X.head(3)
```

Out[47]:

	teacher_prefix	school_state	Date	teacher_number_of_previously_posted_projects	price	quantity	clean_categories	clean_sub
0	Mrs.	CA	2016-04-27 00:27:36	53	725.05	4	Math_Science	Applie Health_
1	Ms.	UT	2016-04-27 00:31:25	4	213.03	8	SpecialNeeds	Sp
2	Mrs.	CA	2016-04-27 00:46:53	10	329.00	1	Literacy_Language	

In [48]:

```
# Train test split

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, stratify = y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size = 0.33, stratify = y_tr
ain)
```

In [49]:

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

```
(4489, 12) (4489,)
(2211, 12) (2211,)
(3300, 12) (3300,)
```

In [50]:

```
print(X.shape)
print(y.shape)
```

```
(10000, 12)
(10000,)
```

In [51]:

```
y_value_counts = project_data1["project_is_approved"].value_counts()
print("The number of projects that are approved for funding ", y_value_counts[1], ", (" ,
(y_value_counts[1]/(y_value_counts[1] + y_value_counts[0])) *100, "%)")
print("The Number of projects that are not approved for funding ", y_value_counts[0], ", (", (y_va
lue_counts[0]/(y_value_counts[1] + y_value_counts[0]))*100, "%)")
```

```
The number of projects that are approved for funding  8620 , ( 86.2 %)
The Number of projects that are not approved for funding  1380 , ( 13.8 %)
```

OverSampling the data due to imbalanced dataset

In [52]:

```
from sklearn.datasets import make_classification
from imblearn.over_sampling import RandomOverSampler
from collections import Counter

ros = RandomOverSampler(sampling_strategy = 'minority', random_state = 42)
X_train, y_train = ros.fit_resample(X_train, y_train)
print("After Resample the dataset, the 1s and 0s are ", Counter(y_train))

X_train = pd.DataFrame(X_train, columns = X.columns)
X_train.head(2)
```

```
After Resample the dataset, the 1s and 0s are  Counter({0: 3869, 1: 3869})
```

Out[52]:

	teacher_prefix	school_state	Date	teacher_number_of_previously_posted_projects	price	quantity	clean_categories	clean_subc
0	Mrs.	NY	2016-05-10 11:37:03	7	56.64	48	Literacy_Language	
1	Mrs.	TX	2016-05-24 00:04:32	16	170.7	25	Literacy_Language	

No of Datapoints for each model :

```
- BOW (SET1)        : 50,000k
- TFIDF (SET2)      : 50,000k
- AVG W2V (SET3)    : 10,000k
- TFIDF W2V (SET4)  : 10,000k
```

2.2 BOW: Make Data Model Ready: encoding Text, numerical, categorical features on 50k datapoints

we are going to consider

```
2.2.1 Text:
 - project_title : text data
 - project_essay : text data
 - project_resource_summary: text data (optinal)

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

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

2.2.1 Vectorizing Text

(i) Vectorizing project Essay

In [32]:

```
# Project Essay

vectorizer = CountVectorizer(min_df = 10, ngram_range = (1,4), max_features = 2000)
vectorizer.fit(X_train["preprocessed_essays"].values)  # Fit has to happen only on train data

# We use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train["preprocessed_essays"].values)
X_cv_essay_bow = vectorizer.transform(X_cv["preprocessed_essays"].values)
X_test_essay_bow = vectorizer.transform(X_test["preprocessed_essays"].values)

print("After vectorization ")
print(X_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
print("\n", "*"*90)
```

```
After vectorization
(37700, 2000) (37700,)
(11055, 2000) (11055,)
(16500, 2000) (16500,)

 ******************************************************************************************
```

In [33]:

```
# print("YOU SHOULD NOT DO SOMETHING LIKE THIS")
# vectorizer = CountVectorizer()
# x_train_bow = vectorizer.fit_transform(X_train['essay'].values)
# x_cv_bow = vectorizer.fit_transform(X_cv['essay'].values)
# x_test_bow = vectorizer.fit_transform(X_test['essay'].values)

# print(x_train_bow.shape, y_train.shape)
# print(x_cv_bow.shape, y_cv.shape)
# print(x_test_bow.shape, y_test.shape)

print("NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME")
```

```
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

YOU SHOULD NOT DO LIKE THIS

1. THE VOCABULARY SHOULD BUILT ONLY WITH THE WORDS OF TRAIN DATA

```
vectorizer = CountVectorizer()
x_train_bow = vectorizer.fit_transform(X_train)
x_cv_bow = vectorizer.fit_transform(X_cv)
x_test_bow = vectorizer.fit_transform(X_test)
```

2. DATA LEAKAGE PROBLEM: IF WE DO LIKE THIS WE ARE LOOKING AT THE TEST DATA BEFORE MODELING

```
vectorizer = CountVectorizer()
X_bow = vectorizer.fit_transfomr(X)
X_train, X_test, y_train, y_test = train_test_split(X_bow, Y, test_size=0.33)
```

3. YOU SHOULD PASS THE PROBABILITY SCORES NOT THE PREDICTED VALUES

```
y_pred =  neigh.predict(X)
roc_auc_score(y_ture,y_pred)
```

WHAT ARE THESE FUNCTIONS: FIT, TRANSFORM, FIT_TRANSFORM

When you do
vec = CountVectorizer()

then it will initiate the *CountVectorizer* with default parameters.

vec.fit(Train_text): Means that internally it is only learning the vocabulary of `Text` i.e. unique n-grams

bag_of_words = vec.transform(Train_text): Means it is applying that learned parameters (vocabulary) to the data and thus giving you output i.e. words in Bag of words formate.

Now, as you should know that the *vocabulary(unique n-grams)* can be different for *Train Text* and *Test Text* thus they will give you different dimensional matrices for Train and Test.

So what you should do is - *vec = CountVectorizer()*, *vec.fit(Train_text)*

which learns the vocabulary of *Train Text* and then apply or transform your both *Train Text* and *Test Text* using that
learned vocabulary to ensure the same dimensions for both of them by doing -

bag_of_words_train = vec.transform(Train_text)
and
bag_of_words_test = vec.transform(Test_text)

so to conclude

```
model = CountVectorizer()
model.fit(train_text)
train_bow = model.transform(train_text)
test_bow = model.transform(test_text)
```

or

```
model = CountVectorizer()
train_bow = model.fit_transform(train_text)
test_bow = model.transform(test_text)
```

(ii) Vectorizing Project titles

In [34]:

```
## Project Titles

vectorizer = CountVectorizer(min_df = 10, ngram_range = (1,4), max_features = 2000)
vectorizer.fit(X_train["preprocessed_titles"].values) # fitting is done on train data alone

# using fitted countvectorizer to convert text to vector
X_train_title_bow = vectorizer.transform(X_train["preprocessed_titles"].values)
X_cv_title_bow = vectorizer.transform(X_cv["preprocessed_titles"].values)
X_test_title_bow = vectorizer.transform(X_test["preprocessed_titles"].values)

print("After Vectorization of title")
print(X_train_title_bow.shape, y_train.shape)
print(X_cv_title_bow.shape, y_cv.shape)
print(X_test_title_bow.shape, y_test.shape)
```

```
After Vectorization of title
(37700, 2000) (37700,)
(11055, 2000) (11055,)
(16500, 2000) (16500,)
```

(iii) Vectorizing Resource Summary

In [35]:

```
## Resource Summary

vectorizer = CountVectorizer(min_df = 10, ngram_range = (1,4), max_features= 2000)
vectorizer.fit(X_train["preprocessed_resource_summary"].values) # Fitting is done on train data
only

# Using fitted CountVectorizer to convert text to vector
X_train_summary_bow = vectorizer.transform(X_train["preprocessed_resource_summary"].values)
X_cv_summary_bow = vectorizer.transform(X_cv["preprocessed_resource_summary"].values)
X_test_summary_bow = vectorizer.transform(X_test["preprocessed_resource_summary"].values)

print("After Vectorizing of Project Resource Summary ")
print(X_train_summary_bow.shape, y_train.shape)
print(X_cv_summary_bow.shape, y_cv.shape)
print(X_test_summary_bow.shape, y_test.shape)
```

```
After Vectorizing of Project Resource Summary
(37700, 2000) (37700,)
(11055, 2000) (11055,)
(16500, 2000) (16500,)
```

2.2.2 Vcetorizing Cateforical data

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

(i). Vectorizing School_state

In [53]:

```
## School State

vectorizer = CountVectorizer()
vectorizer.fit(X_train["school_state"].values) ## Fitting is done to train data alone

# We use the CountVectorizer to convert text to vector
X_train_state_one = vectorizer.transform(X_train["school_state"].values)
X_cv_state_one = vectorizer.transform(X_cv["school_state"].values)
X_test_state_one = vectorizer.transform(X_test["school_state"].values)

print("After vectorizations of School state")
print(X_train_state_one.shape, y_train.shape)
print(X_cv_state_one.shape, y_cv.shape)
print(X_test_state_one.shape, y_test.shape)
print("\n",vectorizer.get_feature_names())
```

```
After vectorizations of School state
(7738, 51) (7738,)
(2211, 51) (2211,)
(3300, 51) (3300,)

 ['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', '
ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm'
, 'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', '
wv', 'wy']
```

(ii). Vectorizing Clean_Categories

In [54]:

```
# Clean Categories

vectorizer = CountVectorizer()
vectorizer.fit(X_train["clean_categories"].values) # Fitting is done for train data alone
#We use the CountVectorizer to convert text to vector
X_train_categories_one = vectorizer.transform(X_train["clean_categories"].values)
X_cv_categories_one  = vectorizer.transform(X_cv["clean_categories"].values)
X_test_categories_one = vectorizer.transform(X_test["clean_categories"].values)

print("After vectorizations of categories")
print(X_train_categories_one.shape, y_train.shape)
print(X_cv_categories_one.shape, y_cv.shape)
print(X_test_categories_one.shape, y_test.shape)
print(vectorizer.get_feature_names())
```

```
After vectorizations of categories
(7738, 7) (7738,)
(2211, 7) (2211,)
(3300, 7) (3300,)
['appliedlearning', 'health_sports', 'history_civics', 'literacy_language', 'math_science',
'music_arts', 'specialneeds']
```

(iii). Vectorizing Clean_SubCategories

In [55]:

```
## Clean SubCategoiries

vectorizer = CountVectorizer()
vectorizer.fit(X_train["clean_subcategories"].values) # Fitting done for train data only

# WE use the CountVectorizer to convert text to vector
X_train_subcategories_one = vectorizer.transform(X_train["clean_subcategories"].values)
X_cv_subcategories_one = vectorizer.transform(X_cv["clean_subcategories"].values)
X_test_subcategories_one = vectorizer.transform(X_test["clean_subcategories"].values)

print("After vectorizations of subcategories")
print(X_train_subcategories_one.shape, y_train.shape)
print(X_cv_subcategories_one.shape, y_cv.shape)
print(X_test_subcategories_one.shape, y_test.shape)
print(vectorizer.get_feature_names())
```

```
After vectorizations of subcategories
(7738, 28) (7738,)
(2211, 28) (2211,)
(3300, 28) (3300,)
['appliedsciences', 'charactereducation', 'civics_government', 'college_careerprep',
'communityservice', 'earlydevelopment', 'economics', 'environmentalscience', 'esl',
'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness', 'health_lifescience', '
health_wellness', 'history_geography', 'literacy', 'literature_writing', 'mathematics', 'music', '
nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socialsciences',
'specialneeds', 'teamsports', 'visualarts']
```

(iv). Vectorizing project_grade_Category

In [56]:

```
#This step is to intialize a vectorizer with vocab from train data

from collections import Counter
my_counter4 = Counter()
for word in X_train['project_grade_category'].values:
    my_counter4.update(word.split(','))


# dict sort by value python: https://stackoverflow.com/a/613218/4084039
project_grade_category_dict = dict(my_counter4)
sorted_project_grade_category_dict = dict(sorted(project_grade_category_dict.items(), key=lambda
kv: kv[1]))
```

```
# Project grade category

vectorizer = CountVectorizer(vocabulary=list(sorted_project_grade_category_dict.keys()), lowercase
=False, binary=True)

vectorizer.fit(X_train["project_grade_category"].values) # Fit only train data

#We use the CountVcetorizer to convert text to vector
X_train_grade_one = vectorizer.transform(X_train["project_grade_category"].values)
X_cv_grade_one = vectorizer.transform(X_cv["project_grade_category"].values)
X_test_grade_one = vectorizer.transform(X_test["project_grade_category"].values)

print("After vectorizations of prooject grade category")
print(X_train_grade_one.shape, y_train.shape)
print(X_cv_grade_one.shape, y_cv.shape)
print(X_test_grade_one.shape, y_test.shape)
print(vectorizer.get_feature_names())

# print(X_train_grade_one)
print(X_train_grade_one.toarray())
```

```
After vectorizations of prooject grade category
(7738, 4) (7738,)
(2211, 4) (2211,)
(3300, 4) (3300,)
['Grades_9_12', 'Grades_6_8', 'Grades_3_5', 'Grades_PreK_2']
[[0 0 0 1]
 [0 0 0 1]
 [0 0 0 1]
 ...
 [1 0 0 0]
 [0 1 0 0]
 [0 0 0 1]]
```

(v). Vectrizing teacher prefix

In [57]:

```
# teacher prefix
vectorizer = CountVectorizer()
vectorizer.fit(X_train["teacher_prefix"].values) # fitting has to happen only on train set

# we use countvectorizer to convert text to vector
X_train_prefix_one = vectorizer.transform(X_train["teacher_prefix"].values)
X_cv_prefix_one = vectorizer.transform(X_cv["teacher_prefix"].values)
X_test_prefix_one = vectorizer.transform(X_test["teacher_prefix"].values)

print("After vectorizations of teacher prefix")
print(X_train_prefix_one.shape, y_train.shape)
print(X_cv_prefix_one.shape, y_cv.shape)
print(X_test_prefix_one.shape, y_test.shape)
print(vectorizer.get_feature_names())

print(X_train_prefix_one.toarray())
```

```
After vectorizations of teacher prefix
(7738, 6) (7738,)
(2211, 6) (2211,)
(3300, 6) (3300,)
['dr', 'mr', 'mrs', 'ms', 'null', 'teacher']
[[0 0 1 0 0 0]
 [0 0 1 0 0 0]
 [0 0 1 0 0 0]
 ...
 [0 0 0 1 0 0]
 [0 0 1 0 0 0]
 [0 0 1 0 0 0]]
```

2.2.3. Vectorizing Numerical Data

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

(i) Vectorizing quantity

In [58]:

```
# for quantity

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


normalizer.fit(X_train["quantity"].values.reshape(-1, 1))

X_train_quantity_norm = normalizer.transform(X_train["quantity"].values.reshape(-1,1))
X_cv_quantity_norm = normalizer.transform(X_cv["quantity"].values.reshape(-1,1))
X_test_quantity_norm = normalizer.transform(X_test["quantity"].values.reshape(-1,1))

print("After vectorizations Quantity")
print(X_train_quantity_norm.shape, y_train.shape)
print(X_cv_quantity_norm.shape, y_cv.shape)
print(X_test_quantity_norm.shape, y_test.shape)
```

```
After vectorizations Quantity
(7738, 1) (7738,)
(2211, 1) (2211,)
(3300, 1) (3300,)
```

(ii) Vectorizing teacher previously posted projects

In [59]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

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

normalizer.fit(X_train["teacher_number_of_previously_posted_projects"].values.reshape(-1,1))

X_train_teacher_previously_norm =
normalizer.transform(X_train["teacher_number_of_previously_posted_projects"].values.reshape(-1,1))
X_cv_teacher_previously_norm =
normalizer.transform(X_cv["teacher_number_of_previously_posted_projects"].values.reshape(-1,1))
X_test_teacher_previously_norm =
normalizer.transform(X_test["teacher_number_of_previously_posted_projects"].values.reshape(-1,1))



print("After vectorizations price")
print(X_train_teacher_previously_norm.shape, y_train.shape)
print(X_cv_teacher_previously_norm.shape, y_cv.shape)
print(X_test_teacher_previously_norm.shape, y_test.shape)
```

```
After vectorizations price
(7738, 1) (7738,)
(2211, 1) (2211,)
(3300, 1) (3300,)
```

(iii) Vectorizing price

In [60]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

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

normalizer.fit(X_train["price"].values.reshape(-1, 1))

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

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

```
After vectorizations price
(7738, 1) (7738,)
(2211, 1) (2211,)
(3300, 1) (3300,)
```

2.3 Concatenating all the features

In [44]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039

from scipy.sparse import hstack

X_tr = hstack((X_train_essay_bow, X_train_title_bow, X_train_summary_bow, \
               X_train_categories_one, X_train_subcategories_one, X_train_grade_one,
X_train_prefix_one, X_train_state_one, \
              X_train_price_norm, X_train_quantity_norm, X_train_teacher_previously_norm)).tocsr()

X_cr = hstack((X_cv_essay_bow, X_cv_title_bow, X_cv_summary_bow, \
              X_cv_categories_one, X_cv_subcategories_one, X_cv_grade_one, X_cv_prefix_one, X_cv_st
ate_one, \
              X_cv_price_norm, X_cv_quantity_norm, X_cv_teacher_previously_norm)).tocsr()

X_te = hstack((X_test_essay_bow, X_test_title_bow, X_test_summary_bow, \
              X_test_categories_one, X_test_subcategories_one, X_test_grade_one, X_test_prefix_one,
X_test_state_one, \
              X_test_price_norm, X_test_quantity_norm, X_test_teacher_previously_norm)).tocsr()

print("Final Data Matrix : ")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
```

```
Final Data Matrix :
(37700, 6099) (37700,)
(11055, 6099) (11055,)
(16500, 6099) (16500,)
```

In [ ]:

3. Applying KNN

3.1 Applying KNN brute force on BOW, SET 1 (50k datapoints)

In [197]:

```
### 3.1.1 Hyperparameter tuning, (you can follow any one of these)

## 3.1.1.1 Method 1 : SImple for loop (For problems based on memory issues)
```

In [198]:

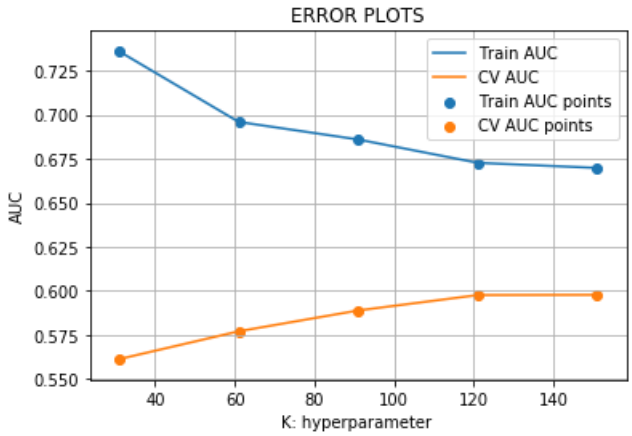
```
def batch_predict(clf, data):
    #roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posit
ive class
    # not the predicted outputs

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

    # We will be predicting for the last data points
    if data.shape[0]%1000 != 0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop : ])[:, 1])

    return y_data_pred
```

In [199]:

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

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []

#k = [51, 81, 111, 141, 171]
k = [31, 61, 91, 121, 151]
for i in tqdm(k):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs = -1)
    neigh.fit(X_tr, y_train)

    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs

    train_auc.append(roc_auc_score(y_train, y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
    """
    #******************************
   # predict the classlabel / response on the crossvalidation train
   # pred = knn.predict(X_cv)
```

```
    # Evaluate CV accuracy
    acc = accuracy_score(y_cv, y_cv_pred, normalize = True ) * float(100)
    print("\n CV accuracy for k = %d is %d%%" %(i, acc))

    #*****************************
       """
plt.plot(k, train_auc, label = "Train AUC")
plt.plot(k ,cv_auc, label = "CV AUC")

plt.scatter(k, train_auc, label = "Train AUC points")
plt.scatter(k, cv_auc, label = "CV AUC points")

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

```
100%|██████████████████████████████████████| 5/5 [31:58<00:00, 391.55s/it]
```



Testing the performance of the model on test data, plotting ROC Curves

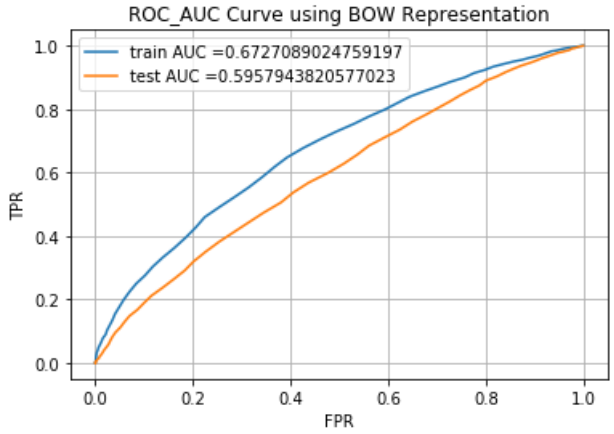
In [200]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap between t
he train and cv is less
# Note: based on the method you use you might get different hyperparameter values as best one
# so, you choose according to the method you choose, you use gridsearch if you are having more com
puting power and note it will take more time
# if you increase the cv values in the GridSearchCV you will get more rebust results.

#here we are choosing the best_k based on forloop results
best_k = 121
```

In [201]:

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


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

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

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
```

```
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC_AUC Curve using BOW Representation")
plt.grid()
plt.show()
```



In [202]:

```
# the function to predict, with defined threshold
# we will pick a threshold that will give the least fpr

def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    # print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold ", np.round(t,
3))
    return t

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

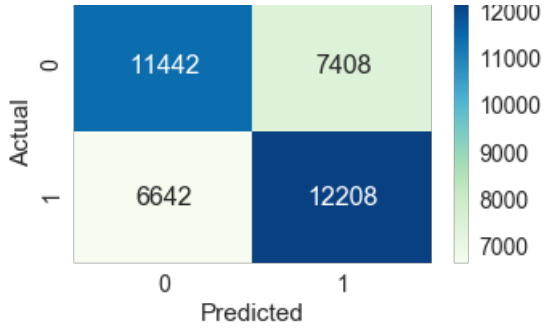
In [79]:

```
# confusion matrix for classifiers : https://www.kaggle.com/agungor2/various-confusion-matrix-plot
s

def getheatmapconfusionmat(data, y_value):
    df_cm = pd.DataFrame(data, columns = np.unique(y_value), index = np.unique(y_train))
    df_cm.index.name = "Actual"
    df_cm.columns.name = "Predicted"
    plt.figure(figsize = (5,3))
    sns.set(font_scale = 1.4) # label size
    sns.heatmap(df_cm, cmap = "GnBu", annot = True, annot_kws ={"size":16}, fmt = 'd') # font size
```

In [204]:

```
from sklearn.metrics import confusion_matrix


best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("The Train Confusion Matrix")
data1 = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
getheatmapconfusionmat(data1, y_train)
```

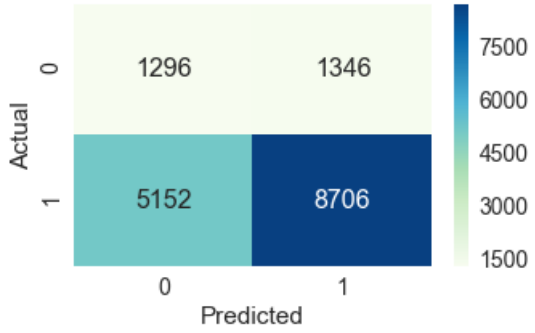
```
The Train Confusion Matrix
```



In [205]:

```
print("The Test Confusion Matrix ")
data2 = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
getheatmapconfusionmat(data2, y_train)
```

```
The Test Confusion Matrix
```



TFIDF Making Model Ready:

For Project Essays

In [206]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2
vectorizer = TfidfVectorizer(min_df = 10)
vectorizer.fit(X_train['preprocessed_essays'].values) # Fitting made on train data

# We are using TFIDFVectorizer to convert text to vector for only text features
X_train_essay_tfidf = vectorizer.transform(X_train["preprocessed_essays"].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv["preprocessed_essays"].values)
X_test_essay_tfidf = vectorizer.transform(X_test["preprocessed_essays"].values)

selector = SelectKBest(chi2, k = 2000)
selector.fit(X_train_essay_tfidf, y_train)
X_train_essay_2000 = selector.transform(X_train_essay_tfidf)
X_cv_essay_2000 = selector.transform(X_cv_essay_tfidf)
X_test_essay_2000 = selector.transform(X_test_essay_tfidf)

print("AFter TFIDF Vectorizing of project essays ")
print(X_train_essay_2000.shape)
print(X_cv_essay_2000.shape)
print(X_test_essay_2000.shape)
```

```
AFter TFIDF Vectorizing of project essays
(37700, 2000)
(11055, 2000)
(16500, 2000)
```

For project titles

In [207]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2

vectorizer = TfidfVectorizer(min_df = 5)
vectorizer.fit(X_train["preprocessed_titles"].values) # fit has to done only on train data

# We are using TfidfVectorizer to convert text to vector
X_train_title_tfidf = vectorizer.transform(X_train["preprocessed_titles"].values)
X_cv_title_tfidf = vectorizer.transform(X_cv["preprocessed_titles"].values)
X_test_title_tfidf = vectorizer.transform(X_test["preprocessed_titles"].values)

print("After TFIDF Vectorizing of Project Titles" )
print(X_train_title_tfidf.shape)
print(X_cv_title_tfidf.shape)
print(X_test_title_tfidf.shape)
```

```
After TFIDF Vectorizing of Project Titles
(37700, 2930)
(11055, 2930)
(16500, 2930)
```

For Resource Summary

In [208]:

```
vectorizer = TfidfVectorizer(min_df = 5)
vectorizer.fit(X_train["preprocessed_resource_summary"].values)

# We use the fitted Tfidf Vectorizer to convert text to vector
X_train_summary_tfidf = vectorizer.transform(X_train["preprocessed_resource_summary"].values)
X_cv_summary_tfidf = vectorizer.transform(X_cv["preprocessed_resource_summary"].values)
X_test_summary_tfidf = vectorizer.transform(X_test["preprocessed_resource_summary"].values)

print("After TFIDF Vectorizing of Resource Summary")
print(X_train_summary_tfidf.shape)
print(X_cv_summary_tfidf.shape)
print(X_test_summary_tfidf.shape)
```

```
After TFIDF Vectorizing of Resource Summary
(37700, 5352)
(11055, 5352)
(16500, 5352)
```

Concatenating All Features (TFIDF)

In [209]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039

from scipy.sparse import hstack

X_tr = hstack((X_train_essay_2000, X_train_title_tfidf, X_train_summary_tfidf, \
               X_train_categories_one, X_train_subcategories_one, X_train_grade_one,
X_train_prefix_one, X_train_state_one, \
               X_train_price_norm, X_train_quantity_norm, X_train_teacher_previously_norm)).tocsr()

X_cr = hstack((X_cv_essay_2000, X_cv_title_tfidf, X_cv_summary_tfidf, \
               X_cv_categories_one, X_cv_subcategories_one, X_cv_grade_one, X_cv_prefix_one, X_cv_st
ate_one, \
               X_cv_price_norm, X_cv_quantity_norm, X_cv_teacher_previously_norm)).tocsr()

X_te = hstack((X_test_essay_2000, X_test_title_tfidf, X_test_summary_tfidf, \
               X_test_categories_one, X_test_subcategories_one, X_test_grade_one, X_test_prefix_one,
X_test_state_one, \
               X_test_price_norm, X_test_quantity_norm, X_test_teacher_previously_norm)).tocsr()

print("Final Data Matrix")
```

```
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
```

```
Final Data Matrix
(37700, 10385) (37700,)
(11055, 10385) (11055,)
(16500, 10385) (16500,)
```

3.2 Applying KNN brute force on TFIDF, SET 2 on 50k data points

In [210]:

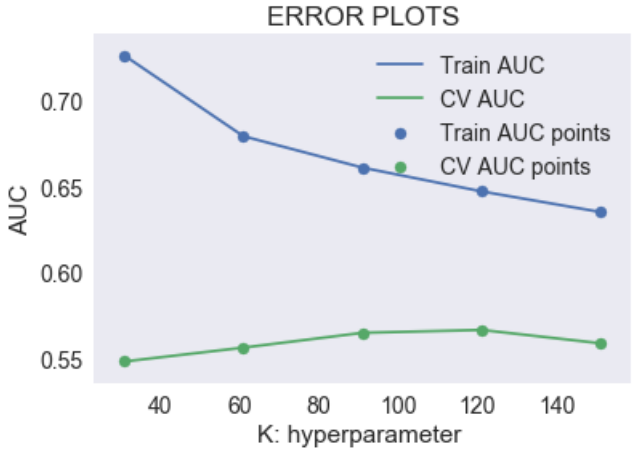
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs

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

    return y_data_pred
```

In [211]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []

#k = [3,15,25,51, 101]
k = [31, 61, 91, 121, 151]
for i in tqdm(k):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs = -1)
    neigh.fit(X_tr, y_train)

    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs

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

plt.plot(k, train_auc, label = "Train AUC")
plt.plot(k ,cv_auc, label = "CV AUC")

plt.scatter(k, train_auc, label = "Train AUC points")
plt.scatter(k, cv_auc, label = "CV AUC points")
```

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

```
100%|██████████████████████████████████████████| 5/5 [31:14<00:00, 372.92s/it]
```

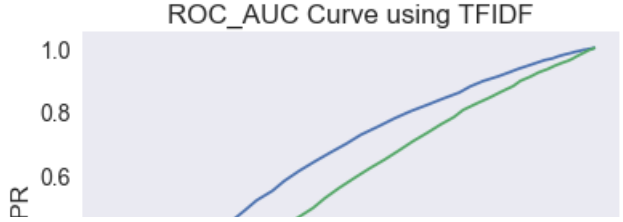


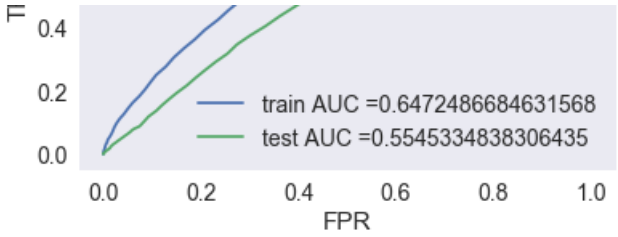
In [213]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap between t
he train and cv is less
# Note: based on the method you use you might get different hyperparameter values as best one
# so, you choose according to the method you choose, you use gridsearch if you are having more com
puting power and note it will take more time
# if you increase the cv values in the GridSearchCV you will get more rebust results.

#here we are choosing the best_k based on forloop results
best_k = 121
```

In [214]:

```
from sklearn.metrics import roc_curve, auc


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

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

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

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





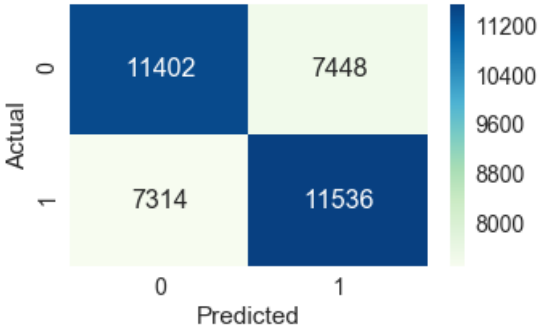
In [215]:

```
# Function to predict, with defined threshold
# we will pick a threshold that give the least fpr

def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr * (1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("The maximum values of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

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

In [216]:

```
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("The Train Confusion Matrix")
data1 = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
getheatmapconfusionmat(data1, y_train)
```

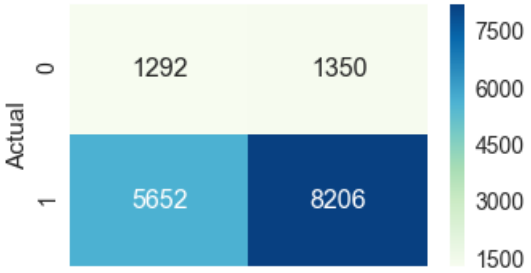
```
The maximum values of tpr*(1-fpr) 0.3701805317704339 for threshold 0.479
The Train Confusion Matrix
```



In [217]:

```
print("The Test Confusion Matrix ")
data2 = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
getheatmapconfusionmat(data2, y_train)
```

```
The Test Confusion Matrix
```



0 1

Predicted

Making Model ready Average W2V

Using Above Codes for vectorization of categorical data, numerical data and applying Average W2V for Text data with 10k points.

Project Essays

In [61]:

```
with open('glove_vectors', "rb") as f:
    model = pickle.load(f)
    glove_words = set(model.keys())
```

In [62]:

```
# average w2v # computing average word to vec for each review

avg_w2v_vectors_train_essay = [];

for sentence in tqdm(X_train["preprocessed_essays"].values):
    vector = np.zeros(300)
    cnt_words = 0
    for word in sentence.split():
        if word in glove_words:
            vector += model[word]
            cnt_words += 1

    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train_essay.append(vector)

print(len(avg_w2v_vectors_train_essay))
print(len(avg_w2v_vectors_train_essay[0]))
```

```
100%|████████████████████████████████████| 7738/7738 [00:06<00:00, 1242.78it/s]
```

```
7738
300
```

In [63]:

```
avg_w2v_vectors_cv_essay = [];

for sentence in tqdm(X_cv["preprocessed_essays"].values):
    vector = np.zeros(300)
    cnt_words = 0
    for word in sentence.split():
        if word in glove_words:
            vector += model[word]
            cnt_words +=1

    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_cv_essay.append(vector)

print(len(avg_w2v_vectors_cv_essay))
print(len(avg_w2v_vectors_cv_essay[0]))
```

```
100%|████████████████████████████████████| 2211/2211 [00:01<00:00, 1793.08it/s]
```

```
2211
300
```

In [64]:

```
avg_w2v_vectors_test_essay = [];

for sentence in tqdm(X_test["preprocessed_essays"].values):
    vector = np.zeros(300)
    cnt_words = 0
    for word in sentence.split():
        if word in glove_words:
            vector += model[word]
            cnt_words +=1

    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test_essay.append(vector)

print(len(avg_w2v_vectors_test_essay))
print(len(avg_w2v_vectors_test_essay[0]))
```

```
100%|██████████████████████████████████| 3300/3300 [00:01<00:00, 1775.05it/s]
```

```
3300
300
```

project Title

In [65]:

```
avg_w2v_vectors_train_title = [];

for sentence in tqdm(X_train["preprocessed_titles"].values):
    vector = np.zeros(300)
    cnt_words = 0
    for word in sentence.split():
        if word in glove_words:
            vector += model[word]
            cnt_words += 1

    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train_title.append(vector)

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

```
100%|██████████████████████████████████| 7738/7738 [00:00<00:00, 29645.81it/s]
```

```
7738
300
```

In [66]:

```
avg_w2v_vectors_cv_title = [];

for sentence in tqdm(X_cv["preprocessed_titles"].values):
    vector = np.zeros(300)
    cnt_words = 0
    for word in sentence.split():
        if word in glove_words:
            vector += model[word]
            cnt_words += 1

    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_cv_title.append(vector)

print(len(avg_w2v_vectors_cv_title))
print(len(avg_w2v_vectors_cv_title[0]))
```

```
100%|██████████████████████████████████| 2211/2211 [00:00<00:00, 26961.92it/s]
```

```
2211
300
```

In [67]:

```
avg_w2v_vectors_test_title = [];

for sentence in tqdm(X_test["preprocessed_titles"].values):
    vector = np.zeros(300)
    cnt_words = 0
    for word in sentence.split():
        if word in glove_words:
            vector += model[word]
            cnt_words += 1

    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test_title.append(vector)

print(len(avg_w2v_vectors_test_title))
print(len(avg_w2v_vectors_test_title[0]))
```

```
100%|██████████████████████████████████| 3300/3300 [00:00<00:00, 26188.96it/s]
```

```
3300
300
```

Resource Summary

In [68]:

```
avg_w2v_vectors_train_summary = [];

for sentence in tqdm(X_train["preprocessed_resource_summary"].values):
    vector = np.zeros(300)
    cnt_words = 0
    for word in sentence.split():
        if word in glove_words:
            vector += model[word]
            cnt_words += 1

    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train_summary.append(vector)

print(len(avg_w2v_vectors_train_summary))
print(len(avg_w2v_vectors_train_summary[0]))
```

```
100%|██████████████████████████████████| 7738/7738 [00:00<00:00, 14682.26it/s]
```

```
7738
300
```

In [69]:

```
avg_w2v_vectors_cv_summary = [];

for sentence in tqdm(X_cv["preprocessed_resource_summary"].values):
    vector = np.zeros(300)
    cnt_words = 0
    for word in sentence.split():
        if word in glove_words:
            vector += model[word]
            cnt_words += 1

    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_cv_summary.append(vector)
```

```
print(len(avg_w2v_vectors_cv_summary))
print(len(avg_w2v_vectors_cv_summary[0]))
```

```
100%|██████████████████████████████████| 2211/2211 [00:00<00:00, 13647.35it/s]
```

```
2211
300
```

In [70]:

```
avg_w2v_vectors_test_summary = [];

for sentence in tqdm(X_test["preprocessed_resource_summary"].values):
    vector = np.zeros(300)
    cnt_words = 0
    for word in sentence.split():
        if word in glove_words:
            vector += model[word]
            cnt_words += 1

    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test_summary.append(vector)

print(len(avg_w2v_vectors_test_summary))
print(len(avg_w2v_vectors_test_summary[0]))
```

```
100%|██████████████████████████████████| 3300/3300 [00:00<00:00, 15067.62it/s]
```

```
3300
300
```

Concatenating all features

In [71]:

```
from scipy.sparse import hstack

X_tr = hstack((avg_w2v_vectors_train_essay, avg_w2v_vectors_train_title,
avg_w2v_vectors_train_summary, \
               X_train_categories_one, X_train_subcategories_one, X_train_state_one,
X_train_prefix_one, X_train_grade_one, \
               X_train_price_norm, X_train_quantity_norm, X_train_teacher_previously_norm)).tocsr()

X_cr = hstack((avg_w2v_vectors_cv_essay, avg_w2v_vectors_cv_title, avg_w2v_vectors_cv_title, \
               X_cv_categories_one, X_cv_subcategories_one, X_cv_state_one, X_cv_grade_one,
X_cv_prefix_one, \
               X_cv_price_norm, X_cv_quantity_norm, X_cv_teacher_previously_norm)).tocsr()

X_te = hstack((avg_w2v_vectors_test_essay, avg_w2v_vectors_test_title,
avg_w2v_vectors_test_summary, \
               X_test_categories_one, X_test_subcategories_one, X_test_grade_one, X_test_prefix_one,
X_test_state_one, \
               X_test_price_norm, X_test_quantity_norm, X_test_teacher_previously_norm)).tocsr()

print("Final data Matrix for Avg W2V ")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
```

```
Final data Matrix for Avg W2V
(7738, 999) (7738,)
(2211, 999) (2211,)
(3300, 999) (3300,)
```

3.3 Applying KNN brute force on AVG W2V, SET 3 on 10k data points

Hyperparameter tunning

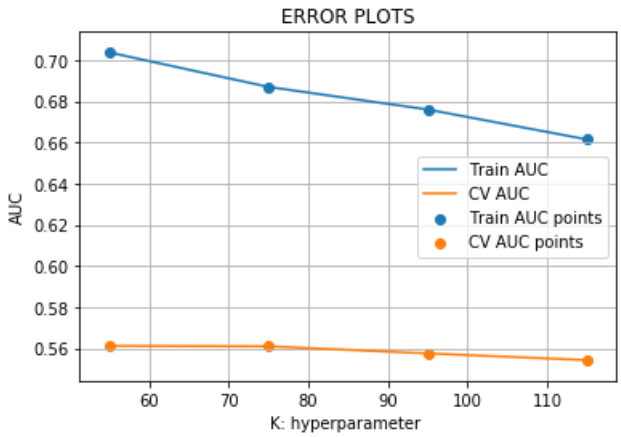
In [72]:

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

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

    return y_data_pred
```

In [73]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []

# k = [15, 35, 55, 75, 95]
k = [55, 75, 95, 115]
for i in tqdm(k):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs = -1)
    neigh.fit(X_tr, y_train)

    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs

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

plt.plot(k, train_auc, label = "Train AUC")
plt.plot(k ,cv_auc, label = "CV AUC")

plt.scatter(k, train_auc, label = "Train AUC points")
plt.scatter(k, cv_auc, label = "CV AUC points")

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

```
100%|██████████████████████████████████████████| 4/4 [15:48<00:00, 230.58s/it]
```



In [247]:

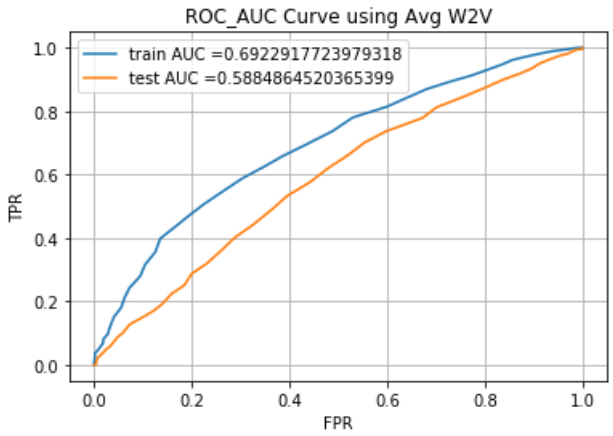
```
### Testing the prerformance of the model on test data, plotting ROC curves
```

In [74]:

```
best_k = 65
```

In [75]:

```
from sklearn.metrics import roc_curve, auc


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

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

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC_AUC Curve using Avg W2V")
plt.grid()
plt.show()
```



In [76]:

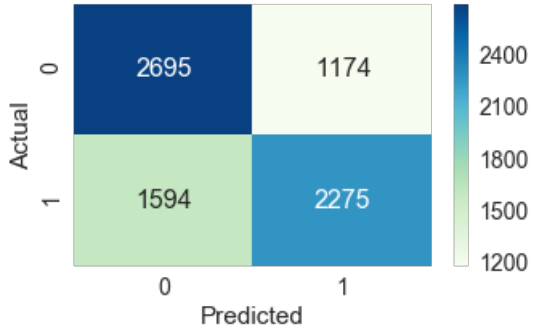
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
```

```
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

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

In [80]:

```
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("The Train Confusion Matrix")
data1 = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
getheatmapconfusionmat(data1, y_train)
```

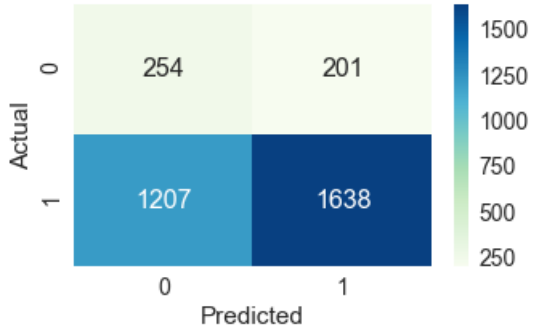
```
the maximum value of tpr*(1-fpr) 0.4095837435377975 for threshold 0.508
The Train Confusion Matrix
```



In [81]:

```
print("The Test Confusion Matrix ")
data2 = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
getheatmapconfusionmat(data2, y_train)
```

```
The Test Confusion Matrix
```



Making model ready for TFIDF weighted W2V

Using Above Codes for vectorization of categorical data, numerical data and applying TFIDF weighted W2V for Text data on 10k points.

In [82]:

```
tfidf_model = TfidfVectorizer()
```

```
tfidf_model.fit(X_train["preprocessed_essays"].values)

dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [83]:

```
tfidf_weighted_w2v_train_essay = [];

for sentence in tqdm(X_train["preprocessed_essays"].values):
    vector = np.zeros(300)
    tf_idf_weight = 0
    for word in sentence.split():
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word]

            tf_idf = dictionary[word] * (sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf)
            tf_idf_weight += tf_idf

    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_weighted_w2v_train_essay.append(vector)

print(len(tfidf_weighted_w2v_train_essay))
print(len(tfidf_weighted_w2v_train_essay[0]))
```

```
100%|██████████████████████████████████████| 7738/7738 [00:51<00:00, 151.00it/s]
```

```
7738
300
```

In [84]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_cv["preprocessed_essays"].values)

dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [85]:

```
tfidf_weighted_w2v_cv_essay = [];

for sentence in tqdm(X_cv["preprocessed_essays"].values):
    vector = np.zeros(300)
    tf_idf_weight = 0;
    for word in sentence.split():
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word]

            tf_idf = dictionary[word] * (sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf)
            tf_idf_weight += tf_idf

    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_weighted_w2v_cv_essay.append(vector)

print(len(tfidf_weighted_w2v_cv_essay))
print(len(tfidf_weighted_w2v_cv_essay[0]))
```

```
100%|██████████████████████████████████████| 2211/2211 [00:17<00:00, 127.92it/s]
```

```
2211
300
```

In [86]:

```
tfidf_model = TfidfVectorizer()
```

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_test["preprocessed_essays"].values)

dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [87]:

```
tfidf_weighted_w2v_test_essay = [];

for sentence in tqdm(X_test["preprocessed_essays"].values):
    vector = np.zeros(300)
    tf_idf_weight = 0;

    for word in sentence.split():
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word]

            tf_idf = dictionary[word] * (sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf)
            tf_idf_weight += tf_idf

    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_weighted_w2v_test_essay.append(vector)

print(len(tfidf_weighted_w2v_test_essay))
print(len(tfidf_weighted_w2v_test_essay[0]))
```

```
100%|██████████████████████████████████████| 3300/3300 [00:23<00:00, 138.36it/s]
```

```
3300
300
```

project title

In [88]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train["preprocessed_titles"].values)

dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [89]:

```
tfidf_weighted_w2v_train_title = [];

for sentence in tqdm(X_train["preprocessed_titles"].values):
    vector = np.zeros(300)
    tf_idf_weight = 0;
    for word in sentence.split():
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word]

            tf_idf = dictionary[word] * (sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf)
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_weighted_w2v_train_title.append(vector)

print(len(tfidf_weighted_w2v_train_title))
print(len(tfidf_weighted_w2v_train_title[0]))
```

```
100%|██████████████████████████████████████| 7738/7738 [00:01<00:00, 5221.02it/s]
```

```
7738
300
```

In [90]:

```
tfidf_model =  TfidfVectorizer()
tfidf_model.fit(X_cv["preprocessed_titles"].values)
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [91]:

```
tfidf_weighted_w2v_cv_title = [];
for sentence in tqdm(X_cv["preprocessed_titles"].values):
    vector = np.zeros(300)
    tf_idf_weight = 0
    for word in sentence.split():
        if(word in glove_words) and (word in tfidf_words):
            vec = model[word]
            tf_idf = dictionary[word] * (sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf)
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_weighted_w2v_cv_title.append(vector)

print(len(tfidf_weighted_w2v_cv_title))
print(len(tfidf_weighted_w2v_cv_title[0]))
```

```
100%|██████████████████████████████████| 2211/2211 [00:00<00:00, 4724.09it/s]
```

```
2211
300
```

In [92]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_test["preprocessed_titles"].values)
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [93]:

```
tfidf_weighted_w2v_test_title = [];
for sentence in tqdm(X_test["preprocessed_titles"].values):
    vector = np.zeros(300)
    tf_idf_weight = 0
    for word in sentence.split():
        if(word in glove_words) and (word in tfidf_words):
            vec = model[word]
            tf_idf = dictionary[word] * (sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf)
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_weighted_w2v_test_title.append(vector)
print(len(tfidf_weighted_w2v_test_title))
print(len(tfidf_weighted_w2v_test_title[0]))
```

```
100%|██████████████████████████████████| 3300/3300 [00:00<00:00, 4761.63it/s]
```

```
3300
300
```

Project resource Sumamry

In [94]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train["preprocessed_resource_summary"].values)
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
```

```
tfidf_words = set(tfidf_model.get_feature_names())
```

In [95]:

```
tfidf_weighted_w2v_train_summary = [];

for sentence in tqdm(X_train["preprocessed_resource_summary"].values):
    vector = np.zeros(300)
    tf_idf_weight = 0;
    for word in sentence.split():
        if(word in glove_words) and (word in tfidf_words):
            vec = model[word]

            tf_idf = dictionary[word] * sentence.count(word)/len(sentence.split())
            vector += (vec * tf_idf)
            tf_idf_weight += tf_idf

    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_weighted_w2v_train_summary.append(vector)

print(len(tfidf_weighted_w2v_train_summary))
print(len(tfidf_weighted_w2v_train_summary[0]))
```

```
100%|██████████████████████████████████| 7738/7738 [00:03<00:00, 2331.46it/s]
```

```
7738
300
```

In [96]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_cv["preprocessed_resource_summary"].values)
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [97]:

```
tfidf_weight_w2v_cv_summary = [];

for sentence in tqdm(X_cv["preprocessed_resource_summary"].values):
    vector = np.zeros(300)
    tf_idf_weight = 0;
    for word in sentence.split():
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word]

            tf_idf = dictionary[word]* sentence.count(word)/len(sentence.split())
            vector += (vec * tf_idf)
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_weight_w2v_cv_summary.append(vector)

print(len(tfidf_weight_w2v_cv_summary))
print(len(tfidf_weight_w2v_cv_summary[0]))
```

```
100%|██████████████████████████████████| 2211/2211 [00:00<00:00, 3278.36it/s]
```

```
2211
300
```

In [98]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_test["preprocessed_resource_summary"].values)
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [99]:

```
tfidf_weighted_w2v_test_summary =[];

for sentence in tqdm(X_test["preprocessed_resource_summary"].values):
    vector = np.zeros(300)
    tf_idf_weight = 0;
    for word in sentence.split():
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word]

            tf_idf = dictionary[word] * (sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf)
            tf_idf_weight += tf_idf

    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_weighted_w2v_test_summary.append(vector)

print(len(tfidf_weighted_w2v_test_summary))
print(len(tfidf_weighted_w2v_test_summary[0]))
```

```
100%|████████████████████████████████████| 3300/3300 [00:00<00:00, 4296.75it/s]
```

```
3300
300
```

Concatenating All Features

In [100]:

```
from scipy.sparse import hstack

X_tr = hstack((tfidf_weighted_w2v_train_essay, tfidf_weighted_w2v_train_title,
tfidf_weighted_w2v_train_title, \
               X_train_categories_one, X_train_subcategories_one, X_train_grade_one,
X_train_prefix_one, X_train_state_one,\
               X_train_quantity_norm, X_train_teacher_previously_norm, X_train_price_norm)).tocsr()

X_cr = hstack((tfidf_weighted_w2v_cv_essay, tfidf_weighted_w2v_cv_title,
tfidf_weight_w2v_cv_summary, \
              X_cv_categories_one, X_cv_subcategories_one, X_cv_grade_one, X_cv_state_one,
X_cv_prefix_one,\
               X_cv_quantity_norm, X_cv_price_norm, X_cv_teacher_previously_norm)).tocsr()

X_te = hstack((tfidf_weighted_w2v_test_essay, tfidf_weighted_w2v_test_title,
tfidf_weighted_w2v_test_summary, \
               X_test_categories_one, X_test_subcategories_one, X_test_grade_one, X_test_prefix_one,
X_test_state_one, \
               X_test_quantity_norm, X_test_price_norm, X_test_teacher_previously_norm)).tocsr()

print("Final Data Matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
```

```
Final Data Matrix
(7738, 999) (7738,)
(2211, 999) (2211,)
(3300, 999) (3300,)
```

3.4 Applying KNN brute force on TFIDF W2V, SET 4 on 10k data points

In [101]:

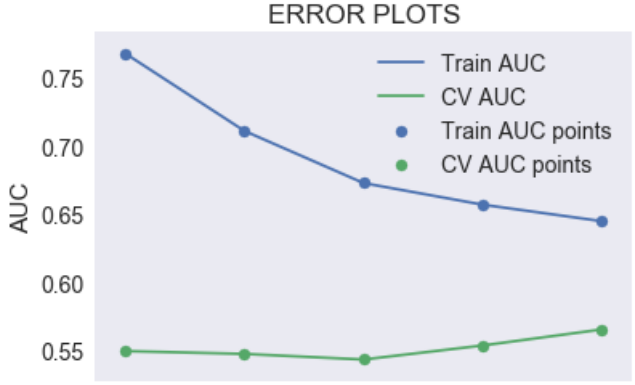
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
```

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

    return y_data_pred
```

In [102]:

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

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [21, 41, 61, 81, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X_tr, y_train)

    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_cr)

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

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

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

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

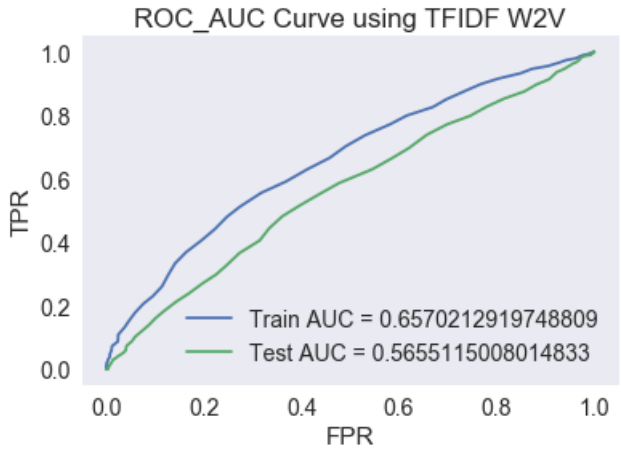
```
100%|██████████████████████████████████████████| 5/5 [19:52<00:00, 238.60s/it]
```



Testing the performance of the model on the test data, plotting ROC curves

In [103]:

```
best_k = 81


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

neigh = KNeighborsClassifier(n_neighbors = best_k, n_jobs = -1)
neigh.fit(X_tr, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

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

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

plt.plot(train_fpr, train_tpr, label = "Train AUC = "+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label = "Test AUC = "+str(auc(test_fpr, test_tpr)))

plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC_AUC Curve using TFIDF W2V")
plt.grid()
plt.show()
```



In [104]:

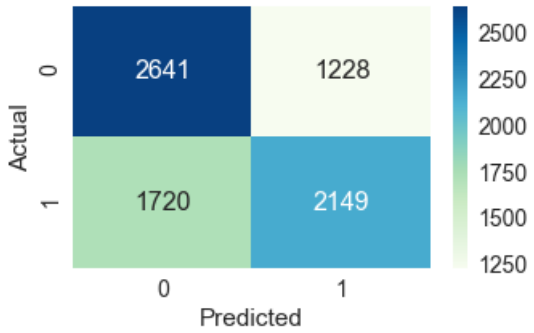
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

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

```
    return predictions
```

In [105]:

```
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("The Train Confusion Matrix")
data1 = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
getheatmapconfusionmat(data1, y_train)
```

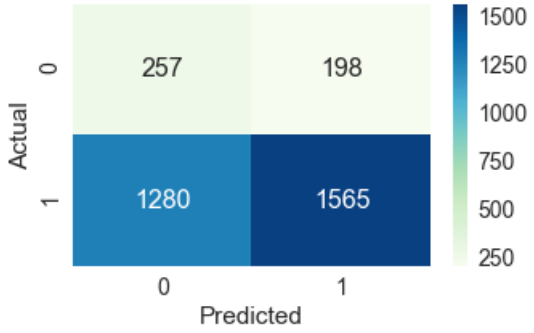
```
the maximum value of tpr*(1-fpr) 0.37914676714346246 for threshold 0.506
The Train Confusion Matrix
```



In [106]:

```
print("The Test Confusion Matrix ")
data2 = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
getheatmapconfusionmat(data2, y_train)
```

```
The Test Confusion Matrix
```



In [108]:

```
#http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "Hyperparameter", "AUC"]
x.add_row(["BOW", 121, 0.59])
x.add_row(["TFIDF", 121, 0.55])
x.add_row(["TFIDF W2V", 65, 0.58])
x.add_row(["TFIDF AVG W2V", 81, 0.56])

print(x)
```

```
+---------------+----------------+------+
|   Vectorizer  | Hyperparameter | AUC  |
+---------------+----------------+------+
|      BOW      |      121       | 0.59 |
|     TFIDF     |      121       | 0.55 |
|   TFIDF W2V   |       65       | 0.58 |
| TFIDF AVG W2V |       81       | 0.56 |
+---------------+----------------+------+
```

Conclusion

- The results vary from BOW, TFIDF compared to TFIDF w2v and TFIDF Avg w2v because of different data points 50,000k
and 10,000k data points respectively.

In [ ]:

In [ ]:

In [ ]: