In [1]:

```
# Importing all the necessary libraries and 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
import re
import string
import pickle
import os

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

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


from tqdm import tqdm
from chart_studio import plotly  #Importing plotly from chart_studio as plotly is deprecated
according to jupyter
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

Reading Data

In [2]:

```
# Importing data with pandas
# For avoid memory issues and to reduce run time I'm only taking 70k points

project_data = pd.read_csv('train_data.csv', nrows= 70000)
resource_data = pd.read_csv('resources.csv')

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

```
Number of data points in train data (70000, 17)

 --------------------------------------------------

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

In [3]:

In [3]:

```
# 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()
```

Out[3]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_s
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	2016-04-27 00:46:53	Grades PreK-2	L
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2	
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5	L
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-27 01:10:09	Grades 3-5	Math &

In [4]:

```
# Printing total no. of data points in Resource Data and the features it have.

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

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

Out[4]:

	id	description	quantity	price
0	p233245	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.00
1	p069063	Bouncy Bands for Desks (Blue support pipes)	3	14.95
2	p069063	Cory Stories: A Kid's Book About Living With Adhd	1	8.45
3	p069063	Dixon Ticonderoga Wood-Cased #2 HB Pencils, Bo...	2	13.59
4	p069063	EDUCATIONAL INSIGHTS FLUORESCENT LIGHT FILTERS...	3	24.95

Preprocessing project_subject_categories

In [5]:

```
# 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

catogories = list(project_data['project_subject_categories'].values)
cat_list = []
for i in catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split():
            j=j.replace('The','')
        j = j.replace(' ','')
        temp+=j.strip()+" "
        temp = temp.replace('&','_')
    cat_list.append(temp.strip())

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

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

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

Preprocessing project_subject_subcategories

In [6]:

```
# 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_catogories = list(project_data['project_subject_subcategories'].values)
sub_cat_list = []
for i in sub_catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split():
            j=j.replace('The','')
        j = j.replace(' ','')
        temp +=j.strip()+" "
        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)

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]))
```

Preprocessing of Project_grade_category

In [7]:

```
# Preprocessing Project_grade_category
# Removing Special characters and 'Grade' word to make this category ready for the vectorization

sub_grade = list(project_data['project_grade_category'].values)
grade_cat_list = []
for i in sub_grade:
    for j in i.split(' '):
        j=j.replace('Grades','')
        j=j.replace('-', '_')
    grade_cat_list.append(j.lower().strip())

project_data['clean_grade_category'] = grade_cat_list
project_data.drop(['project_grade_category'], axis=1, inplace=True)

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

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

In [8]:

```
# Printing top values to see the changes and our updated data

project_data.head()
```

Out[8]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_title	project_essay_1	pr
55660	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 ...	
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	2016-04-27 00:46:53	Mobile Learning with a Mobile Listening Center	Having a class of 24 students comes with diver...	I
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Flexible Seating for Flexible Learning	I recently read an article about giving studen...	
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Going Deep: The Art of Inner Thinking!	My students crave challenge, they eat obstacle...	W
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-27 01:10:09	Breakout Box to Ignite Engagement!	It's the end of the school year. Routines have...	

Merging Project_essay

In [9]:

```
# Merging all the subcategory of project_data into one category

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)

# Printing top values to see the updated Data
project_data.head()
```

Out[9]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_title	project_essay_1	pr
55660	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 ...	
							Mobile	Having a class of	I

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_title	project_essay_1	pr
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	2016-04-27 00:46:53	Learning with a Mobile Listening Center	24 students comes with diver...	
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Flexible Seating for Flexible Learning	I recently read an article about giving studen...	
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Going Deep: The Art of Inner Thinking!	My students crave challenge, they eat obstacle...	W
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-27 01:10:09	Breakout Box to Ignite Engagement!	It's the end of the school year. Routines have...	

In [10]:

```
# Merging price from resource_data to project_data before splitiing the data

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')

# 'techer_prefix' has some missing values so we're filling it with the most common value which is
'Mrs.'
project_data["teacher_prefix"].fillna("Mrs.", inplace= True)
```

Splitting Data in train, CV and test data

In [11]:

```
# I have divided my train, cv and test in 60:25:20
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(project_data,
project_data['project_is_approved'], test_size=0.20, stratify=project_data['project_is_approved'])
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.25, stratify=y_train)
```

In [12]:

```
# Printing no. of total values my Train, Cv and Test data have

print(y_train.value_counts())
print(y_test.value_counts())
print(y_cv.value_counts())
```

```
1    35631
0     6369
Name: project_is_approved, dtype: int64
1    11877
0     2123
Name: project_is_approved, dtype: int64
1    11877
0     2123
Name: project_is_approved, dtype: int64
```

Observations

- As we can see that we have an imbalance dataset and that leads to the failure of KNN
- So to avoid this problem we need to perform upsampling

Upsampling the data

In [13]:

```
# Dividing data into majority and minority so that we can upsample minority class

majority_data = X_train[X_train.project_is_approved==1]
minority_data = X_train[X_train.project_is_approved==0]
```

In [14]:

```
from sklearn.utils import resample

minority_data_upsampled = resample(minority_data, replace=True, n_samples=35631, random_state=10)
x_train_upsampled = pd.concat([majority_data, minority_data_upsampled])

# After applying Upsampling checking and printing total no. of datapoints for each class (i.e 0 an
d 1 class)
x_train_upsampled.project_is_approved.value_counts()
```

Out[14]:

```
1    35631
0    35631
Name: project_is_approved, dtype: int64
```

In [15]:

```
# Updating y_train according to the upsampled data

y_train_upsampled = x_train_upsampled.project_is_approved
print(y_train_upsampled.value_counts())
```

```
1    35631
0    35631
Name: project_is_approved, dtype: int64
```

In [16]:

```
# Dropping 'project_is_approved' column form cv and test data

X_train.drop(["project_is_approved"], axis = 1, inplace = True)
x_train_upsampled.drop(["project_is_approved"], axis = 1, inplace = True)
X_test.drop(["project_is_approved"], axis = 1, inplace = True)
X_cv.drop(["project_is_approved"], axis = 1, inplace = True)
```

Preparing Data for model

In [17]:

```
# Printing All the features after preprocessing data

project_data.columns
```

Out[17]:

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

Text preprocessing for Train, CV and Train Data

Preprocessing of Project_essay

In [18]:

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

def decontracted(phrase):
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)
    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 [19]:

```
# https://gist.github.com/sebleier/554280

stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [20]:

```
# Preprocessing Project_essay on Train_data

train_preprocessed_essays = []

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

```
100%|██████████████████████████████████████████████████████████████████████| 71262/71262 [01:
16<00:00, 926.92it/s]
```

In [21]:

In [21]:

```
# Preprocessing Project_essay on CV data

cv_preprocessed_essays = []

for sentance in tqdm(X_cv['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    cv_preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████| 14000/14000 [00:
15<00:00, 915.35it/s]
```

In [22]:

```
# Preprocessing Project_essay on Test data

test_preprocessed_essays = []

for sentance in tqdm(X_test['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    test_preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████| 14000/14000 [00:
15<00:00, 915.50it/s]
```

Preprocessing Project_title

In [23]:

```
def decontracted2(phrase):
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\s'", "s", phrase)
    return phrase
```

In [24]:

```
# Preprocessing Project_title on Train_data

train_preprocessed_title = []
for title in tqdm(x_train_upsampled['project_title'].values):
    sent = decontracted2(title)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    train_preprocessed_title.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████████████| 71262/71262
[00:02<00:00, 27088.13it/s]
```

In [25]:

```
# Preprocessing Project_title on CV_data

cv_preprocessed_title = []
for title in tqdm(X_cv['project_title'].values):
    sent = decontracted2(title)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    cv_preprocessed_title.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████| 14000/14000
[00:00<00:00, 32009.80it/s]
```

In [26]:

```
# Preprocessing Project_title on Test_data

test_preprocessed_title = []
for title in tqdm(X_test['project_title'].values):
    sent = decontracted2(title)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    test_preprocessed_title.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████| 14000/14000
[00:00<00:00, 30079.74it/s]
```

Vectorizing Categorical Data

One-hot encoding on clean_categories

In [27]:

```
# Performing one-hot encoding on clean_categories for Train, CV and Test Data

vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True
)
vectorizer.fit(x_train_upsampled['clean_categories'].values)

X_train_cat_onehot = vectorizer.transform(x_train_upsampled['clean_categories'].values)
X_cv_cat_onehot = vectorizer.transform(X_cv['clean_categories'].values)
X_test_cat_onehot = vectorizer.transform(X_test['clean_categories'].values)

print(vectorizer.get_feature_names())
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
```

In [28]:

```
print("Printing shape of Train, CV and Test data after vectorizing clean_categories")

print(X_train_cat_onehot.shape, y_train_upsampled.shape)
print(X_cv_cat_onehot.shape, y_cv.shape)
print(X_test_cat_onehot.shape, y_test.shape)
```

```
Printing shape of Train, CV and Test data after vectorizing clean_categories
(71262, 9) (71262,)
(14000, 9) (14000,)
(14000, 9) (14000,)
```

one-hot encoding on clean_sub_categories

In [29]:

```
# Performing one-hot encoding on clean_sub_categories for Train, CV and Test Data
```

```
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=
True)
vectorizer.fit(x_train_upsampled['clean_subcategories'].values)

X_train_subcat_onehot = vectorizer.transform(x_train_upsampled['clean_subcategories'].values)
X_cv_subcat_onehot = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_subcat_onehot = vectorizer.transform(X_test['clean_subcategories'].values)
print(vectorizer.get_feature_names())
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'History_Geography', 'Music', 'Health_LifeScience', 'EarlyDevelopment', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
```

In [30]:

```
print("Printing shape of Train, CV and Test data after vectorizing clean_subcategories")

print(X_train_subcat_onehot.shape, y_train_upsampled.shape)
print(X_cv_subcat_onehot.shape, y_cv.shape)
print(X_test_subcat_onehot.shape, y_test.shape)
```

```
Printing shape of Train, CV and Test data after vectorizing clean_subcategories
(71262, 30) (71262,)
(14000, 30) (14000,)
(14000, 30) (14000,)
```

One-hot encoding on school_state

In [31]:

```
# Performing one-hot encoding on school_state for Train, CV and Test Data

vectorizer = CountVectorizer(lowercase=False, binary=True)
vectorizer.fit(x_train_upsampled['school_state'].values)

X_train_school_state_onehot = vectorizer.transform(x_train_upsampled['clean_subcategories'].values
)
X_cv_school_state_onehot = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_school_state_onehot = vectorizer.transform(X_test['clean_subcategories'].values)
print(vectorizer.get_feature_names())
```

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'IA', 'ID', 'IL', 'IN', 'K
S', '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']
```

In [32]:

```
print("Printing shape of Train, CV and Test data after vectorizing school_state")

print(X_train_school_state_onehot.shape, y_train_upsampled.shape)
print(X_cv_school_state_onehot.shape, y_cv.shape)
print(X_test_school_state_onehot.shape, y_test.shape)
```

```
Printing shape of Train, CV and Test data after vectorizing school_state
(71262, 51) (71262,)
(14000, 51) (14000,)
(14000, 51) (14000,)
```

one-hot encoding on teacher_prefix

In [33]:

```
# Performing one-hot encoding on teacher_prefix for Train, CV and Test Data

x_train_upsampled["teacher_prefix"].fillna("Mrs.", inplace= True)

vectorizer = CountVectorizer(lowercase=False, binary=True)
vectorizer.fit(x_train_upsampled['teacher_prefix'].values)

X_train_teacher_onehot = vectorizer.transform(x_train_upsampled['teacher_prefix'].values)
X_cv_teacher_onehot = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_onehot = vectorizer.transform(X_test['teacher_prefix'].values)
```

In [34]:

```
print("Printing shape of Train, CV and Test data after vectorizing teacher_prefix")

print(X_train_teacher_onehot.shape, y_train_upsampled.shape)
print(X_cv_teacher_onehot.shape, y_cv.shape)
print(X_test_teacher_onehot.shape, y_test.shape)
```

```
Printing shape of Train, CV and Test data after vectorizing teacher_prefix
(71262, 5) (71262,)
(14000, 5) (14000,)
(14000, 5) (14000,)
```

one-hot encoding on clean_grade_category

In [35]:

```
# Performing one-hot encoding on clean_grade_category for Train, CV and Test Data

vectorizer = CountVectorizer(vocabulary=list(sorted_sub_grade_cat_dict.keys()), lowercase=False, b
inary=True)
vectorizer.fit(x_train_upsampled['clean_grade_category'].values)

X_train_clean_grade_onehot = vectorizer.transform(x_train_upsampled['clean_grade_category'].values
)
X_cv_clean_grade_onehot = vectorizer.transform(X_cv['clean_grade_category'].values)
X_test_clean_grade_onehot = vectorizer.transform(X_test['clean_grade_category'].values)
print(vectorizer.get_feature_names())
```

```
['9_12', '6_8', '3_5', 'prek_2']
```

In [36]:

```
print("Printing shape of Train, CV and Test data after vectorizing clean_grade_category")

print(X_train_clean_grade_onehot.shape, y_train_upsampled.shape)
print(X_cv_clean_grade_onehot.shape, y_cv.shape)
print(X_test_clean_grade_onehot.shape, y_test.shape)
```

```
Printing shape of Train, CV and Test data after vectorizing clean_grade_category
(71262, 4) (71262,)
(14000, 4) (14000,)
(14000, 4) (14000,)
```

Vectorizing Numerical Features

In [37]:

```
# Vectorizing Price Feature
# Before vectorizing we need to standardize our data so performing standard scaler
from sklearn.preprocessing import StandardScaler

price_scalar = StandardScaler()
price_scalar.fit(x_train_upsampled['price'].values.reshape(-1,1))

print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")
X_train_price_std = price_scalar.transform(x_train_upsampled['price'].values.reshape(-1,1))
X_train_price_std = price_scalar.transform(X_train_upsampled['price'].values.reshape(-1,1))
X_cv_price_std = price_scalar.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_std = price_scalar.transform(X_test['price'].values.reshape(-1,1))
```

```
Mean : 324.58197552692883, Standard deviation : 390.04823905809354
```

In [38]:

```
print("Printing shape of Train, CV and Test data after vectorizing price")

print(X_train_price_std.shape, y_train_upsampled.shape)
print(X_cv_price_std.shape, y_cv.shape)
print(X_test_price_std.shape, y_test.shape)
```

```
Printing shape of Train, CV and Test data after vectorizing price
(71262, 1) (71262,)
(14000, 1) (14000,)
(14000, 1) (14000,)
```

In [39]:

```
# Vectorizing teacher_number_of_previously_posted_projects

previously_posted_projects_scalar = StandardScaler()
previously_posted_projects_scalar.fit(x_train_upsampled['teacher_number_of_previously_posted_projec
s'].values.reshape(-1,1))

print(f"Mean : {previously_posted_projects_scalar.mean_[0]}, Standard deviation :
{np.sqrt(previously_posted_projects_scalar.var_[0])}")

X_train_posted_projects_std = previously_posted_projects_scalar.transform(x_train_upsampled['teach
er_number_of_previously_posted_projects'].values.reshape(-1,1))
X_cv_posted_projects_std =
previously_posted_projects_scalar.transform(X_cv['teacher_number_of_previously_posted_projects'].v
alues.reshape(-1,1))
X_test_posted_projects_std =
previously_posted_projects_scalar.transform(X_test['teacher_number_of_previously_posted_projects']
.values.reshape(-1,1))
```

```
Mean : 9.323903342594932, Standard deviation : 24.007673405606965
```

In [40]:

```
print("Printing shape of Train, CV and Test data after vectorizing
teacher_number_of_previously_posted_projects")

print(X_train_posted_projects_std.shape, y_train_upsampled.shape)
print(X_cv_posted_projects_std.shape, y_cv.shape)
print(X_test_posted_projects_std.shape, y_test.shape)
```

```
Printing shape of Train, CV and Test data after vectorizing
teacher_number_of_previously_posted_projects
(71262, 1) (71262,)
(14000, 1) (14000,)
(14000, 1) (14000,)
```

Bag of Words on Project Essay for train, cv and test data

In [41]:

```
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(train_preprocessed_essays)

X_train_essay_bow = vectorizer.transform(train_preprocessed_essays)
X_cv_essay_bow = vectorizer.transform(cv_preprocessed_essays)
X_test_essay_bow = vectorizer.transform(test_preprocessed_essays)
```

In [42]:

In [42]:

```
print("Shape of train_matrix after BoW on project_essay : ", X_train_essay_bow.shape,
y_train_upsampled.shape)
print("\nShape of cv_matrix after BoW on project_essay : ", X_cv_essay_bow.shape, y_cv.shape)
print("\nShape of test_matrix after BoW on project_essay : ", X_test_essay_bow.shape, y_test.shape
)
```

```
Shape of train_matrix after BoW on project_essay :  (71262, 14247) (71262,)

Shape of cv_matrix after BoW on project_essay :  (14000, 14247) (14000,)

Shape of test_matrix after BoW on project_essay :  (14000, 14247) (14000,)
```

Bag of Words on Project title for train, cv and test data

In [43]:

```
vectorizer = CountVectorizer(min_df=6)
vectorizer.fit(train_preprocessed_title)

X_train_title_bow = vectorizer.transform(train_preprocessed_title)
X_cv_title_bow = vectorizer.transform(cv_preprocessed_title)
X_test_title_bow = vectorizer.transform(test_preprocessed_title)
```

In [44]:

```
print("Shape of train_matrix after BoW on project_title : ", X_train_title_bow.shape,
y_train_upsampled.shape)
print("\nShape of cv_matrix after BoW on project_title : ", X_cv_title_bow.shape, y_cv.shape)
print("\nShape of test_matrix after BoW on project_title : ", X_test_title_bow.shape, y_test.shape
)
```

```
Shape of train_matrix after BoW on project_title :  (71262, 3966) (71262,)

Shape of cv_matrix after BoW on project_title :  (14000, 3966) (14000,)

Shape of test_matrix after BoW on project_title :  (14000, 3966) (14000,)
```

Tf-IDF Vectorizer on preprocessed_essays for train, cv and test data

In [45]:

```
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(train_preprocessed_essays)

X_train_essay_tf = vectorizer.transform(train_preprocessed_essays)
X_cv_essay_tf = vectorizer.transform(cv_preprocessed_essays)
X_test_essay_tf = vectorizer.transform(test_preprocessed_essays)
```

In [46]:

```
print("Shape of train_matrix after tfidf on project_essay : ", X_train_essay_tf.shape,
y_train_upsampled.shape)
print("\nShape of cv_matrix after tfidf on project_essay : ", X_cv_essay_tf.shape, y_cv.shape)
print("\nShape of test_matrix after tfidf on project_essay : ", X_test_essay_tf.shape,
y_test.shape)
```

```
Shape of train_matrix after tfidf on project_essay :  (71262, 14247) (71262,)

Shape of cv_matrix after tfidf on project_essay :  (14000, 14247) (14000,)

Shape of test_matrix after tfidf on project_essay :  (14000, 14247) (14000,)
```

Tf-IDF Vectorizer on preprocessed_title for train, cv and test

Tf-IDF Vectorizer on preprocessed_title for train, cv and test data

In [47]:

```
vectorizer = TfidfVectorizer(min_df=6)
vectorizer.fit(train_preprocessed_title)

X_train_title_tf = vectorizer.transform(train_preprocessed_title)
X_cv_title_tf = vectorizer.transform(cv_preprocessed_title)
X_test_title_tf = vectorizer.transform(test_preprocessed_title)
```

In [48]:

```
print("Shape of train_matrix after tfidf on project_title : ", X_train_title_tf.shape,
y_train_upsampled.shape)
print("\nShape of cv_matrix after tfidf on project_title : ", X_cv_title_tf.shape, y_cv.shape)
print("\nShape of test_matrix after tfidf on project_title : ", X_test_title_tf.shape,
y_test.shape)
```

```
Shape of train_matrix after tfidf on project_title :  (71262, 3966) (71262,)

Shape of cv_matrix after tfidf on project_title :  (14000, 3966) (14000,)

Shape of test_matrix after tfidf on project_title :  (14000, 3966) (14000,)
```

Avg W2V on preprocessed_essay for train, cv and test data

In [49]:

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

In [50]:

```
# Avg W2V on Project_essay for Train Data

X_train_essay_avgW2V = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_train_upsampled['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_train_essay_avgW2V.append(vector)

print(len(X_train_essay_avgW2V))
print(len(X_train_essay_avgW2V[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 71262/71262
[00:52<00:00, 1357.00it/s]
```

```
71262
300
```

In [51]:

```
# Avg W2V on Project_essay for CV Data

X_cv_essay_avgW2V = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_cv_essay_avgW2V.append(vector)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 14000/14000
[00:10<00:00, 1336.14it/s]
```

In [52]:

```
# Avg W2V on Project_essay for Test Data

X_test_essay_avgW2V = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_test_essay_avgW2V.append(vector)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 14000/14000
[00:10<00:00, 1309.55it/s]
```

# Avg W2V on preprocessed_title for train, cv and test data

In [53]:

```
# Avg W2V on Project_title for Train Data

X_train_title_avgW2V = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(train_preprocessed_title): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_train_title_avgW2V.append(vector)
```

```
100%|█████████████████████████████████████████████████████████████████████████| 71262/71262
[00:01<00:00, 36894.26it/s]
```

In [54]:

```
# Avg W2V on Project_title for CV Data

X_cv_title_avgW2V = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(cv_preprocessed_title): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_cv_title_avgW2V.append(vector)
```

```
100%|█████████████████████████████████████████████████████████████████████████| 14000/14000
[00:00<00:00, 40623.42it/s]
```

In [55]:

```
# Avg W2V on Project_title for Test Data

X_test_title_avgW2V = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(test_preprocessed_title): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_test_title_avgW2V.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████████████| 14000/14000
[00:00<00:00, 38854.34it/s]
```

Tf-idf weighted W2V on preprocessed_essay for train, cv and test data

In [56]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(train_preprocessed_essays)

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

In [57]:

```
# Tf-Idf W2V on Project_essay for Train Data

X_train_essay_tfidf_W2V = []

for sentence in tqdm(train_preprocessed_essays):
    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
    X_train_essay_tfidf_W2V.append(vector)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 71262/71262 [04:
21<00:00, 272.47it/s]
```

In [58]:

```
# Tf-Idf W2V on Project_essay for CV Data

X_cv_essay_tfidf_W2V = []

for sentence in tqdm(cv_preprocessed_essays):
    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
    X_cv_essay_tfidf_W2V.append(vector)
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 14000/14000 [00:52<00:00, 265.00it/s]
```

In [59]:

```
# Tf-Idf W2V on Project_essay for Test Data

X_test_essay_tfidf_W2V = []

for sentence in tqdm(test_preprocessed_essays):
    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
    X_test_essay_tfidf_W2V.append(vector)
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 14000/14000 [00:53<00:00, 262.19it/s]
```

Tf-idf weighted W2V on preprocessed_title for train, cv and test data

In [60]:

```
# Tf-Idf W2V on Project_title for Train Data

X_train_title_tfidf_W2V = []

for sentence in tqdm(train_preprocessed_title):
    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
    X_train_title_tfidf_W2V.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 71262/71262 [00:04<00:00, 17396.56it/s]
```

In [61]:

```
# Tf-Idf W2V on Project_title for CV Data

X_cv_title_tfidf_W2V = []

for sentence in tqdm(cv_preprocessed_title):
    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
    X_cv_title_tfidf_W2V.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████████████| 14000/14000
[00:00<00:00, 18520.55it/s]
```

In [62]:

```
# Tf-Idf W2V on Project_title for Test Data

X_test_title_tfidf_W2V = []

for sentence in tqdm(test_preprocessed_title):
    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
    X_test_title_tfidf_W2V.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████████████| 14000/14000
[00:00<00:00, 18269.84it/s]
```

Merging all the features

In [63]:

```
# Before we merge all the features we need to convert avg_W2V and tf-idf_avgW2V list to ndarray

# Coverting avgW2V of essays into ndarray
X_train_essay_avgW2V = np.array(X_train_essay_avgW2V)
X_cv_essay_avgW2V = np.array(X_cv_essay_avgW2V)
X_test_essay_avgW2V = np.array(X_test_essay_avgW2V)

# Coverting avgW2V of title into ndarray
X_train_title_avgW2V = np.array(X_train_title_avgW2V)
X_cv_title_avgW2V = np.array(X_cv_title_avgW2V)
X_test_title_avgW2V = np.array(X_test_title_avgW2V)

# Coverting tf-Idf_avgW2V of essays into ndarray
X_train_essay_tfidf_W2V = np.array(X_train_essay_tfidf_W2V)
X_cv_essay_tfidf_W2V = np.array(X_cv_essay_tfidf_W2V)
X_test_essay_tfidf_W2V = np.array(X_test_essay_tfidf_W2V)

# Coverting tf-Idf_avgW2V of title into ndarray
X_train_title_tfidf_W2V = np.array(X_train_title_tfidf_W2V)
X_cv_title_tfidf_W2V = np.array(X_cv_title_tfidf_W2V)
X_test_title_tfidf_W2V = np.array(X_test_title_tfidf_W2V)
```

In [64]:

```
# Merging all the features for Set-1
from scipy.sparse import hstack

X_train_s1 = hstack((X_train_essay_bow, X_train_title_bow, X_train_posted_projects_std,
X_train_price_std, X_train_clean_grade_onehot, X_train_teacher_onehot, X_train_school_state_onehot
, X_train_subcat_onehot, X_train_cat_onehot)).tocsr()
X_cv_s1 = hstack((X_cv_essay_bow, X_cv_title_bow, X_cv_posted_projects_std, X_cv_price_std, X_cv_cl
ean_grade_onehot, X_cv_teacher_onehot, X_cv_school_state_onehot, X_cv_subcat_onehot, X_cv_cat_oneho
t)).tocsr()
X_test_s1 = hstack((X_test_essay_bow, X_test_title_bow, X_test_posted_projects_std,
X_test_price_std, X_test_clean_grade_onehot, X_test_teacher_onehot, X_test_school_state_onehot,
X_test_subcat_onehot, X_test_cat_onehot)).tocsr()

print("Final Data matrix of Set-1\n")
print(X_train_s1.shape, y_train_upsampled.shape)
print(X_cv_s1.shape, y_cv.shape)
print(X_test_s1.shape, y_test.shape)
```

```
Final Data matrix of Set-1

(71262, 18314) (71262,)
(14000, 18314) (14000,)
(14000, 18314) (14000,)
```

In [65]:

```
# Merging all the features for Set-2

X_train_s2 = hstack((X_train_essay_tf, X_train_title_tf, X_train_posted_projects_std,
X_train_price_std, X_train_clean_grade_onehot, X_train_teacher_onehot, X_train_school_state_onehot
, X_train_subcat_onehot, X_train_cat_onehot)).tocsr()
X_cv_s2 = hstack((X_cv_essay_tf, X_cv_title_tf, X_cv_posted_projects_std, X_cv_price_std, X_cv_clea
n_grade_onehot, X_cv_teacher_onehot, X_cv_school_state_onehot, X_cv_subcat_onehot, X_cv_cat_onehot)
).tocsr()
X_test_s2 = hstack((X_test_essay_tf, X_test_title_tf, X_test_posted_projects_std, X_test_price_std
, X_test_clean_grade_onehot, X_test_teacher_onehot, X_test_school_state_onehot,
X_test_subcat_onehot, X_test_cat_onehot)).tocsr()

print("Final Data matrix of Set-2\n")
print(X_train_s2.shape, y_train_upsampled.shape)
print(X_cv_s2.shape, y_cv.shape)
print(X_test_s2.shape, y_test.shape)
```

```
Final Data matrix of Set-2

(71262, 18314) (71262,)
(14000, 18314) (14000,)
(14000, 18314) (14000,)
```

In [66]:

```
# Merging all the features for Set-3

X_train_s3 = hstack((X_train_essay_avgW2V, X_train_title_avgW2V, X_train_posted_projects_std,
X_train_price_std, X_train_clean_grade_onehot, X_train_teacher_onehot, X_train_school_state_onehot
, X_train_subcat_onehot, X_train_cat_onehot)).tocsr()
X_cv_s3 = hstack((X_cv_essay_avgW2V, X_cv_title_avgW2V, X_cv_posted_projects_std, X_cv_price_std, X
_cv_clean_grade_onehot, X_cv_teacher_onehot, X_cv_school_state_onehot, X_cv_subcat_onehot,
X_cv_cat_onehot)).tocsr()
X_test_s3 = hstack((X_test_essay_avgW2V, X_test_title_avgW2V, X_test_posted_projects_std,
X_test_price_std, X_test_clean_grade_onehot, X_test_teacher_onehot, X_test_school_state_onehot,
X_test_subcat_onehot, X_test_cat_onehot)).tocsr()

print("Final Data matrix of Set-3\n")
print(X_train_s3.shape, y_train_upsampled.shape)
print(X_cv_s3.shape, y_cv.shape)
print(X_test_s3.shape, y_test.shape)
```

```
Final Data matrix of Set-3

(71262, 701) (71262,)
(14000, 701) (14000,)
(14000, 701) (14000,)
```

In [67]:

```
# Merging all the features for Set-4

X_train_s4 = hstack((X_train_essay_tfidf_W2V, X_train_title_tfidf_W2V, X_train_posted_projects_std
, X_train_price_std, X_train_clean_grade_onehot, X_train_teacher_onehot,
X_train_school_state_onehot, X_train_subcat_onehot, X_train_cat_onehot)).tocsr()
X_cv_s4 = hstack((X_cv_essay_tfidf_W2V, X_cv_title_tfidf_W2V, X_cv_posted_projects_std,
X_cv_price_std, X_cv_clean_grade_onehot, X_cv_teacher_onehot, X_cv_school_state_onehot, X_cv_subcat
_onehot, X_cv_cat_onehot)).tocsr()
X_test_s4 = hstack((X_test_essay_tfidf_W2V, X_test_title_tfidf_W2V, X_test_posted_projects_std, X_
test_price_std, X_test_clean_grade_onehot, X_test_teacher_onehot, X_test_school_state_onehot,
X_test_subcat_onehot, X_test_cat_onehot)).tocsr()
```

```
print("Final Data matrix of Set-4\n")
print(X_train_s4.shape, y_train_upsampled.shape)
print(X_cv_s4.shape, y_cv.shape)
print(X_test_s4.shape, y_test.shape)
```

```
Final Data matrix of Set-4

(71262, 701) (71262,)
(14000, 701) (14000,)
(14000, 701) (14000,)
```

Applying KNN on these Sets

Applying KNN Brute Force on Set-1 with BOW

In [68]:

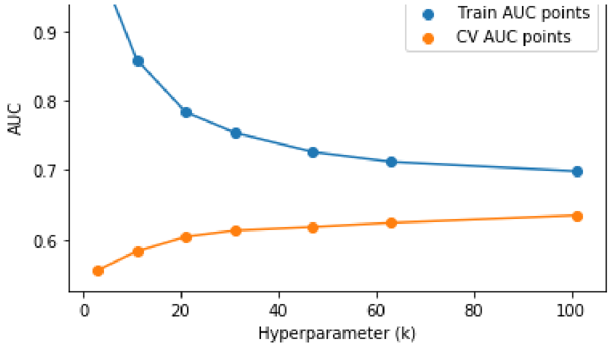
```
# My system can't process all the data-points at once so using Batch_prediction and dividing my wh
ole data into batches of
# size 1000.

def batch_predict(clf, data):
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [69]:

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

train_auc = []
cv_auc = []
K = [3, 11, 21, 31, 47, 63, 101]

for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors = i, algorithm = 'brute')
    neigh.fit(X_train_s1, y_train_upsampled)

    y_train_pred = batch_predict(neigh, X_train_s1)
    y_cv_pred = batch_predict(neigh, X_cv_s1)

    train_auc.append(roc_auc_score(y_train_upsampled, 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("Hyperparameter (k)")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
100%|████████████████████████████████████████████████████████████████████████████████| 7/7
[1:22:19<00:00, 705.66s/it]
```

ERROR PLOTS
1.0
Train AUC
CV AUC



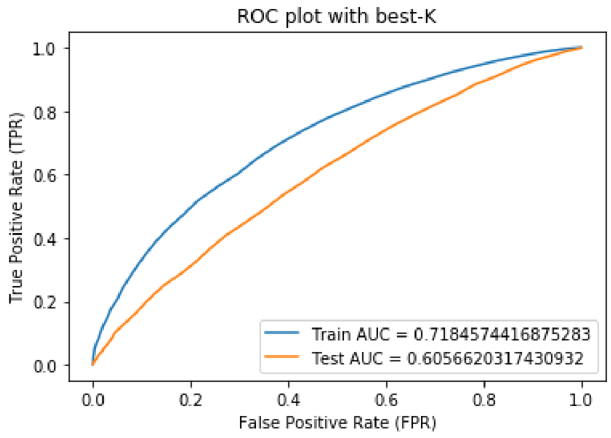
In [70]:

```
Aoc_score_cv = [x for x in cv_auc]
best_k_cv = K[Aoc_score_cv.index(max(Aoc_score_cv))]
print("Maximum AUC score of cv is : " +  str(max(Aoc_score_cv)))
print("Corresponding best k value of cv is : ", best_k_cv)
```

```
Maximum AUC score of cv is : 0.6347187142063904
Corresponding best k value of cv is :  101
```

In [71]:

```
neigh = KNeighborsClassifier(n_neighbors = 55, algorithm='brute')
neigh.fit(X_train_s1, y_train_upsampled)

y_train_pred = batch_predict(neigh, X_train_s1)
y_test_pred = batch_predict(neigh, X_test_s1)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train_upsampled, 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.ylabel("True Positive Rate (TPR)")
plt.xlabel("False Positive Rate (FPR)")
plt.title("ROC plot with best-K")
plt.show()
```



Observations

- From Error plot we can see that the maximum AUC score for cv is 0.634
- As we increase the value of k the AUC score increases and hence plot gets better
- In ROC plot with best K we get Test AUC = 0.6056

In [72]:

```
# Finding best threshould which will give us the minimal value of False Positive Rate(FPR)
```

```
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    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 [73]:

```
# Finding Best Threshould Value

best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
```

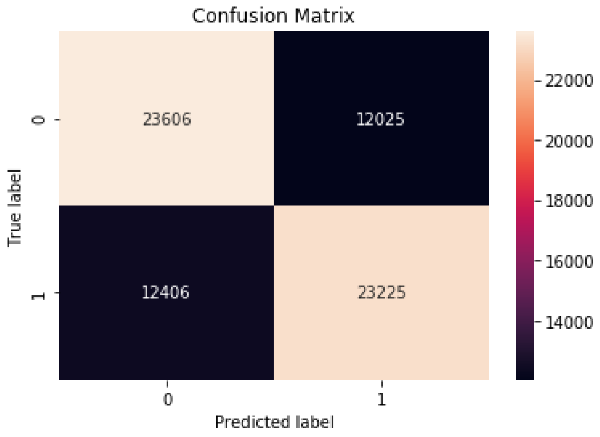
```
the maximum value of tpr*(1-fpr) 0.43183924017766856 for threshold 0.418
```

In [74]:

```
# Plotiing Plot for Confusion Matrix for training data
ax = plt.subplot()

sns.heatmap(confusion_matrix(y_train_upsampled, predict_with_best_t(y_train_pred, best_t)), annot=True, ax = ax, fmt='g');
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.set_title('Confusion Matrix')
```

Out[74]:

```
Text(0.5,1,'Confusion Matrix')
```

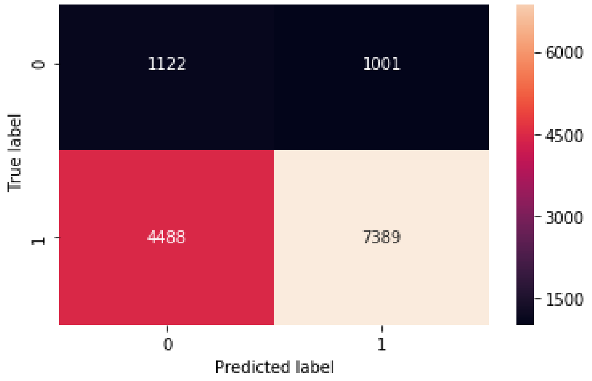


In [75]:

```
# Plotiing Plot for Confusion Matrix for Test Data
ax = plt.subplot()

sns.heatmap(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)), annot=True, ax = ax
, fmt='g');
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.set_title('Confusion Matrix')
```

Out[75]:

```
Text(0.5,1,'Confusion Matrix')
```



Observations

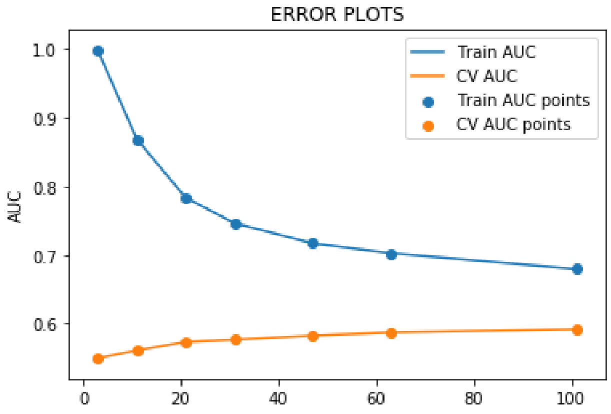
- As seen in the Plot of confusion matrix the value of True_Positive (TP) and False_Negative (FN) is high.
- Which means that along with predicting True points as true, we are also predicting false for the true points.
- Our Test and CV data is highly imbalanced so this might be the reason for getting high FN value.
- Because of this imbalance data we're not getting high TN value.

Applying KNN brute force on TFIDF, SET 2

In [76]:

```
train_auc = []
cv_auc = []
K = [3, 11, 21, 31, 47, 63, 101]

for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors = i, algorithm = 'brute')
    neigh.fit(X_train_s2, y_train_upsampled)

    y_train_pred = batch_predict(neigh, X_train_s2)
    y_cv_pred = batch_predict(neigh, X_cv_s2)

    train_auc.append(roc_auc_score(y_train_upsampled, 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("Hyperparameter (k)")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
100%|██████████████████████████████████████████████████████████████████| 7/7
[2:06:46<00:00, 1086.64s/it]
```



Hyperparameter (k)

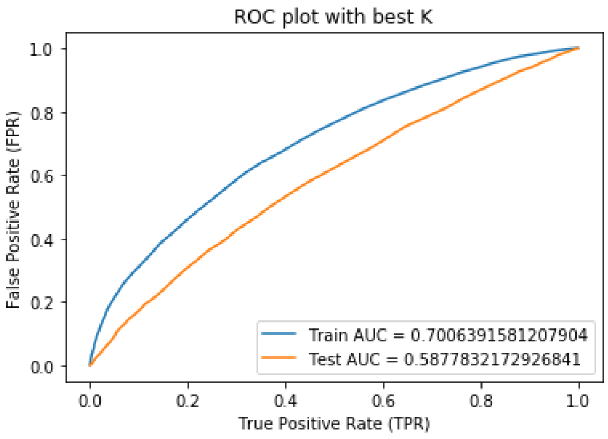
In [80]:

```
Aoc_score_cv = [x for x in cv_auc]
best_k_cv = K[Aoc_score_cv.index(max(Aoc_score_cv))]
print("Maximum AUC score of cv is : " +  str(max(Aoc_score_cv)))
print("Corresponding best k value of cv is : ", best_k_cv)
```

```
Maximum AUC score of cv is : 0.6026812907351381
Corresponding best k value of cv is :  101
```

In [77]:

```
neigh = KNeighborsClassifier(n_neighbors = 65, algorithm='brute')
neigh.fit(X_train_s2, y_train_upsampled)

y_train_pred = batch_predict(neigh, X_train_s2)
y_test_pred = batch_predict(neigh, X_test_s2)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train_upsampled, 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("True Positive Rate (TPR)")
plt.ylabel("False Positive Rate (FPR)")
plt.title("ROC plot with best K")
plt.show()
```



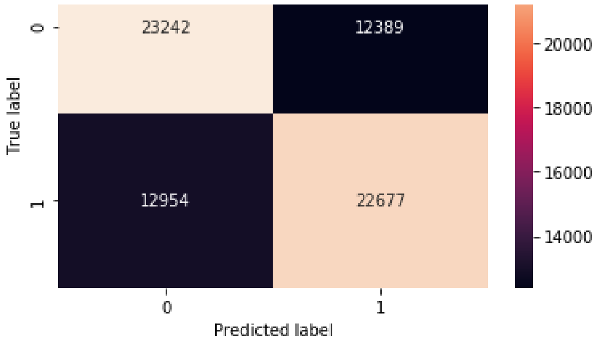
In [78]:

```
# Printing Confusion Matrix for Train Data

best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
ax = plt.subplot()

sns.heatmap(confusion_matrix(y_train_upsampled, predict_with_best_t(y_train_pred, best_t)), annot=True, ax = ax, fmt='g')
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.set_title('Confusion Matrix')
```

```
the maximum value of tpr*(1-fpr) 0.4151481190146198 for threshold 0.492
```

Out[78]:

```
Text(0.5,1,'Confusion Matrix')
```

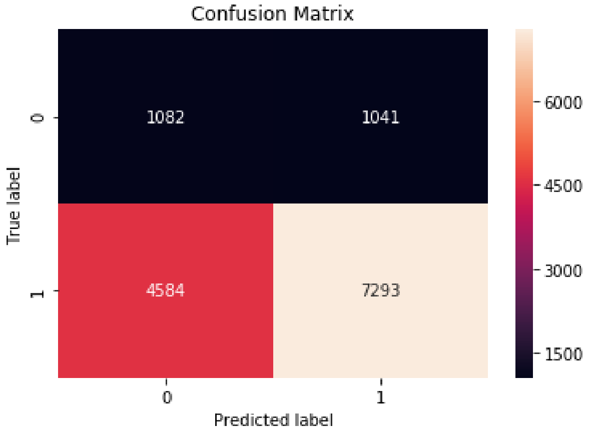


In [79]:

```
# Printing Confusion Matrix for Test Data

ax = plt.subplot()

sns.heatmap(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)), annot=True, ax = ax
, fmt='g')
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.set_title('Confusion Matrix')
```

Out[79]:

```
Text(0.5,1,'Confusion Matrix')
```



Observations

- Maximum Auc score on CV is 0.602
- After plotting Roc using the best k we got the AUC score on test is 0.587.
- Now coming to the Confusion Matrix, values of TP and FN is large as expected.
- In our Cross Validation data positive class is dominating so because of that we are might getting such values.

Applying KNN brute force on AVG W2V, SET 3

In [80]:

```
# Reducing number of datapoints because I was getting memory issues

X_train_s3 = X_train_s3[29630:41630]
X_cv_s3 = X_train_s3[:6050]
X_test_s3 = X_test_s3[:8100]
```

In [81]:

```
y_train_s3 = y_train_upsampled[29630:41630]
y_cv_s3 = y_cv[:6050]
y_test_s3 = y_test[:8100]
```

```
y_test_s3 = y_test[:8100]
```

In [82]:

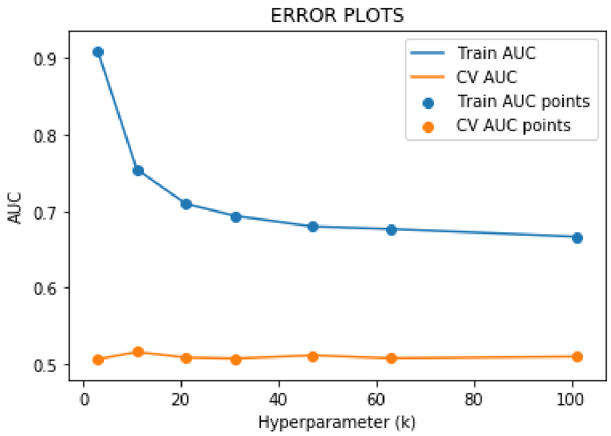
```
# Printing shape of Train, CV and test data after reducing the no. of datapoints

print(X_train_s3.shape, y_train_s3.shape)
print(X_cv_s3.shape, y_cv_s3.shape)
print(X_test_s3.shape, y_test_s3.shape)
```

```
(12000, 701) (12000,)
(6050, 701) (6050,)
(8100, 701) (8100,)
```

In [83]:

```
train_auc = []
cv_auc = []
K = [3, 11, 21, 31, 47, 63, 101]

for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors = i, algorithm = 'brute')
    neigh.fit(X_train_s3, y_train_s3)

    y_train_pred = batch_predict(neigh, X_train_s3)
    y_cv_pred = batch_predict(neigh, X_cv_s3)

    train_auc.append(roc_auc_score(y_train_s3, y_train_pred))
    cv_auc.append(roc_auc_score(y_cv_s3, 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("Hyperparameter (k)")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
100%|████████████████████████████████████████████████████████████████| 7/7 [51:
21<00:00, 440.15s/it]
```



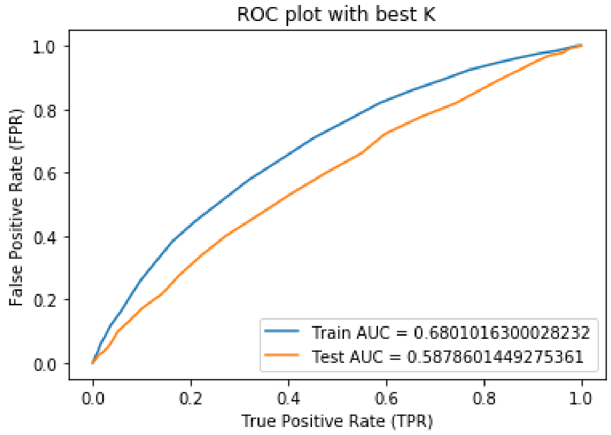
In [84]:

```
Aoc_score_cv = [x for x in cv_auc]
best_k_cv = K[Aoc_score_cv.index(max(Aoc_score_cv))]
print("Maximum AUC score of cv is : " +  str(max(Aoc_score_cv)))
print("Corresponding best k value of cv is : ", best_k_cv)
```

```
Maximum AUC score of cv is : 0.5159888289937727
Corresponding best k value of cv is :  11
```

In [85]:

```
# Plotting ROC with the best k that we've got on Cv data

neigh = KNeighborsClassifier(n_neighbors = 45, algorithm='brute')
neigh.fit(X_train_s3, y_train_s3)

y_train_pred = batch_predict(neigh, X_train_s3)
y_test_pred = batch_predict(neigh, X_test_s3)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train_s3, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test_s3, 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("True Positive Rate (TPR)")
plt.ylabel("False Positive Rate (FPR)")
plt.title("ROC plot with best K")
plt.show()
```



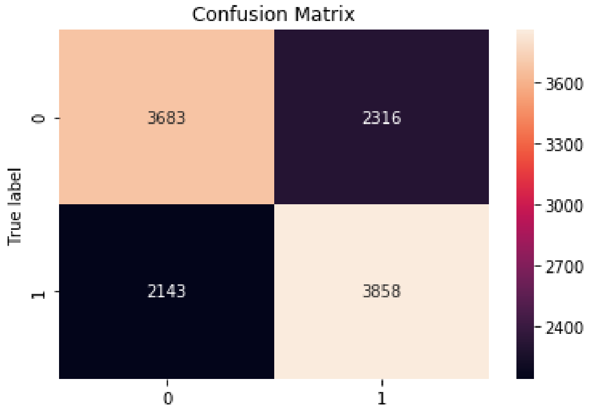
In [86]:

```
# Plotting Confusion Matrix of Train data of Set-3

best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
ax = plt.subplot()

sns.heatmap(confusion_matrix(y_train_s3, predict_with_best_t(y_train_pred, best_t)), annot=True, ax
= ax, fmt='g');
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.set_title('Confusion Matrix')
```

```
the maximum value of tpr*(1-fpr) 0.394694844297079 for threshold 0.511
```

Out[86]:

```
Text(0.5,1,'Confusion Matrix')
```



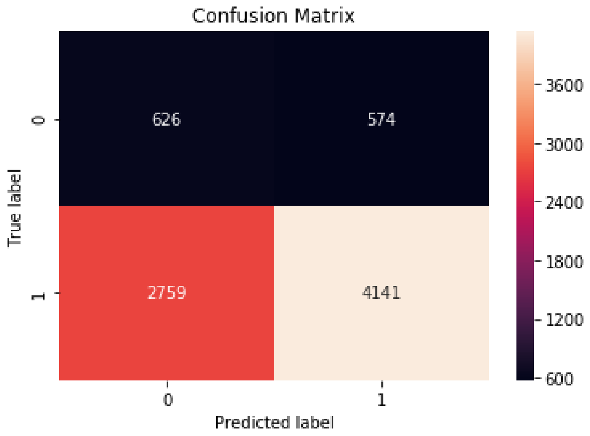
Predicted label

In [87]:

```
ax = plt.subplot()

sns.heatmap(confusion_matrix(y_test_s3, predict_with_best_t(y_test_pred, best_t)), annot=True, ax =
ax, fmt='g');
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.set_title('Confusion Matrix')
```

Out[87]:

```
Text(0.5,1,'Confusion Matrix')
```



Observations

- Maximum Auc score we got on CV is 0.515
- On Testing data I'm getting AUC score of 0.587
- On AvgW2V also we're getting high values of FN, TP just because of the imbalancing in CV data.

Applying KNN brute force on TFIDF W2V, SET 4

In [88]:

```
X_train_s4 = X_train_s4[29630:41630]
X_cv_s4 = X_train_s4[:6050]
X_test_s4 = X_test_s4[:8100]
```

In [89]:

```
y_train_s4 = y_train_upsampled[29630:41630]
y_cv_s4 = y_cv[:6050]
y_test_s4 = y_test[:8100]
```

In [90]:

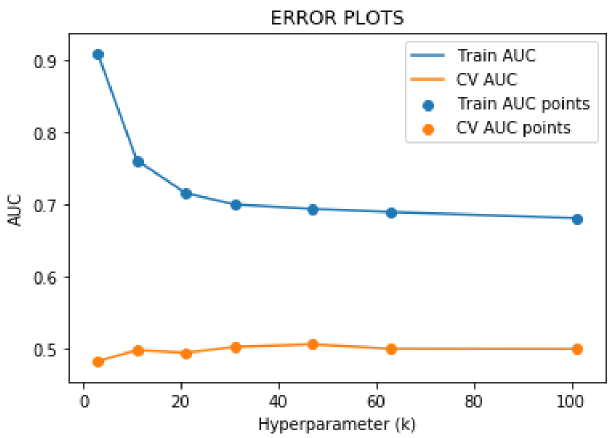
```
print(X_train_s4.shape, y_train_s4.shape)
print(X_cv_s4.shape, y_cv_s4.shape)
print(X_test_s4.shape, y_test_s4.shape)
```

```
(12000, 701) (12000,)
(6050, 701) (6050,)
(8100, 701) (8100,)
```

In [91]:

```
train_auc = []
train_auc = []
cv_auc = []
K = [3, 11, 21, 31, 47, 63, 101]

for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors = i, algorithm = 'brute')
    neigh.fit(X_train_s4, y_train_s4)

    y_train_pred = batch_predict(neigh, X_train_s4)
    y_cv_pred = batch_predict(neigh, X_cv_s4)

    train_auc.append(roc_auc_score(y_train_s4, y_train_pred))
    cv_auc.append(roc_auc_score(y_cv_s4, 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("Hyperparameter (k)")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
100%|████████████████████████████████████████████████████████████████| 7/7 [30:
40<00:00, 262.94s/it]
```



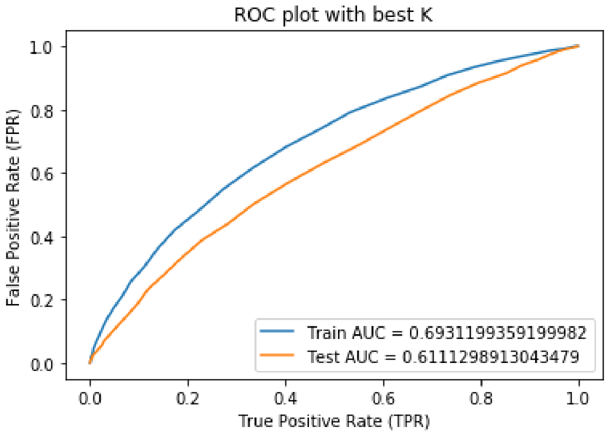
In [92]:

```
Aoc_score_cv = [x for x in cv_auc]
best_k_cv = K[Aoc_score_cv.index(max(Aoc_score_cv))]
print("Maximum AUC score of cv is : " +  str(max(Aoc_score_cv)))
print("Corresponding best k value of cv is : ", best_k_cv)
```

```
Maximum AUC score of cv is : 0.5051183682849902
Corresponding best k value of cv is :  47
```

In [93]:

```
neigh = KNeighborsClassifier(n_neighbors = 47, algorithm='brute')
neigh.fit(X_train_s4, y_train_s4)

y_train_pred = batch_predict(neigh, X_train_s4)
y_test_pred = batch_predict(neigh, X_test_s4)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train_s4, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test_s4, 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("True Positive Rate (TPR)")
plt.ylabel("False Positive Rate (FPR)")
plt.title("ROC plot with best K")
plt.show()
plt.show()
```



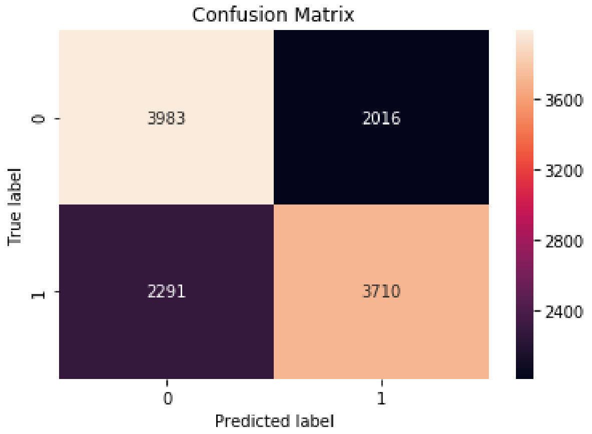
In [94]:

```
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
ax = plt.subplot()

sns.heatmap(confusion_matrix(y_train_s4, predict_with_best_t(y_train_pred, best_t)), annot=True, ax
= ax, fmt='g');
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.set_title('Confusion Matrix')
```

```
the maximum value of tpr*(1-fpr) 0.41047028917973016 for threshold 0.511
```

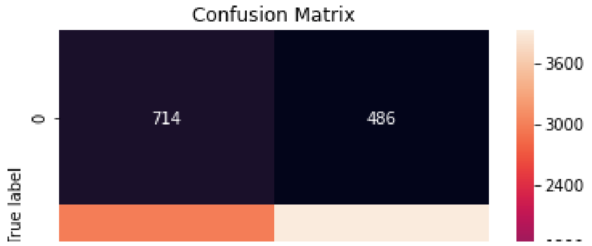
Out[94]:

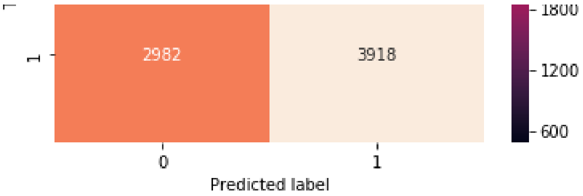
```
Text(0.5,1,'Confusion Matrix')
```



In [95]:

```
ax = plt.subplot()

sns.heatmap(confusion_matrix(y_test_s4, predict_with_best_t(y_test_pred, best_t)), annot=True, ax =
ax, fmt='g');
ax.set_xlabel('Predicted label');
ax.set_ylabel('True label');
ax.set_title('Confusion Matrix');
```





Observations

- On CV data I'm getting AUC score = 0.505
- Using best k we get Auc score of 0.611 on the test data
- By looking at confusion matrix, here also we're getting high values of TP and FN.

Feature Selection using 'SelectKBest'

In [96]:

```
print(X_train_s2.shape, y_train_upsampled.shape)
print(X_cv_s2.shape, y_cv.shape)
print(X_test_s2.shape, y_test.shape)
```

```
(71262, 18314) (71262,)
(14000, 18314) (14000,)
(14000, 18314) (14000,)
```

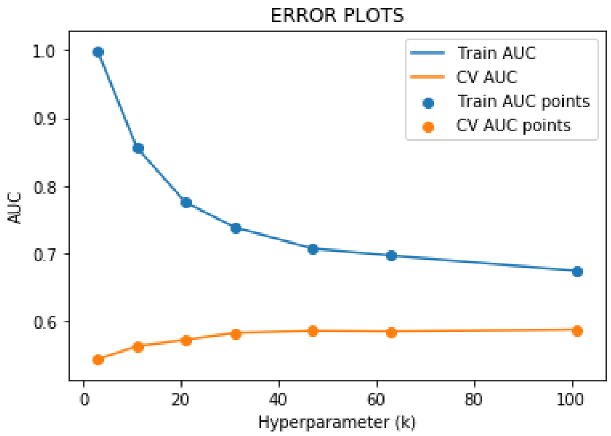
In [98]:

```
# Selecting Top 2000 features

from sklearn.datasets import load_digits
from sklearn.feature_selection import SelectKBest, f_classif

top_2000 = SelectKBest(f_classif, k=2000)
top_2000.fit(X_train_s2, y_train_upsampled)

X_train_s2_new = top_2000.transform(X_train_s2)
X_cv_s2_new = top_2000.transform(X_cv_s2)
X_test_s2_new = top_2000.transform(X_test_s2)
```

In [101]:

```
train_auc = []
cv_auc = []
K = [3, 11, 21, 31, 47, 63, 101]

for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors = i, algorithm = 'brute')
    neigh.fit(X_train_s2_new, y_train_upsampled)

    y_train_pred = batch_predict(neigh, X_train_s2_new)
    y_cv_pred = batch_predict(neigh, X_cv_s2_new)

    train_auc.append(roc_auc_score(y_train_upsampled, 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("Hyperparameter (k)")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
  0%|                                                                                      |
[00:00<?, ?it/s]
 14%|████████████                                                                          | 1/7
[07:09<42:55, 429.24s/it]
 29%|████████████████████████                                                              | 2/7 [14:5
36:47, 441.57s/it]
 43%|████████████████████████████████████                                                  | 3/7
[22:55<30:07, 451.91s/it]
 57%|████████████████████████████████████████████████                                      | 4/7 [30:4
<22:54, 458.22s/it]
 71%|████████████████████████████████████████████████████████████                          | 5/7
[38:41<15:25, 462.51s/it]
 86%|████████████████████████████████████████████████████████████████████████              | 6/7 [46:3
3<07:45, 465.44s/it]
100%|████████████████████████████████████████████████████████████████████████████████████| 7/7 [54:
25<00:00, 466.50s/it]
```



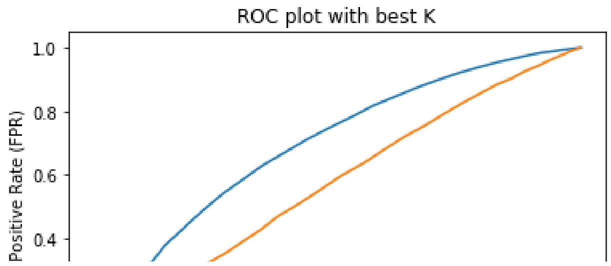
In [102]:

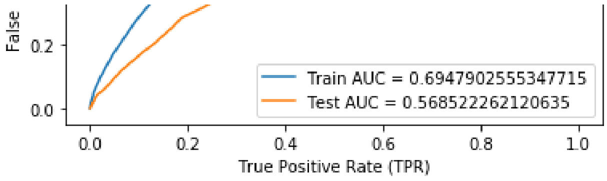
```
Aoc_score_cv = [x for x in cv_auc]
best_k_cv = K[Aoc_score_cv.index(max(Aoc_score_cv))]
print("Maximum AUC score of cv is : " +  str(max(Aoc_score_cv)))
print("Corresponding best k value of cv is : ", best_k_cv)
```

```
Maximum AUC score of cv is : 0.5871409375840155
Corresponding best k value of cv is :  101
```

In [103]:

```
neigh = KNeighborsClassifier(n_neighbors = 65, algorithm='brute')
neigh.fit(X_train_s2_new, y_train_upsampled)

y_train_pred = batch_predict(neigh, X_train_s2_new)
y_test_pred = batch_predict(neigh, X_test_s2_new)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train_upsampled, 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("True Positive Rate (TPR)")
plt.ylabel("False Positive Rate (FPR)")
plt.title("ROC plot with best K")
plt.show()
```





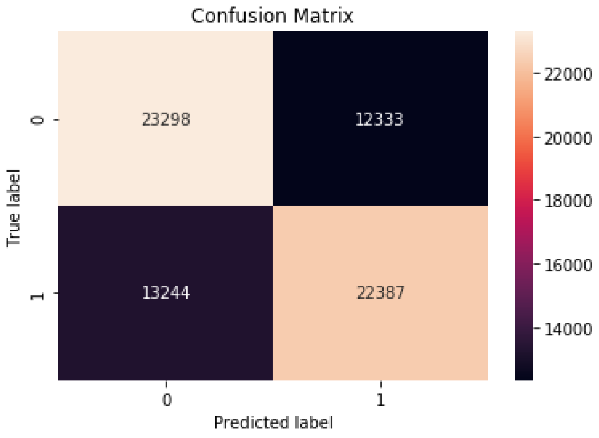
In [104]:

```
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
ax = plt.subplot()

sns.heatmap(confusion_matrix(y_train_upsampled, predict_with_best_t(y_train_pred, best_t)), annot=T
rue, ax = ax, fmt='g')
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label');
ax.set_title('Confusion Matrix')
```

```
the maximum value of tpr*(1-fpr) 0.4108265645140104 for threshold 0.431
```

Out[104]:

```
Text(0.5,1,'Confusion Matrix')
```

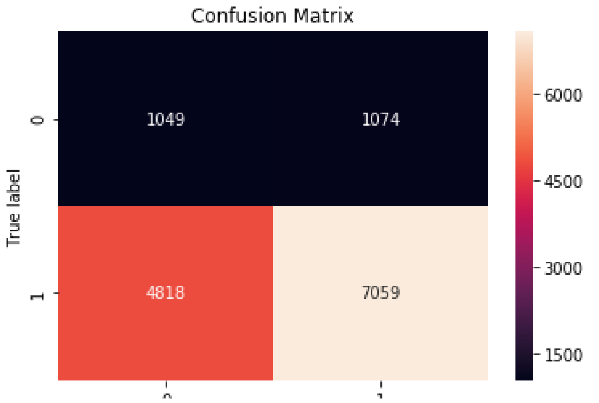


In [105]:

```
# Confusion matrix for test data

ax = plt.subplot()

sns.heatmap(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)), annot=True, ax = ax
, fmt='g')
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label');
ax.set_title('Confusion Matrix')
```

Out[105]:

```
Text(0.5,1,'Confusion Matrix')
```



Observations

- We're getting AUC score of 0.587 on CV data
- On Test data I'm getting AUC score of 0.568 which is low as compared to the AUC scores we had only other techniques
- We can say that because of the value of k we're overfitting on train data as result we're getting Auc score of ~0.84.

In [106]:

```
from prettytable import PrettyTable

t = PrettyTable()
t.field_names= ("Vectorizer", "Model","HyperParameter" ,"AUC")
t.add_row(["BOW", "Brute", 55, 0.605])
t.add_row(["Tf-Idf", "Brute", 65, 0.587])
t.add_row(["AvgW2V", "Brute", 45, 0.587])
t.add_row(["Tf-Idf_W2V", "Brute", 47, 0.611])
t.add_row(["Tf-Idf_Best_K", "Brute", 65, 0.568])
```

In [107]:

```
print(t)
```

```
+---------------+-------+----------------+-------+
|   Vectorizer  | Model | HyperParameter |  AUC  |
+---------------+-------+----------------+-------+
|      BOW      | Brute |       55       | 0.605 |
|     Tf-Idf    | Brute |       65       | 0.587 |
|     AvgW2V    | Brute |       45       | 0.587 |
|   Tf-Idf_W2V  | Brute |       47       | 0.611 |
| Tf-Idf_Best_K | Brute |       65       | 0.568 |
+---------------+-------+----------------+-------+
```

Conclusion

- Donors Choose Dataset is highly imbalance Dataset with Positive class as it's majority class.
- There is Time & Date associated with every project submitted, so this Date feature might be benefit for us since we can take it as a time-series data and apply relevant techniques.
- After preprocessing our data we have applied 4 different techniques to convert our text data into vectors.
- As seen from the above table almost all this techniques gave us kind of similar results but Tf-Idf_W2V has given Slightly better result than any other techniques.
- Since we're upsampling Train data but In CV we're still having the Highly imbalance data which causes the selection of wrong K which results in comparatively less AUC score.
- Because of Imbalancing KNN might not be the best solution.
- And the Run time of KNN is also very high.