In [ ]:

```
#Founded in 2000 by a high school teacher in the Bronx, DonorsChoose.org empowers public school teachers from ac

#DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need c

#Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, there are three

#How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as qu

#How to increase the consistency of project vetting across different volunteers to improve the experience for te

#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 tec

#With an algorithm to pre-screen applications, DonorsChoose.org can auto-approve some applications quickly so th

#Your machine learning algorithm can help more teachers get funded more quickly, and with less cost to DonorsChc

#Data can be downloaded from here - https://www.kaggle.com/c/donorschoose-application-screening/data
```

In [1]:

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

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

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

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

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

from tqdm import tqdm
import os

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

1.1 Reading Data

In [2]:

```
dft = pd.read_csv('train_data.csv', nrows=80000)
dfr= pd.read_csv('resources.csv')
```

In [3]:

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

```
Number of data points in train data (80000, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]:

```
print(dfr.shape)
print(dfr.columns.values)
```

```
(1541272, 4)
['id' 'description' 'quantity' 'price']
```

In [5]:

```
# 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(dft.columns)]


#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
dft['Date'] = pd.to_datetime(dft['project_submitted_datetime'])
dft.drop('project_submitted_datetime', axis=1, inplace=True)# we drop the col
dft.sort_values(by=['Date'], inplace=True)# sort the values y date


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

1.3 Text preprocessing

In [6]:
```
# merge two column text dataframe:
dft["essay"] = dft["project_essay_1"].map(str) +\
                        dft["project_essay_2"].map(str) + \
                        dft["project_essay_3"].map(str) + \
                       dft["project_essay_4"].map(str)
```

In [7]:
```
dft.head(2)
```

Out[7]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_sub
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 [8]:

```
# 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 [9]:

```
# 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', 'af
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'f
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few',
            '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',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'had
            "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',
            'won', "won't", 'wouldn', "wouldn't"]
```

Preprocessing of project_subject_categories

In [10]:

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

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

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

from collections import Counter
my_counter = Counter()
for word in dft['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 of project_subject_subcategories

In [11]:

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

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

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

dft['clean_subcategories'] = sub_cat_list
dft.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 dft['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 [12]:
```
print(dft['project_grade_category'][:20])#   we have to remove the grades from every row
```

```
55660    Grades PreK-2
76127       Grades 3-5
51140    Grades PreK-2
473      Grades PreK-2
41558       Grades 3-5
29891       Grades 3-5
79026       Grades 3-5
23374    Grades PreK-2
49228    Grades PreK-2
72638      Grades 9-12
7176     Grades PreK-2
70898       Grades 3-5
72593    Grades PreK-2
35006       Grades 3-5
5145        Grades 3-5
48237      Grades 9-12
64637    Grades PreK-2
52282      Grades 9-12
46375       Grades 3-5
36468    Grades PreK-2
Name: project_grade_category, dtype: object
```

In [13]:

```
d= list(dft['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 d:
    # consider we have text like this:
    for j in i.split(' '): #    # split by spae
        j=j.replace('Grades','')# clean grades from the row
    grade_cat_list.append(j.strip())



dft['clean_grade'] = grade_cat_list
dft.drop(['project_grade_category'], axis=1, inplace=True)

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

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

Preparing our data for the models and splitting data into train and cv(or test)

In [14]:

```
#Splitting Data into train and Test sklearn https://scikit-learn.org/stable/modules/generated/sklearn.model_sele
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(dft,
                                                    dft['project_is_approved'],
                                                    stratify= dft['project_is_approved'],
                                                    test_size = 0.33
                                                   )
```

In [15]:

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

In [16]:

```
print(y_train.value_counts())
print(y_test.value_counts())
print(y_cv.value_counts())
# huge imbalance
```

```
1    30469
0     5443
Name: project_is_approved, dtype: int64
1    22398
0     4002
Name: project_is_approved, dtype: int64
1    15007
0     2681
Name: project_is_approved, dtype: int64
```

In [17]:

```
#droping the y labels
#https://stackoverflow.com/questions/13411544/delete-column-from-pandas-dataframe-by-column-name
#x_train =
X_train.drop(["project_is_approved"], axis = 1, inplace = True)
#x_test =
X_test.drop(["project_is_approved"], axis = 1, inplace = True)
#x_cv =
X_cv.drop(["project_is_approved"], axis = 1, inplace = True)
```

Preprocess train,test and cv data

In [18]:
```
# Preprocessing Train Data of Project Essays

from tqdm import tqdm
train_preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(X_train['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%|██████████████████████████████████████████████████████████████████████████| 35912/35912 [00:52<00:00, 68
5.91it/s]
```

In [19]:
```
#Preprocessing Test Data of Project Essays

# Combining all the above students
from tqdm import tqdm
test_preprocessed_essays = []
# tqdm is for printing the status bar
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)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    test_preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████| 26400/26400 [00:43<00:00, 60
2.92it/s]
```

In [20]:
```
#Preprocessing Cross Validation Data  of Project Essays

# Combining all the above students
from tqdm import tqdm
cv_preprocessed_essays = []
# tqdm is for printing the status bar
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)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    cv_preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████| 17688/17688 [00:32<00:00, 54
0.82it/s]
```

In [21]:
```
#Preprocessing Train Data for Project Titles
from tqdm import tqdm
train_preprocessed_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(X_train['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    train_preprocessed_titles.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████| 35912/35912 [00:02<00:00, 1254
8.81it/s]
```

In [22]:

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

```
100%|██████████████████████████████████████████████████████████| 26400/26400 [00:02<00:00, 1220
3.53it/s]
```

In [23]:

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

```
100%|██████████████████████████████████████████████████████████| 17688/17688 [00:01<00:00, 1282
4.52it/s]
```

In [24]:

```
cv_preprocessed_titles[1]
```

Out[24]:

```
'class needs flexible seating'
```

Encoding

vectorize categorical data

In [25]:

```
#project*_subject_categories convert categorical to vectors*
# convert train,cv and test data of clean_categories into vectors


# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True)
vectorizer.fit(X_train['clean_categories'].values)

# firstly convert fit the train data into the vectoriaer then it learn the vocablery

# we use the fitted CountVectorizer to convert the text to vector
X_train_cat = vectorizer.transform(X_train['clean_categories'].values)
X_cv_cat = vectorizer.transform(X_cv['clean_categories'].values)
X_test_cat = 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 [26]:

```
print("After vectorizations")
print(X_train_cat.shape, y_train.shape)
print(X_cv_cat.shape, y_cv.shape)
print(X_test_cat.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(35912, 9) (35912,)
(17688, 9) (17688,)
(26400, 9) (26400,)
====================================================================================================
```

In [27]:

```
# convert train,cv and test data of clean_categories into vectors
#project*_subject_subcategories convert categorical to vectors

# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=True)
vectorizer.fit(X_train['clean_subcategories'].values)

# firstly convert fit the train data into the vectoriaer then it learn the vocablery

# we use the fitted CountVectorizer to convert the text to vector
X_train_subcat = vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_subcat = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_subcat = vectorizer.transform(X_test['clean_subcategories'].values)

print(vectorizer.get_feature_names())
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Civics_Government', 'Extracurricu
lar', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingArts',
'CharacterEducation', 'TeamSports', 'Other', 'College_CareerPrep', 'History_Geography', 'Music', 'Health_LifeS
cience', 'EarlyDevelopment', 'ESL', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'A
ppliedSciences', 'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
```

In [28]:

```
print("After vectorizations")
print(X_train_subcat.shape, y_train.shape)
print(X_cv_subcat.shape, y_cv.shape)
print(X_test_subcat.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(35912, 30) (35912,)
(17688, 30) (17688,)
(26400, 30) (26400,)
====================================================================================================
```

In [29]:

```
#school_state convert categorical to vectors
# now time to cont the each words
from collections import Counter
my_counter = Counter()
for word in dft['school_state'].values:
    my_counter.update(word.split())# count the words

school_state_dict = dict(my_counter)# store in dicionary
sorted_school_state_dict = dict(sorted(school_state_dict.items(), key=lambda kv: kv[1]))# sort it
print(sorted_school_state_dict)
```

```
{'VT': 58, 'WY': 79, 'ND': 106, 'MT': 168, 'RI': 206, 'SD': 221, 'NE': 236, 'NH': 237, 'DE': 250, 'AK': 256,
'WV': 354, 'ME': 369, 'HI': 369, 'DC': 382, 'NM': 398, 'KS': 460, 'IA': 486, 'ID': 501, 'AR': 734, 'CO': 858,
'MN': 870, 'OR': 904, 'KY': 955, 'MS': 955, 'NV': 1016, 'MD': 1087, 'TN': 1202, 'CT': 1235, 'UT': 1270, 'AL':
1273, 'WI': 1331, 'VA': 1513, 'AZ': 1561, 'NJ': 1625, 'OK': 1710, 'WA': 1715, 'LA': 1764, 'MA': 1765, 'OH': 18
19, 'MO': 1896, 'IN': 1897, 'PA': 2237, 'MI': 2341, 'SC': 2881, 'GA': 2908, 'IL': 3178, 'NC': 3737, 'FL': 456
8, 'NY': 5391, 'TX': 5406, 'CA': 11262}
```

In [30]:

```
# convert train,cv and test data of clean_categories into vectors


# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_school_state_dict.keys()), lowercase=False, binary=True)
vectorizer.fit(dft['school_state'].values)

# firstly convert fit the train data into the vectoriaer then it learn the vocablery

# we use the fitted CountVectorizer to convert the text to vector
X_train_school_state = vectorizer.transform(X_train['school_state'].values)
X_cv_school_state = vectorizer.transform(X_cv['school_state'].values)
X_test_school_state = vectorizer.transform(X_test['school_state'].values)

print(vectorizer.get_feature_names())
```

```
['VT', 'WY', 'ND', 'MT', 'RI', 'SD', 'NE', 'NH', 'DE', 'AK', 'WV', 'ME', 'HI', 'DC', 'NM', 'KS', 'IA', 'ID',
'AR', 'CO', 'MN', 'OR', 'KY', 'MS', 'NV', 'MD', 'TN', 'CT', 'UT', 'AL', 'WI', 'VA', 'AZ', 'NJ', 'OK', 'WA', 'L
A', 'MA', 'OH', 'MO', 'IN', 'PA', 'MI', 'SC', 'GA', 'IL', 'NC', 'FL', 'NY', 'TX', 'CA']
```

In [31]:

```
print("After vectorizations")
print(X_train_school_state .shape, y_train.shape)
print(X_cv_school_state .shape, y_cv.shape)
print(X_test_school_state .shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(35912, 51) (35912,)
(17688, 51) (17688,)
(26400, 51) (26400,)
====================================================================================================
```

In [32]:

```
# convert train,cv and test data of clean_categories into vectors
#project_grade_category *categorical** to vectors

# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_project_grade_category_dict.keys()), lowercase=False, binary
vectorizer.fit(dft['clean_grade'].values)

# firstly convert fit the train data into the vectoriaer then it learn the vocablery

# we use the fitted CountVectorizer to convert the text to vector
X_train_project_grade_category = vectorizer.transform(X_train['clean_grade'].values)
X_cv_project_grade_category = vectorizer.transform(X_cv['clean_grade'].values)
X_test_project_grade_category = vectorizer.transform(X_test['clean_grade'].values)

print(vectorizer.get_feature_names())
```

```
['9-12', '6-8', '3-5', 'PreK-2']
```

In [33]:

```
print("After vectorizations")
print(X_train_project_grade_category  .shape, y_train.shape)
print(X_cv_project_grade_category  .shape, y_cv.shape)
print(X_test_project_grade_category  .shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(35912, 4) (35912,)
(17688, 4) (17688,)
(26400, 4) (26400,)
====================================================================================================
```

In [34]:

```
#https://stackoverflow.com/questions/42224700/attributeerror-float-object-has-no-attribute-split
#teacher_prefix categorical to vectors
dft['teacher_prefix']=dft['teacher_prefix'].fillna(" ")# fill the null values with space


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


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

In [35]:
```
# convert train,cv and test data of clean_categories into vectors


# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_teacher_prefix_dict.keys()), lowercase=False, binary=True)
vectorizer.fit(dft['teacher_prefix'].values.astype('U'))

# firstly convert fit the train data into the vectoriaer then it learn the vocablery

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_prefix = vectorizer.transform(X_train['teacher_prefix'].values.astype('U'))
X_cv_teacher_prefix= vectorizer.transform(X_cv['teacher_prefix'].values.astype('U'))
X_test_teacher_prefix = vectorizer.transform(X_test['teacher_prefix'].values.astype('U'))

print(vectorizer.get_feature_names())


# when i executeed this error comes
#np.nan is an invalid document, expected byte or unicode string.
# then iconvert to unicode     just writ .astype('U') after the .values in fit and trainform
#https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-valueerror-np-nan-is-an-invalid-d
```

```
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
```

In [36]:
```
print("After vectorizations")
print(X_train_teacher_prefix.shape, y_train.shape)
print(X_cv_teacher_prefix.shape, y_cv.shape)
print(X_test_teacher_prefix.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(35912, 5) (35912,)
(17688, 5) (17688,)
(26400, 5) (26400,)
====================================================================================================
```

ENCODING:

Bow featurization

In [37]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10)# its a countvectors used for convert text to vectors
vectorizer.fit(train_preprocessed_essays)# that is learned from trainned  data

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer.transform(train_preprocessed_essays)
X_cv_bow = vectorizer.transform(cv_preprocessed_essays)
X_test_bow = vectorizer.transform(test_preprocessed_essays)



print("After vectorizations")
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("="*100)
# so the dimension of all are the same by using first fit and then transform
```

```
After vectorizations
(35912, 10591) (35912,)
(17688, 10591) (17688,)
(26400, 10591) (26400,)
====================================================================================================
```

In [38]:

```
vectorizer.fit(train_preprocessed_titles)# that is learned from trainned  data



# we use the fitted CountVectorizer to convert the text to vector
X_train_bow_title = vectorizer.transform(train_preprocessed_titles)
X_cv_bow_title= vectorizer.transform(cv_preprocessed_titles)
X_test_bow_title = vectorizer.transform(test_preprocessed_titles)



print("After vectorizations")
print(X_train_bow_title.shape, y_train.shape)
print(X_cv_bow_title.shape, y_cv.shape)
print(X_test_bow_title.shape, y_test.shape)
print("="*100)
# so the dimension of all are the same by using first fit and then transform
```

```
After vectorizations
(35912, 1625) (35912,)
(17688, 1625) (17688,)
(26400, 1625) (26400,)
====================================================================================================
```

In [39]:

```
#for titles
from sklearn.feature_extraction.text import TfidfVectorizer
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = TfidfVectorizer(min_df=10)# its a countvectors used for convert text to vectors
vectorizer.fit(train_preprocessed_titles)# that is learned from trainned  data



# we use the fitted CountVectorizer to convert the text to vector
X_train_tf_title = vectorizer.transform(train_preprocessed_titles)
X_cv_tf_title= vectorizer.transform(cv_preprocessed_titles)
X_test_tf_title = vectorizer.transform(test_preprocessed_titles)



print("After vectorizations")
print(X_train_tf_title.shape, y_train.shape)
print(X_cv_tf_title.shape, y_cv.shape)
print(X_test_tf_title.shape, y_test.shape)
print("="*100)
# so the dimension of alll are the same by using first fit and then transform
```

```
After vectorizations
(35912, 1625) (35912,)
(17688, 1625) (17688,)
(26400, 1625) (26400,)
====================================================================================================
```

In [40]:

```
#for essay
from sklearn.feature_extraction.text import TfidfVectorizer
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = TfidfVectorizer(min_df=10)# its a countvectors used for convert text to vectors
vectorizer.fit(train_preprocessed_essays)# that is learned from trainned  data



# we use the fitted CountVectorizer to convert the text to vector
X_train_tf_essay = vectorizer.transform(train_preprocessed_essays)
X_cv_tf_essay= vectorizer.transform(cv_preprocessed_essays)
X_test_tf_essay = vectorizer.transform(test_preprocessed_essays)



print("After vectorizations")
print(X_train_tf_essay.shape, y_train.shape)
print(X_cv_tf_essay.shape, y_cv.shape)
print(X_test_tf_essay.shape, y_test.shape)
print("="*100)
# so the dimension of alll are the same by using first fit and then transform
```

```
After vectorizations
(35912, 10591) (35912,)
(17688, 10591) (17688,)
(26400, 10591) (26400,)
====================================================================================================
```

Using Pretrained Models: Avg W2V

In [41]:

```
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039

def loadGloveModel(gloveFile):

    print ("Loading Glove Model")

    f = open(gloveFile,'r', encoding = 'utf8')

    model = {}

    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding

    print ("Done.",len(model)," words loaded!")

    return model
```

In [42]:

```
model = loadGloveModel('glove.42B.300d.txt')
```

```
Loading Glove Model

1917495it [08:02, 3977.89it/s]

Done. 1917495  words loaded!
```

In [43]:

```
glove_words = set(model.keys())
```

In [44]:
```
# average Word2Vec
# compute average word2vec for each review.
def func(wordlist):


  train_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
  for sentence in tqdm(wordlist): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length    # we are taking the 300 dimensions  very larg
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_avg_w2v_vectors.append(vector)

  print(len(train_avg_w2v_vectors))
  print(len(train_avg_w2v_vectors[0]))
  return train_avg_w2v_vectors
```

In [45]:
```
train_avg_w2v_vectors=func(train_preprocessed_essays)
test_avg_w2v_vectors=func(test_preprocessed_essays)
cv_avg_w2v_vectors=func(cv_preprocessed_essays)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 35912/35912 [00:18<00:00, 192
4.62it/s]

35912
300

100%|██████████████████████████████████████████████████████████████████████████| 26400/26400 [00:17<00:00, 150
8.48it/s]

26400
300

100%|██████████████████████████████████████████████████████████████████████████| 17688/17688 [00:11<00:00, 157
6.35it/s]

17688
300
```

In [46]:
```
cv_avg_w2v_vectors_title=func(cv_preprocessed_titles)


test_avg_w2v_vectors_title=func(test_preprocessed_titles)


train_avg_w2v_vectors_title=func(train_preprocessed_titles)
```

```
100%|█████████████████████████████████████████████████████████████████████████| 17688/17688 [00:00<00:00, 2749
0.93it/s]

17688
300

100%|█████████████████████████████████████████████████████████████████████████| 26400/26400 [00:00<00:00, 3189
9.96it/s]

26400
300

100%|█████████████████████████████████████████████████████████████████████████| 35912/35912 [00:01<00:00, 3119
3.90it/s]

35912
300
```

TFIDF weighted W2V

In [47]:
```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(train_preprocessed_essays)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [48]:

```
# average Word2Vec
# compute average word2vec for each review.
def tf_idf_done(word_list):

  train_title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
  for sentence in tqdm(word_list): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split():#.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(ser
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_title_tfidf_w2v_vectors.append(vector)

  print(len(train_title_tfidf_w2v_vectors))
  print(len(train_title_tfidf_w2v_vectors[0]))
  return train_title_tfidf_w2v_vectors
```

In [49]:

```
#For essays
train_tfidf_w2v_vectors=tf_idf_done(train_preprocessed_essays)
test_tfidf_w2v_vectors=tf_idf_done(test_preprocessed_essays)
cv_tfidf_w2v_vectors=tf_idf_done(cv_preprocessed_essays)
```

```
100%|██████████████████████████████████████████████████| 35912/35912 [02:55<00:00, 20
4.30it/s]

35912
300

100%|██████████████████████████████████████████████████| 26400/26400 [02:05<00:00, 20
9.71it/s]

26400
300

100%|██████████████████████████████████████████████████| 17688/17688 [01:26<00:00, 20
5.22it/s]

17688
300
```

In [50]:

```
train_title_tfidf_w2v_vectors=tf_idf_done(train_preprocessed_titles)
test_title_tfidf_w2v_vectors=tf_idf_done(test_preprocessed_titles)
cv_title_tfidf_w2v_vectors=tf_idf_done(cv_preprocessed_titles)
```

```
100%|████████████████████████████████████████████████| 35912/35912 [00:02<00:00, 1360
2.56it/s]

35912
300

100%|████████████████████████████████████████████████| 26400/26400 [00:02<00:00, 1267
6.10it/s]

26400
300

100%|█████████████████████████████████████████████████| 17688/17688 [00:01<00:00, 991
1.94it/s]

17688
300
```

Vectorizing Numerical features

In [51]:

```
price_data = dfr.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
dft = pd.merge(dft, price_data, on='id', how='left')
print(price_data.head(2))



#merging
# we also have to do this in train,test and cv
# so also merge the resource data with the trian,cv and test

X_train = pd.merge(X_train, price_data, on = "id", how = "left")
#print(x_train.columns)
X_test = pd.merge(X_test, price_data, on = "id", how = "left")
X_cv = pd.merge(X_cv, price_data, on = "id", how = "left")
```

```
        id   price  quantity
0  p000001  459.56         7
1  p000002  515.89        21
```

In [52]:

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

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

price_scalar = StandardScaler()

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

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

```
Mean : 299.4370358097572, Standard deviation : 372.76933088770426
```

Out[52]:

```
array([[-0.40091022],
       [-0.31219584],
       [-0.25808732],
       ...,
       [ 0.81670604],
       [-0.12958962],
       [-0.65125271]])
```

In [53]:

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

Out[53]:

```
array([[ 0.76002219],
       [ 0.90885418],
       [-0.72320069],
       ...,
       [ 0.08783707],
       [-0.31249093],
       [ 0.04499556]])
```

In [54]:

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

Out[54]:

```
array([[ 0.76002219],
       [ 0.90885418],
       [-0.72320069],
       ...,
       [ 0.08783707],
       [-0.31249093],
       [ 0.04499556]])
```

In [55]:

```
print(train_price_standar.shape, y_train.shape)
print(test_price_standar.shape, y_test.shape)
print(cv_price_standar.shape, y_cv.shape)
```

```
(35912, 1) (35912,)
(26400, 1) (26400,)
(17688, 1) (17688,)
```

In [56]:

```
# previous_year_projects
price_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)) # finding the mea
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
train_prev_proj_standar = price_scalar.transform(X_train['teacher_number_of_previously_posted_projects'].values.
train_prev_proj_standar
```

```
Mean : 11.19787257741145, Standard deviation : 28.396306850454696
```

Out[56]:

```
array([[-0.35912672],
       [-0.25347918],
       [ 0.27475853],
       ...,
       [-0.32391087],
       [ 0.27475853],
       [-0.39434257]])
```

In [57]:

```
# Now standardize the data with above maen and variance.
test_prev_proj_standar = price_scalar.transform(X_test['teacher_number_of_previously_posted_projects'].values.re
test_prev_proj_standar
```

Out[57]:

```
array([[-0.39434257],
       [-0.32391087],
       [-0.18304749],
       ...,
       [ 0.13389514],
       [-0.35912672],
       [-0.28869503]])
```

In [58]:

```
# Now standardize the data with above maen and variance.
cv_prev_proj_standar = price_scalar.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshap
cv_prev_proj_standar
```

Out[58]:

```
array([[-0.28869503],
       [-0.39434257],
       [-0.35912672],
       ...,
       [-0.39434257],
       [-0.35912672],
       [-0.32391087]])
```

In [59]:

```
print(train_prev_proj_standar.shape, y_train.shape)
print(test_prev_proj_standar.shape, y_test.shape)
print(cv_prev_proj_standar.shape, y_cv.shape)
```

```
(35912, 1) (35912,)
(26400, 1) (26400,)
(17688, 1) (17688,)
```

In [60]:

```
price_scalar.fit(X_train['quantity'].values.reshape(-1,1)) # finding the mean and standard deviation of this dat
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
train_qnty_standar = price_scalar.transform(X_train['quantity'].values.reshape(-1, 1))
train_qnty_standar
```

```
Mean : 16.84631877923814, Standard deviation : 26.19956294058221
```

Out[60]:

```
array([[-0.56666284],
       [ 0.23487725],
       [-0.49032569],
       ...,
       [-0.10863993],
       [-0.14680851],
       [ 0.69290015]])
```

In [61]:

```
# Now standardize the data with above mean and variance.
cv_qnty_standar = price_scalar.transform(X_cv['quantity'].values.reshape(-1, 1))
cv_qnty_standar
```

Out[61]:

```
array([[ 1.15092306],
       [-0.37581996],
       [-0.29948281],
       ...,
       [-0.41398854],
       [ 0.00586579],
       [-0.18497708]])
```

In [62]:

```
# Now standardize the data with above mean and variance.
test_qnty_standar = price_scalar.transform(X_test['quantity'].values.reshape(-1, 1))
test_qnty_standar
```

Out[62]:

```
array([[ 0.42572013],
       [-0.33765139],
       [-0.26131424],
       ...,
       [ 0.1585401 ],
       [-0.52849427],
       [-0.56666284]])
```

In [63]:

```
print(train_qnty_standar.shape, y_train.shape)
print(test_qnty_standar.shape, y_test.shape)
print(cv_qnty_standar.shape, y_cv.shape)
```

```
(35912, 1) (35912,)
(26400, 1) (26400,)
(17688, 1) (17688,)
```

MERGING

In [64]:

```
#project_categories
print("Shape of Train  ->",X_train_cat.shape)
print("Shape of test  ->",X_test_cat.shape)
print("Shape of cv ->",X_cv_cat.shape)
```

```
Shape of Train  -> (35912, 9)
Shape of test  -> (26400, 9)
Shape of cv -> (17688, 9)
```

In [65]:

```
#project_subcategories
print("Shape of Train  ->",X_train_subcat.shape)
print("Shape of test  ->",X_test_subcat.shape)
print("Shape of cv ->",X_cv_subcat.shape)
```

```
Shape of Train  -> (35912, 30)
Shape of test  -> (26400, 30)
Shape of cv -> (17688, 30)
```

In [66]:

```
#project_school_state
print("Shape of Train  ->",X_train_school_state.shape)
print("Shape of test  ->",X_test_school_state.shape)
print("Shape of cv ->",X_cv_school_state.shape)
```

```
Shape of Train  -> (35912, 51)
Shape of test  -> (26400, 51)
Shape of cv -> (17688, 51)
```

In [67]:

```
#project_grade_category
print("Shape of Train  ->",X_train_project_grade_category.shape)
print("Shape of test  ->",X_test_project_grade_category.shape)
print("Shape of cv ->",X_cv_project_grade_category.shape)
```

```
Shape of Train  -> (35912, 4)
Shape of test  -> (26400, 4)
Shape of cv -> (17688, 4)
```

In [68]:

```
#project_teacher_prefix
print("Shape of Train  ->",X_train_teacher_prefix.shape)
print("Shape of test  ->",X_test_teacher_prefix.shape)
print("Shape of cv ->",X_cv_teacher_prefix.shape)
```

```
Shape of Train  -> (35912, 5)
Shape of test  -> (26400, 5)
Shape of cv -> (17688, 5)
```

All numerical:

In [69]:

```
#project_quantity
print("Shape of Train  ->",train_qnty_standar.shape)
print("Shape of test  ->",test_qnty_standar.shape)
print("Shape of cv ->",cv_qnty_standar.shape)
```

```
Shape of Train  -> (35912, 1)
Shape of test  -> (26400, 1)
Shape of cv -> (17688, 1)
```

In [70]:

```
#project_price
print("Shape of Train  ->",train_price_standar.shape)
print("Shape of test  ->",test_price_standar.shape)
print("Shape of cv ->",cv_price_standar.shape)
```

```
Shape of Train  -> (35912, 1)
Shape of test  -> (26400, 1)
Shape of cv -> (17688, 1)
```

In [71]:

```
##project_previous_year_teacher_projects
print("Shape of Train  ->",train_prev_proj_standar.shape)
print("Shape of test  ->",test_prev_proj_standar.shape)
print("Shape of cv ->",cv_prev_proj_standar.shape)
```

```
Shape of Train  -> (35912, 1)
Shape of test  -> (26400, 1)
Shape of cv -> (17688, 1)
```

All featurization Bow,tf-idf etc ESSAY AND TITLES:

In [72]:

```
#BOW Project_Essays
print("- "*50)
print("Shape of train ",X_train_bow.shape)
print("Shape of test  ",X_test_bow.shape)
print("Shape of cv  ",X_cv_bow.shape)
print("- "*50)
#BOW Project_Titles
print("Shape of train  ",X_train_bow_title.shape)
print("Shape of test  ",X_test_bow_title.shape)
print("Shape of cv  ",X_cv_bow_title.shape)
print("- "*50)
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 
Shape of train  (35912, 10591)
Shape of test   (26400, 10591)
Shape of cv   (17688, 10591)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 
Shape of train   (35912, 1625)
Shape of test   (26400, 1625)
Shape of cv   (17688, 1625)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 
```

In [73]:

```
#TFIDF Project_Essays
print("- "*50)
print("Shape of train ",X_train_tf_essay.shape)
print("Shape of test  ",X_test_tf_essay.shape)
print("Shape of cv  ",X_cv_tf_essay.shape)
print("- "*50)
#TFIDF Project_Title

print("Shape of train  ",X_train_tf_title.shape)
print("Shape of test  ",X_test_tf_title.shape)
print("Shape of  cv ",X_cv_tf_title.shape)
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Shape of train  (35912, 10591)
Shape of test   (26400, 10591)
Shape of cv   (17688, 10591)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Shape of train   (35912, 1625)
Shape of test   (26400, 1625)
Shape of  cv  (17688, 1625)
```

In [74]:

```
# list to np.array
train_avg_w2v_vectors_title=np.array(train_avg_w2v_vectors_title)
test_avg_w2v_vectors_title=np.array(test_avg_w2v_vectors_title)
cv_avg_w2v_vectors_title=np.array(cv_avg_w2v_vectors_title)



train_avg_w2v_vectors=np.array(train_avg_w2v_vectors)
test_avg_w2v_vectors=np.array(test_avg_w2v_vectors)
cv_avg_w2v_vectors=np.array(cv_avg_w2v_vectors)
```

In [75]:
```
#TFIDF Project_Essays
print("- "*50)
print("Shape of train ",train_avg_w2v_vectors.shape)#train_avg_w2v_vectors_title
print("Shape of test  ",test_avg_w2v_vectors.shape)
print("Shape of cv  ",cv_avg_w2v_vectors.shape)
print("- "*50)
#TFIDF Project_Title

print("Shape of train  ",train_avg_w2v_vectors_title.shape)
print("Shape of test  ",test_avg_w2v_vectors_title.shape)
print("Shape of  cv ",cv_avg_w2v_vectors_title.shape)
print("- "*50)
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Shape of train  (35912, 300)
Shape of test   (26400, 300)
Shape of cv   (17688, 300)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Shape of train   (35912, 300)
Shape of test   (26400, 300)
Shape of  cv  (17688, 300)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

In [76]:
```
# list to np.array
train_title_tfidf_w2v_vectors=np.array(train_title_tfidf_w2v_vectors)
test_title_tfidf_w2v_vectors=np.array(test_title_tfidf_w2v_vectors)
cv_title_tfidf_w2v_vectors=np.array(cv_title_tfidf_w2v_vectors)



train_essay_tfidf_w2v_vectors=np.array(train_tfidf_w2v_vectors)
test_essay_tfidf_w2v_vectors=np.array(test_tfidf_w2v_vectors)
cv_essay_tfidf_w2v_vectors=np.array(cv_tfidf_w2v_vectors)
```

In [77]:
```
#TFIDF Project_Essays
print("- "*50)
print("Shape of train ",train_essay_tfidf_w2v_vectors.shape)#train_avg_w2v_vectors_title
print("Shape of test  ",test_essay_tfidf_w2v_vectors.shape)
print("Shape of cv  ",cv_essay_tfidf_w2v_vectors.shape)
print("- "*50)
#TFIDF Project_Title

print("Shape of train  ",train_title_tfidf_w2v_vectors.shape)
print("Shape of test  ",test_title_tfidf_w2v_vectors.shape)
print("Shape of  cv ",cv_title_tfidf_w2v_vectors.shape)
print("- "*50)
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 
Shape of train  (35912, 300)
Shape of test   (26400, 300)
Shape of cv   (17688, 300)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 
Shape of train   (35912, 300)
Shape of test   (26400, 300)
Shape of  cv  (17688, 300)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 
```

In [78]:
```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_set1_train = hstack((X_train_bow_title,X_train_bow,train_prev_proj_standar,train_price_standar,train_qnty_star
                      X_train_teacher_prefix,X_train_cat,X_train_subcat,
                      X_train_project_grade_category,X_train_school_state))


print(X_set1_train.shape, y_train.shape)
```

```
(35912, 12318) (35912,)
```

In [79]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_set1_cv = hstack((X_cv_bow_title,X_cv_bow,cv_prev_proj_standar,cv_price_standar,cv_qnty_standar,
                      X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,
                      X_cv_project_grade_category,X_cv_school_state))


print(X_set1_cv.shape, y_cv.shape)
```

```
(17688, 12318) (17688,)
```

In [80]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_set1_test = hstack((X_test_bow_title,X_test_bow,test_prev_proj_standar,test_price_standar,test_qnty_standar,
                      X_test_teacher_prefix,X_test_cat,X_test_subcat,
                      X_test_project_grade_category,X_test_school_state))


print(X_set1_test.shape, y_test.shape)
```

```
(26400, 12318) (26400,)
```

In [81]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_set2_train = hstack((X_train_tf_essay,X_train_tf_title,train_prev_proj_standar,train_price_standar,train_qnty_
                      X_train_teacher_prefix,X_train_cat,X_train_subcat,
                      X_train_project_grade_category,X_train_school_state))


print(X_set2_train.shape, y_train.shape)
```

```
(35912, 12318) (35912,)
```

In [82]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_set2_cv = hstack((X_cv_tf_essay,X_cv_tf_title,cv_prev_proj_standar,cv_price_standar,cv_qnty_standar,
                     X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,
                     X_cv_project_grade_category,X_cv_school_state))


print(X_set2_cv.shape, y_cv.shape)
```

```
(17688, 12318) (17688,)
```

In [83]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_set2_test = hstack((X_test_tf_essay,X_test_tf_title,test_prev_proj_standar,test_price_standar,test_qnty_standa
                     X_test_teacher_prefix,X_test_cat,X_test_subcat,
                     X_test_project_grade_category,X_test_school_state))


print(X_set2_test.shape, y_test.shape)
```

```
(26400, 12318) (26400,)
```

In [84]:

```
y_train1=y_train[:8200]
y_test1=y_test[:6000]
y_cv1=y_cv[:4800]
```

In [85]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_set3_train = hstack((train_avg_w2v_vectors,train_avg_w2v_vectors_title,train_prev_proj_standar,train_price_sta
                     X_train_teacher_prefix,X_train_cat,X_train_subcat,
                     X_train_project_grade_category,X_train_school_state))


print(X_set3_train.shape, y_train.shape)
```

```
(35912, 702) (35912,)
```

In [86]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_set3_cv = hstack((cv_avg_w2v_vectors,cv_avg_w2v_vectors_title,cv_prev_proj_standar,cv_price_standar,cv_qnty_st
                      X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,
                      X_cv_project_grade_category,X_cv_school_state))


print(X_set3_cv.shape, y_cv.shape)
```

```
(17688, 702) (17688,)
```

In [87]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_set3_test = hstack((test_avg_w2v_vectors,test_avg_w2v_vectors_title,test_prev_proj_standar,test_price_standar,
                      X_test_teacher_prefix,X_test_cat,X_test_subcat,
                      X_test_project_grade_category,X_test_school_state))


print(X_set3_test.shape, y_test.shape)
```

```
(26400, 702) (26400,)
```

In [88]:

```
# convert to dataframe
#https://stackoverflow.com/questions/20763012/creating-a-pandas-dataframe-from-a-numpy-array-how-do-i-specify-th
X_set3_test=pd.DataFrame(X_set3_test.toarray())
#print(X_set4_test[0:10])
X_set3_cv=pd.DataFrame(X_set3_cv.toarray())

X_set3_train=pd.DataFrame(X_set3_train.toarray())


# train take 7000 ,test take 3000
X_set3_test=X_set3_test[:6000]
X_set3_train=X_set3_train[:8200]
X_set3_cv=X_set3_cv[:4800]

print(X_set3_test.shape, y_test1.shape)
print(X_set3_cv.shape, y_cv1.shape)
print(X_set3_train.shape, y_train1.shape)
```

```
(6000, 702) (6000,)
(4800, 702) (4800,)
(8200, 702) (8200,)
```

In [89]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_set4_train = hstack((train_tfidf_w2v_vectors,train_title_tfidf_w2v_vectors,train_prev_proj_standar,train_price
                      X_train_teacher_prefix,X_train_cat,X_train_subcat,
                      X_train_project_grade_category,X_train_school_state))


print(X_set4_train.shape, y_train.shape)
```

```
(35912, 702) (35912,)
```

In [90]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_set4_cv = hstack((cv_tfidf_w2v_vectors,cv_title_tfidf_w2v_vectors,cv_prev_proj_standar,cv_price_standar,cv_qnt
                      X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,
                      X_cv_project_grade_category,X_cv_school_state))


print(X_set4_cv.shape, y_cv.shape)
```

```
(17688, 702) (17688,)
```

In [91]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_set4_test = hstack((test_title_tfidf_w2v_vectors,test_tfidf_w2v_vectors,test_prev_proj_standar,test_price_star
                      X_test_teacher_prefix,X_test_cat,X_test_subcat,
                      X_test_project_grade_category,X_test_school_state))


print(X_set4_test.shape, y_test.shape)
```

```
(26400, 702) (26400,)
```

In [92]:

```
X_set4_test=pd.DataFrame(X_set4_test.toarray())
#print(X_set4_test[0:10])
X_set4_cv=pd.DataFrame(X_set4_cv.toarray())

X_set4_train=pd.DataFrame(X_set4_train.toarray())
```

In [93]:

```
X_set4_test=X_set4_test[:6000]
X_set4_train=X_set4_train[:8200]
X_set4_cv=X_set4_cv[:4800]
```

In [94]:

```
print(X_set4_test.shape, y_test1.shape)
print(X_set4_cv.shape, y_cv1.shape)
print(X_set4_train.shape, y_train1.shape)
```

```
(6000, 702) (6000,)
(4800, 702) (4800,)
(8200, 702) (8200,)
```

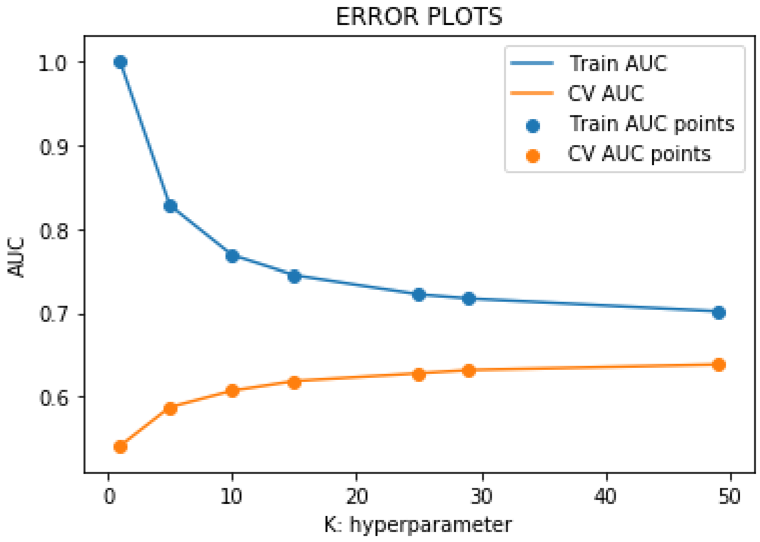
Applying knn section

2.4.1 Applying KNN brute force on BOW, SET 1

In [97]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

import matplotlib.pyplot as plt

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

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

"""


train_auc = []
cv_auc = []
K = [1, 5, 10, 15,25,29,49]# min k causes overfitting, max k causes underfitting
#K = range(1,50,2)
for i in  tqdm(K):

    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute')# takes the k from the  i th list value
    neigh.fit(X_set1_train, y_train)# fit the model


    # 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 =  neigh.predict_proba(X_set1_train)[:,1]#Return probability estimates for the  set1x ,for the
    y_cv_pred =  neigh.predict_proba(X_set1_cv)[:,1]#Return probability estimates for the setcvx,for the class l


    # roc curve
    #Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.
    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.show()
```

```
  0%|                                                                  | 0/7 [00:00
<?, ?it/s]
 14%|████████                                                    | 1/7 [06:34<39:26, 39
4.50s/it]
 29%|████████████████                                            | 2/7 [12:48<32:21, 38
8.24s/it]
 43%|████████████████████████                                   | 3/7 [19:06<25:41, 38
5.37s/it]
 57%|████████████████████████████████                            | 4/7 [25:25<19:10, 38
3.41s/it]
 71%|████████████████████████████████████████                    | 5/7 [31:46<12:45, 38
2.51s/it]
 86%|████████████████████████████████████████████████            | 6/7 [37:59<06:19, 37
9.90s/it]
100%|████████████████████████████████████████████████████████| 7/7 [44:11<00:00, 37
7.47s/it]
```



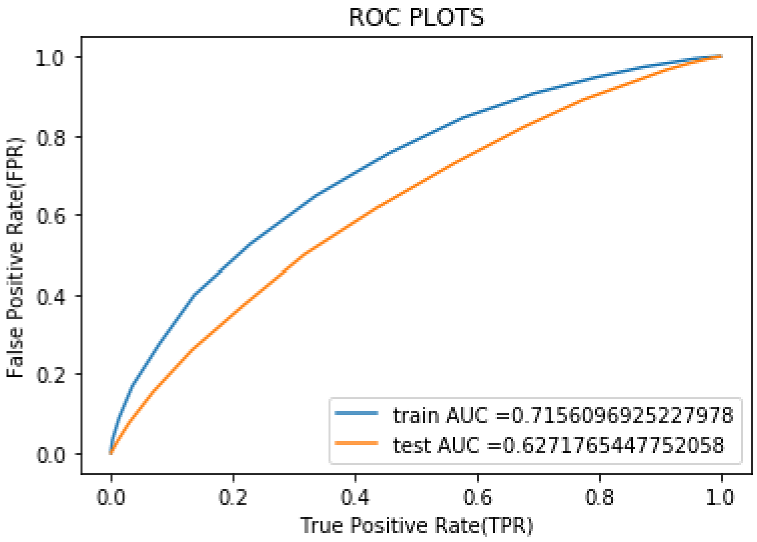
In [98]:

```
score_t_cv = [x for x in cv_auc]
opt_t_cv = K[score_t_cv.index(max(score_t_cv))]
print("Maximum AUC score of cv is:" + ' ' + str(max(score_t_cv)))
print("Corresponding k value of cv is:",opt_t_cv, '\n')
best_k=opt_t_cv
print(best_k)
```

```
Maximum AUC score of cv is: 0.6384592822243068
Corresponding k value of cv is: 49

49
```

Fitting Model to Hyper-Parameter Curve (Using bruteforce KNN)

In [99]:
```
# 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=32,algorithm='brute')
neigh.fit(X_set1_train ,y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_set1_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_set1_test)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("ROC PLOTS")
plt.show()
```



OBSERVATIONS: As we seen form the roc plot ,as we increase the k value this roc curve improve little bit , not more because this is the imbalanced dataset,so lets see in further plots.

Confusion matrix :

In [100]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_set1_train )))
```

```
Train confusion matrix
[[  132  5311]
 [   75 30394]]
```

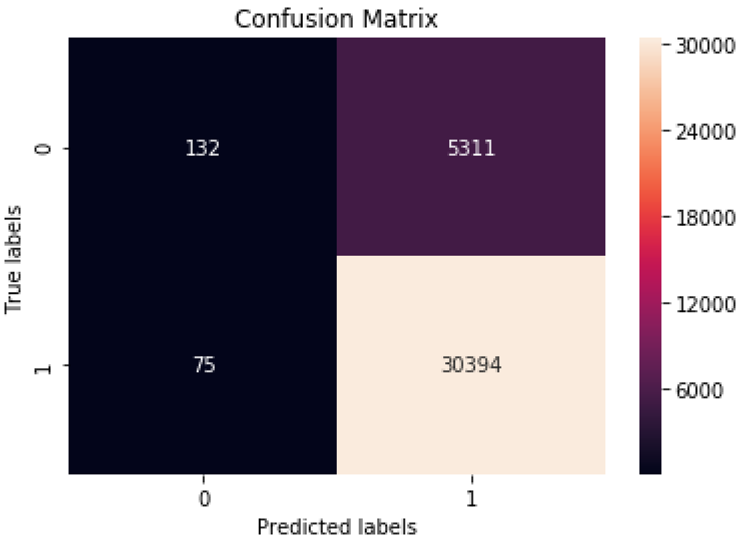
In [101]:

```
from sklearn.metrics import classification_report

print(classification_report(y_train,neigh.predict(X_set1_train) ))
```

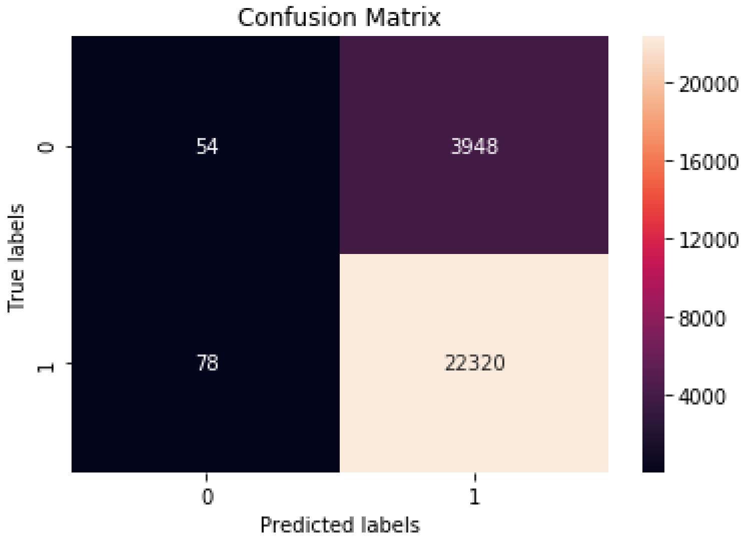
```
              precision    recall  f1-score   support

           0       0.64      0.02      0.05      5443
           1       0.85      1.00      0.92     30469

    accuracy                           0.85     35912
   macro avg       0.74      0.51      0.48     35912
weighted avg       0.82      0.85      0.79     35912
```

In [102]:

```
import seaborn as sns
import matplotlib.pyplot as plt

ax= plt.subplot()

def predict(proba,threshold,fpr,tpr):
    t=threshold[np.argmax(fpr*(1-tpr))]
    print("the maximun value of tpr*(1-fpr)",np.round(max(tpr*(1-fpr)),2) ,"for threshold",np.round(t,2))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions


sns.heatmap(confusion_matrix(y_train, neigh.predict(X_set1_train )), annot=True, ax = ax,fmt='g');
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```



In [103]:
```
import seaborn as sns
import matplotlib.pyplot as plt

ax= plt.subplot()

def predict(proba,threshold,fpr,tpr):
    t=threshold[np.argmax(fpr*(1-tpr))]
    print("the maximun value of tpr*(1-fpr)",np.round(max(tpr*(1-fpr)),2) ,"for threshold",np.round(t,2))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions


sns.heatmap(confusion_matrix(y_test, neigh.predict(X_set1_test)), annot=True, ax = ax,fmt='g');
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```



OBSERVATOINS: As we see from this confusion matrix ,In our prediction true positives is of greater weitage,beacuse of high k value all the negatives are dominating so that true negaties arezero,all are predictee wrong, but for the better prediction we want tp and tn both to be more,but if we choose k to be low then our roc cure,auc value less than ,50 or 50 worst value, if we increasee k then it will dominating the posities values,so lets see in further plots ,what inference we make from this plots, and what is auc and confusion matrix, but from now i am clear that , This imbalancing is not good for our model, and also if our best k to be big then, cause of underfitting, so simply means we have to take more data for overcome underfitting,but more data can;t be handled by my laptop.

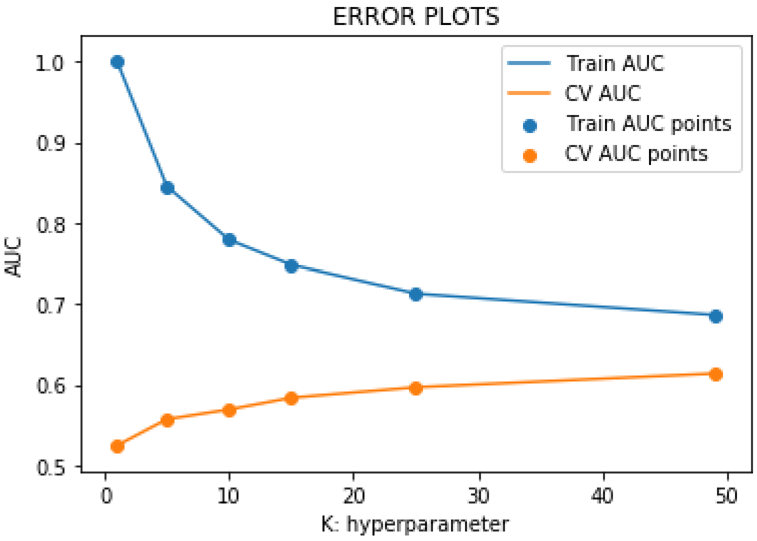
Also their a reason why this auc is not so good,knn is a basic algorithm,means not so good as compared to some advanced ml algorithm, so may be that is the reason for our not so good prediction like roc and confusion matrix is not good.

2.4.2 Applying KNN brute force on TFIDF, SET 2

In [104]:

```
#http://localhost:8888/notebooks/Assignment_SAMPLE_SOLUTION%20(1).ipynb  (for reference) Which you provided

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

import matplotlib.pyplot as plt

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

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

"""


train_auc = []
cv_auc = []
K = [1, 5, 10, 15,25,49]# min k causes overfitting, max k causes underfitting
for i in  tqdm(K):

    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute')# takes the k from the  i th list value
    neigh.fit(X_set2_train, y_train)# fit the model


    # 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 =  neigh.predict_proba(X_set2_train)[:,1]#Return probability estimates for the  set1x ,for the
    y_cv_pred =  neigh.predict_proba(X_set2_cv)[:,1]#Return probability estimates for the setcvx,for the class l


    # roc curve
    #Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.
    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.show()
```

```
  0%|                                                                                       | 0/6 [00:00
<?, ?it/s]
 17%|██████████▌                                                                    | 1/6 [02:59<14:57, 17
9.58s/it]
 33%|█████████████████████▏                                                         | 2/6 [05:56<11:55, 17
8.81s/it]
 50%|████████████████████████████████                                              | 3/6 [08:50<08:52, 17
7.46s/it]
 67%|██████████████████████████████████████████▋                                   | 4/6 [11:39<05:49, 17
4.90s/it]
 83%|█████████████████████████████████████████████████████▎                        | 5/6 [14:37<02:55, 17
5.88s/it]
100%|████████████████████████████████████████████████████████████████████| 6/6 [17:27<00:00, 17
4.09s/it]
```

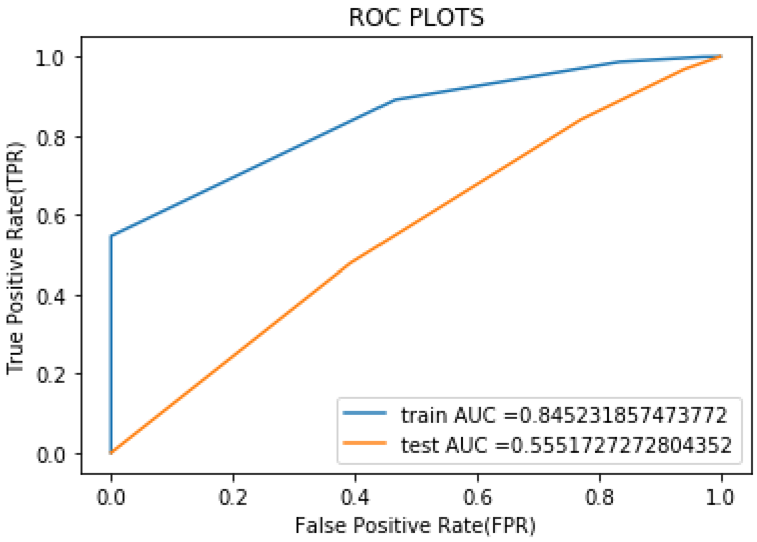


In [105]:
```
score_t_cv_3 = [x for x in cv_auc]
opt_t_cv_3 = K[score_t_cv.index(max(score_t_cv))-1]
print("Maximum AUC score of cv is:" + ' ' + str(max(score_t_cv_3)))
print("Corresponding k value of cv is:",opt_t_cv_3, '\n')
```

```
Maximum AUC score of cv is: 0.6137493041603588
Corresponding k value of cv is: 49
```

Fitting Model to Hyper-Parameter Curve (using brute force KNN):

In [149]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=5,algorithm='brute')
neigh.fit(X_set2_train ,y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_set2_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_set2_test)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("ROC PLOTS")
plt.show()
```



OBSERVATONS: We can see in tf-idf,roc curve improve when we increaes the k value , as i already said this is underfitting ,because of imbalancing, so our imference is not so good in real word scenarios.And confusing matrix also has domating class.

COnfusion matrix

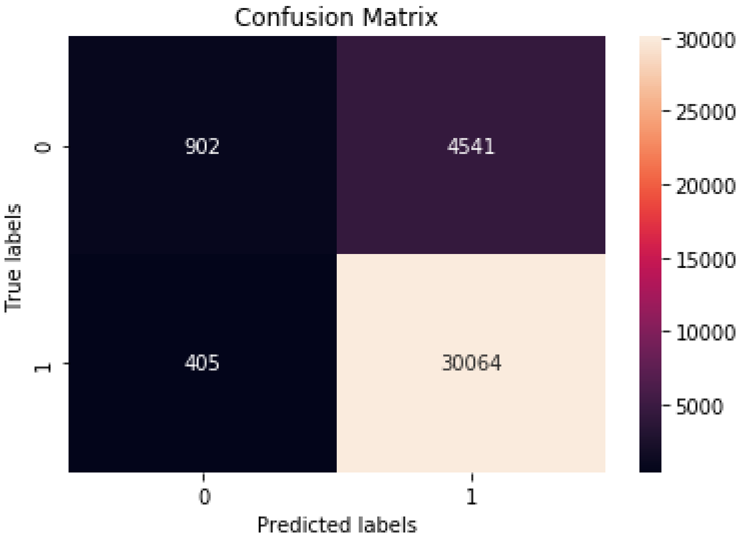
In [150]:

```
from sklearn.metrics import classification_report

print(classification_report(y_train,neigh.predict(X_set2_train) ))
```

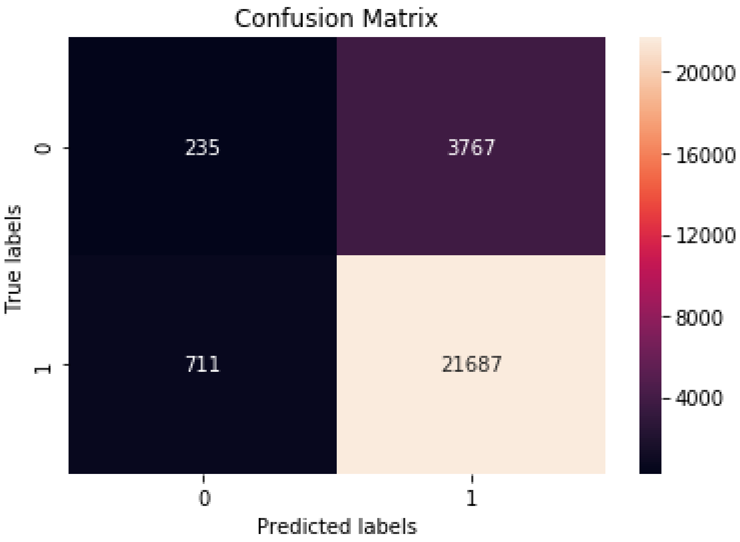
```
              precision    recall  f1-score   support

           0       0.69      0.17      0.27      5443
           1       0.87      0.99      0.92     30469

    accuracy                           0.86     35912
   macro avg       0.78      0.58      0.60     35912
weighted avg       0.84      0.86      0.82     35912
```

In [151]:

```python
import seaborn as sns
import matplotlib.pyplot as plt

ax= plt.subplot()

def predict(proba,threshold,fpr,tpr):
    t=threshold[np.argmax(fpr*(1-tpr))]
    print("the maximun value of tpr*(1-fpr)",np.round(max(tpr*(1-fpr)),2) ,"for threshold",np.round(t,2))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions


sns.heatmap(confusion_matrix(y_train, neigh.predict(X_set2_train )), annot=True, ax = ax,fmt='g');
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```



In [152]:
```
import seaborn as sns
import matplotlib.pyplot as plt

ax= plt.subplot()

def predict(proba,threshold,fpr,tpr):
    t=threshold[np.argmax(fpr*(1-tpr))]
    print("the maximun value of tpr*(1-fpr)",np.round(max(tpr*(1-fpr)),2) ,"for threshold",np.round(t,2))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions


sns.heatmap(confusion_matrix(y_test, neigh.predict(X_set2_test )), annot=True, ax = ax,fmt='g');
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```



Observation: Due to highly imbalance in the data set or due to high k vlaue this is totaly dominating the negative class

Apply the wordtovec for set3

2.4.3 Applying KNN brute force on AVG W2V, SET 3

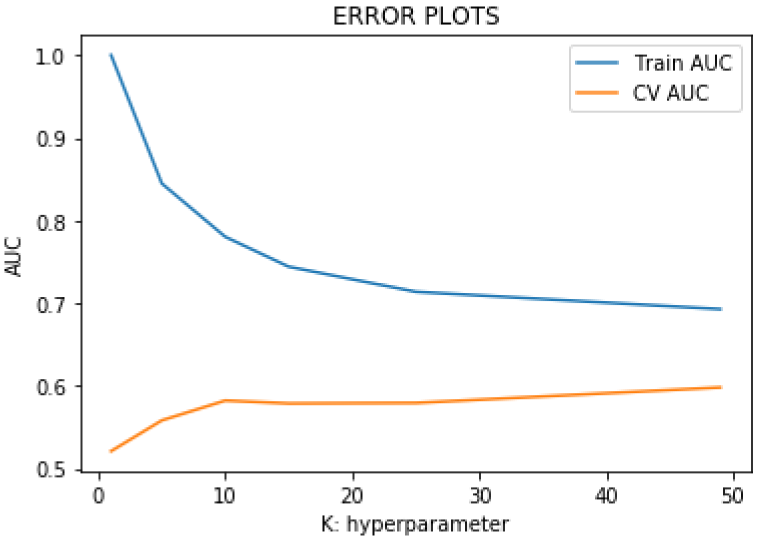
In [113]:

```
print(X_set3_train.shape,y_train1.shape)
```

```
(8200, 702) (8200,)
```

In [114]:
```
#http://localhost:8888/notebooks/Assignment_SAMPLE_SOLUTION%20(1).ipynb  (for reference) Which you provided

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

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

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

"""


train_auc = []
cv_auc = []
K = [1, 5, 10, 15,25,49]# min k causes overfitting, max k causes underfitting
for i in tqdm(K):

    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute')# takes the k from the  i th list value
    neigh.fit(X_set3_train, y_train1)# for the model


    # 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 =  neigh.predict_proba(X_set3_train)[:,1]#Return probability estimates for the  set3x ,for the
    y_cv_pred =  neigh.predict_proba(X_set3_cv)[:,1]#Return probability estimates for the set3cvx,for the class


    # roc curve
    #Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.
    train_auc.append(roc_auc_score(y_train1,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv1, y_cv_pred))

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

```
  0%|                                                                                             | 0/6 [00:00
<?, ?it/s]
 17%|██████████████                                                                       | 1/6 [00:07<00:35,
7.07s/it]
 33%|████████████████████████████                                                         | 2/6 [00:15<00:30,
7.54s/it]
 50%|███████████████████████████████████████████                                          | 3/6 [00:24<00:23,
7.90s/it]
 67%|█████████████████████████████████████████████████████████                            | 4/6 [00:33<00:16,
8.19s/it]
 83%|███████████████████████████████████████████████████████████████████████              | 5/6 [00:41<00:08,
8.32s/it]
100%|█████████████████████████████████████████████████████████████████████████████████████| 6/6 [00:50<00:00,
8.48s/it]
```



In [115]:

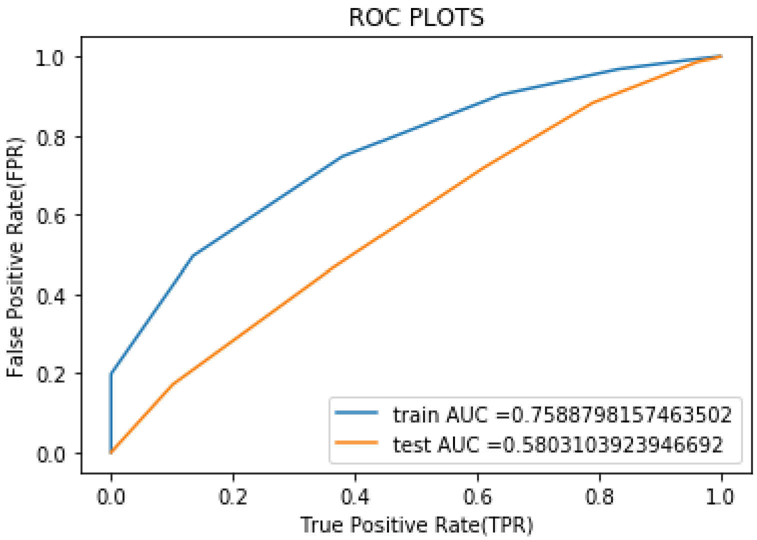
```
scor = [x for x in cv_auc]
opt_t_cv_3 = K[scor.index(max(scor))]
print("Maximum AUC score of cv is:" + ' ' + str(max(scor)))
print("Corresponding k value of cv is:",opt_t_cv_3, '\n')
best_k=opt_t_cv_3
print(best_k)
```

```
Maximum AUC score of cv is: 0.5981152108155476
Corresponding k value of cv is: 49

49
```

Fitting Model to Hyper-Parameter Curve (using Bruteforce KNN):

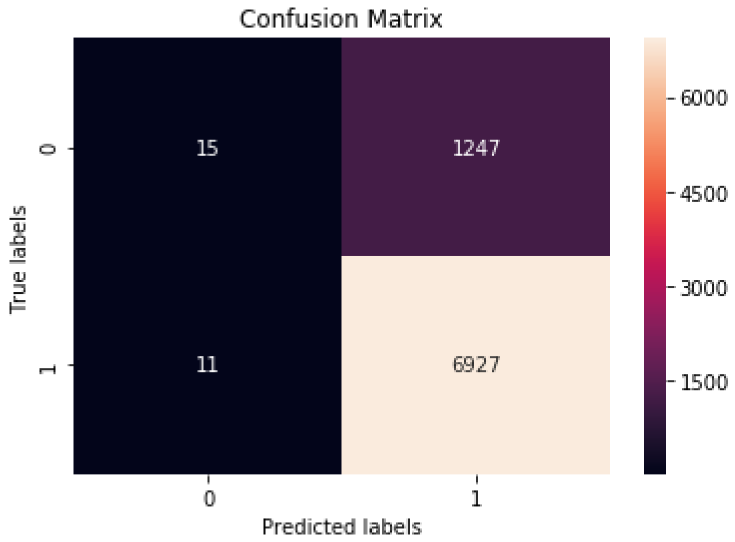
In [131]:

```
# 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=13,algorithm='brute')
neigh.fit(X_set3_train ,y_train1)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train1, neigh.predict_proba(X_set3_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test1, neigh.predict_proba(X_set3_test)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("ROC PLOTS")
plt.show()
```



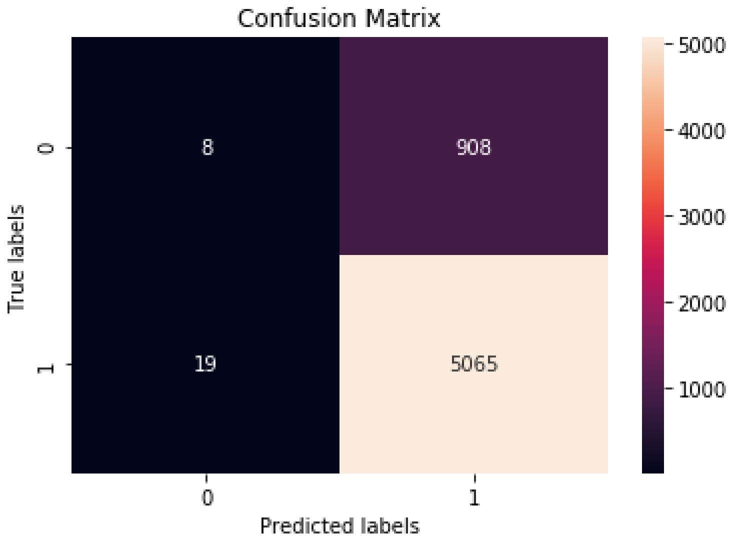
COnfusion matrix

In [132]:
```
import seaborn as sns
import matplotlib.pyplot as plt

ax= plt.subplot()

def predict(proba,threshold,fpr,tpr):
    t=threshold[np.argmax(fpr*(1-tpr))]
    print("the maximun value of tpr*(1-fpr)",np.round(max(tpr*(1-fpr)),2) ,"for threshold",np.round(t,2))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions


sns.heatmap(confusion_matrix(y_train1, neigh.predict(X_set3_train)), annot=True, ax = ax,fmt='g');
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```



In [133]:

```
import seaborn as sns
import matplotlib.pyplot as plt

ax= plt.subplot()

def predict(proba,threshold,fpr,tpr):
    t=threshold[np.argmax(fpr*(1-tpr))]
    print("the maximun value of tpr*(1-fpr)",np.round(max(tpr*(1-fpr)),2) ,"for threshold",np.round(t,2))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions


sns.heatmap(confusion_matrix(y_test1, neigh.predict(X_set3_test)), annot=True, ax = ax,fmt='g');
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```



Observations: We can't make some correct inferences from this confusion matrix also its so bad confusion matrix.'Totaly worst confusion matrix, just because of imbalanced data

2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

In [134]:

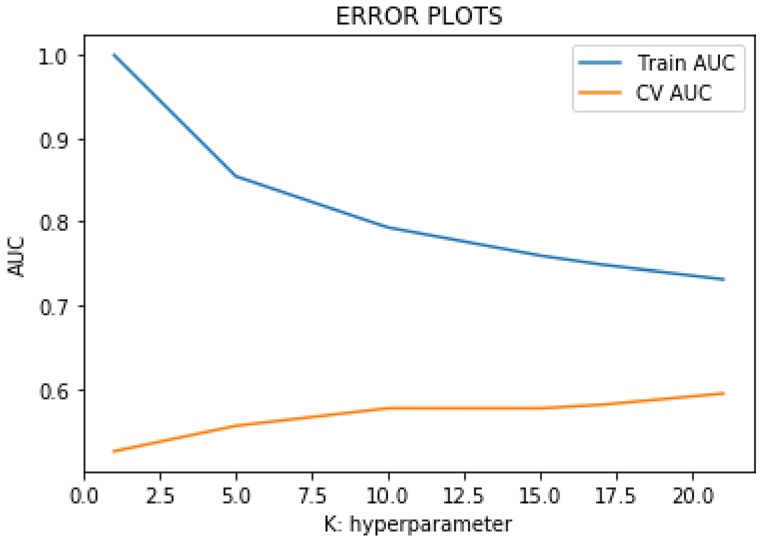
```
print(X_set4_train.shape,y_train1.shape)
```

```
(8200, 702) (8200,)
```

In [135]:

```
#http://localhost:8888/notebooks/Assignment_SAMPLE_SOLUTION%20(1).ipynb  (for reference) Which you provided

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

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

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

"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 17,21]# min k causes overfitting, max k causes underfitting
for i in tqdm(K):

    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute')# takes the k from the  i th list value
    neigh.fit(X_set4_train, y_train1)# for the model


    # 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 =  neigh.predict_proba(X_set4_train)[:,1]#Return probability estimates for the  set3x ,for the
    y_cv_pred =  neigh.predict_proba(X_set4_cv)[:,1]#Return probability estimates for the set3cvx,for the class


    # roc curve
    #Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.
    train_auc.append(roc_auc_score(y_train1,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv1, y_cv_pred))

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

```
  0%|                                                                                            | 0/6 [00:00
<?, ?it/s]
 17%|██████████████                                                                      | 1/6 [00:06<00:34,
6.98s/it]
 33%|████████████████████████████                                                        | 2/6 [00:15<00:30,
7.56s/it]
 50%|██████████████████████████████████████████                                          | 3/6 [00:24<00:23,
7.90s/it]
 67%|████████████████████████████████████████████████████████                            | 4/6 [00:33<00:16,
8.15s/it]
 83%|██████████████████████████████████████████████████████████████████████              | 5/6 [00:42<00:08,
8.36s/it]
100%|████████████████████████████████████████████████████████████████████████████████████| 6/6 [00:51<00:00,
8.51s/it]
```



In [136]:

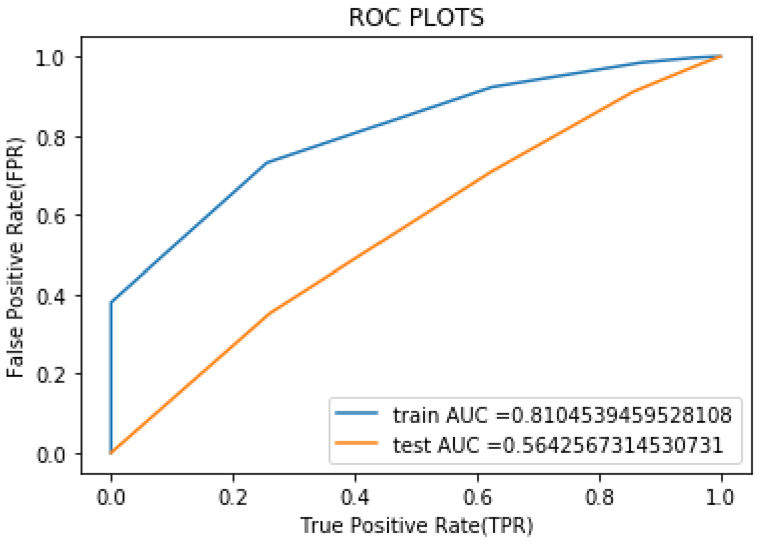
```
sc = [x for x in cv_auc]
opt_t_cv_4 = K[sc.index(max(sc ))]
print("Maximum AUC score of cv is:" + ' ' + str(max(sc )))
print("Corresponding k value of cv is:",opt_t_cv_4, '\n')
best_k=opt_t_cv_4
print(best_k)
```

```
Maximum AUC score of cv is: 0.5944314816476711
Corresponding k value of cv is: 21

21
```

Fitting Model to Hyper-Parameter Curve: (using brute force KNN)

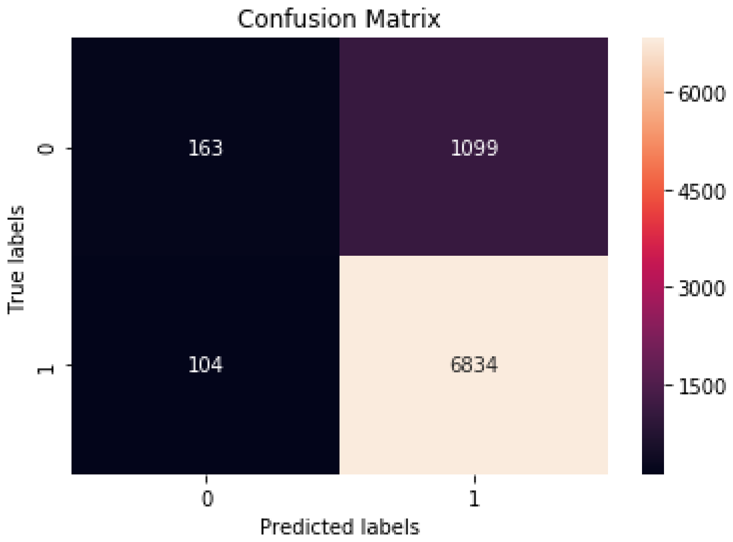
In [140]:

```
# 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=8,algorithm='brute')
neigh.fit(X_set4_train ,y_train1)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train1, neigh.predict_proba(X_set4_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test1, neigh.predict_proba(X_set4_test)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("ROC PLOTS")
plt.show()
```



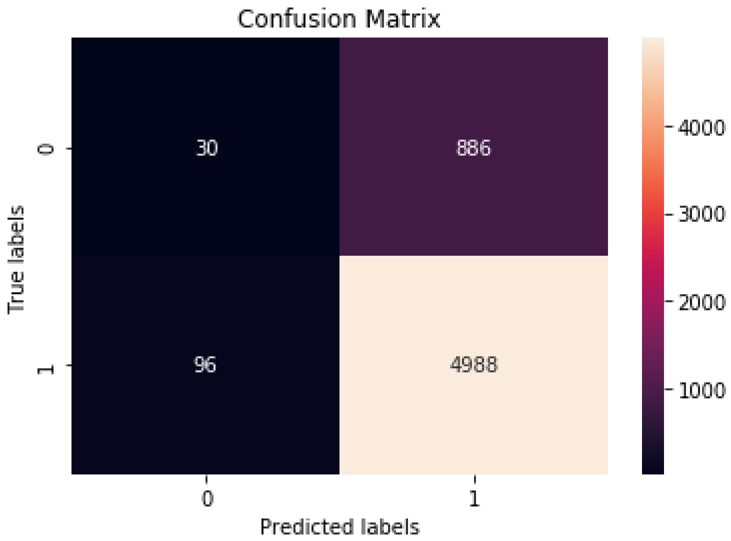
COnfusion matrix

In [141]:
```
import seaborn as sns
import matplotlib.pyplot as plt

ax= plt.subplot()

def predict(proba,threshold,fpr,tpr):
    t=threshold[np.argmax(fpr*(1-tpr))]
    print("the maximun value of tpr*(1-fpr)",np.round(max(tpr*(1-fpr)),2) ,"for threshold",np.round(t,2))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions


sns.heatmap(confusion_matrix(y_train1, neigh.predict(X_set4_train )), annot=True, ax = ax,fmt='g');
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```



In [142]:

```
import seaborn as sns
import matplotlib.pyplot as plt

ax= plt.subplot()

def predict(proba,threshold,fpr,tpr):
    t=threshold[np.argmax(fpr*(1-tpr))]
    print("the maximun value of tpr*(1-fpr)",np.round(max(tpr*(1-fpr)),2) ,"for threshold",np.round(t,2))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions


sns.heatmap(confusion_matrix(y_test1, neigh.predict(X_set4_test )), annot=True, ax = ax,fmt='g');
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```



2.5 Feature selection with `SelectKBest`: (Using Bruteforce KNN)

In [143]:

```
# apply this on tf-idf
print(X_set2_train.shape, y_train.shape)
print(X_set2_test.shape, y_test.shape)
print(X_set2_cv.shape, y_cv.shape)
```

```
(35912, 12318) (35912,)
(26400, 12318) (26400,)
(17688, 12318) (17688,)
```

In [144]:

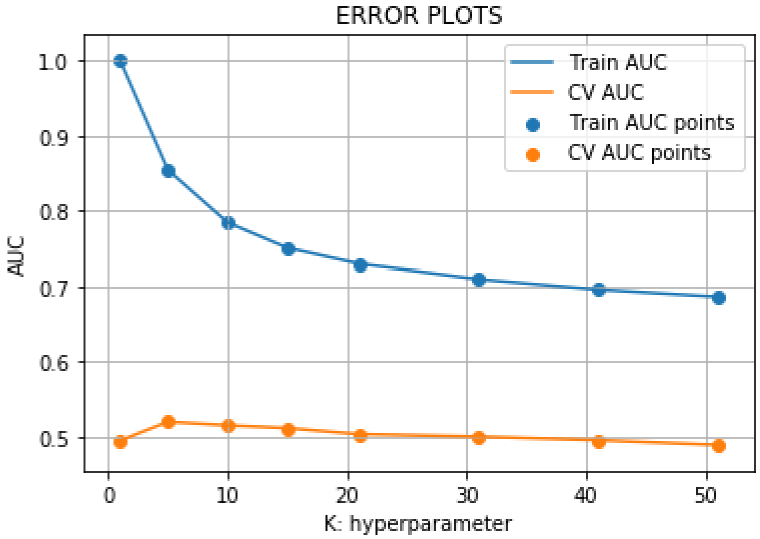
```
#https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html
import warnings
warnings.filterwarnings("ignore")
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif,chi2
#ValueError: Input X must be non-negative.

# not use chi because of error
##https://stackoverflow.com/questions/25792012/feature-selection-using-scikit-learn
X_train2_new = SelectKBest(f_classif, k=2000).fit_transform(X_set2_train, y_train)
X_test2_new = SelectKBest(f_classif, k=2000).fit_transform(X_set2_test, y_test)
X_cv2_new = SelectKBest(f_classif, k=2000).fit_transform(X_set2_cv, y_cv)
```

In [145]:

```
#train_essay_tfidf_w2v_vectors
#test_essay_tfidf_w2v_vectors
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute')
    neigh.fit(X_train2_new, 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 = neigh.predict_proba(X_train2_new)[:,1]#Return probability estimates for the  set3x ,for the c
    y_cv_pred =  neigh.predict_proba(X_cv2_new)[:,1]#Return probability estimates for the set3cvx,for the class


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

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

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

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

```
  0%|                                                                                | 0/8 [00:00
<?, ?it/s]
 12%|██████████▌                                                                    | 1/8 [03:49<26:48, 22
9.77s/it]
```

```
 25%|████████████████████▋                                                              | 2/8 [07:59<23:34, 23
5.75s/it]
 38%|███████████████████████████████▏                                                   | 3/8 [12:09<19:59, 23
9.92s/it]
 50%|██████████████████████████████████████████                                        | 4/8 [16:24<16:18, 24
4.55s/it]
 62%|████████████████████████████████████████████████████▊                              | 5/8 [20:41<12:25, 24
8.34s/it]
 75%|███████████████████████████████████████████████████████████████▍                   | 6/8 [24:58<08:21, 25
0.74s/it]
 88%|█████████████████████████████████████████████████████████████████████████▊         | 7/8 [29:15<04:12, 25
2.84s/it]
100%|████████████████████████████████████████████████████████████████████████████████████| 8/8 [33:32<00:00, 25
3.98s/it]
```



In [146]:

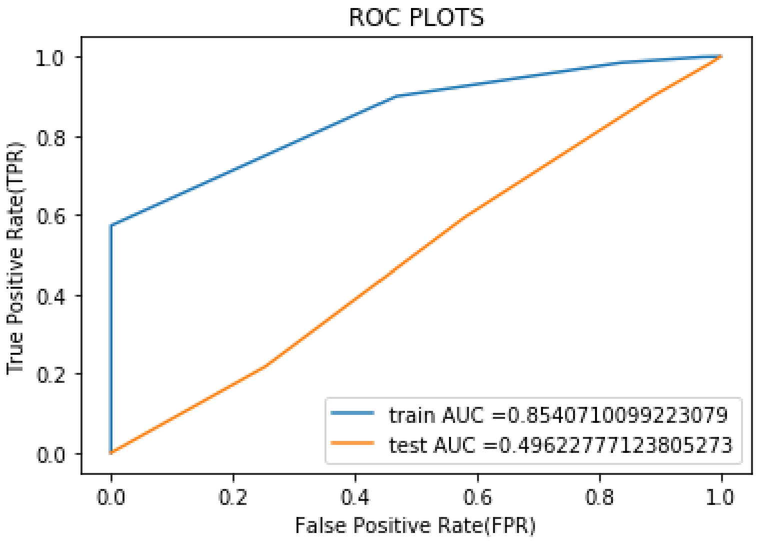
```
sc1 = [x for x in cv_auc]
opt_t_cv_4 = K[sc1.index(max(sc1 ))]
print("Maximum AUC score of cv is:" + ' ' + str(max(sc )))
print("Corresponding k value of cv is:",opt_t_cv_4, '\n')
best_k=opt_t_cv_4
print(best_k)
```

```
Maximum AUC score of cv is: 0.5944314816476711
Corresponding k value of cv is: 5

5
```

Fitting Model to Hyper-Parameter Curve: (Using bruteforce KNN)

In [147]:

```
# 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=5,algorithm='brute')
neigh.fit(X_train2_new ,y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train2_new)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test2_new)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("ROC PLOTS")
plt.show()
```

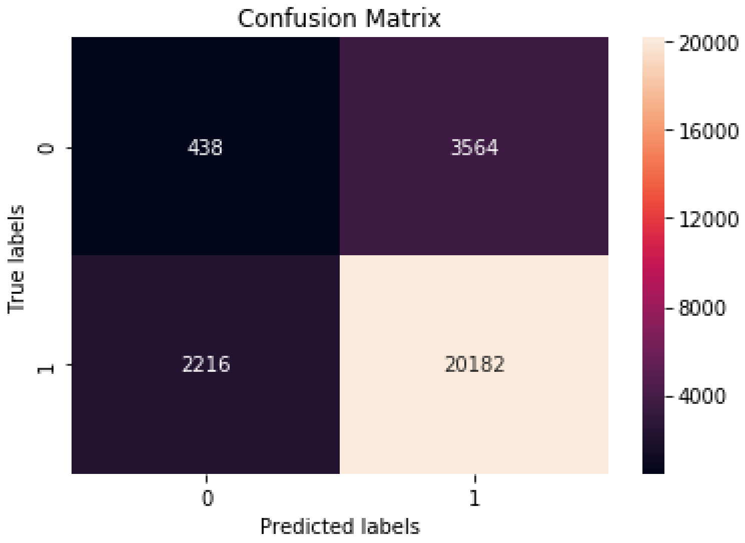


Observations: Finding the top 2000 featues not helpful,their are lots of reasons of it

1. In cv data because of less data or highly imbalance their is underfitting so k=1 is best, meaning totlay random roc curve

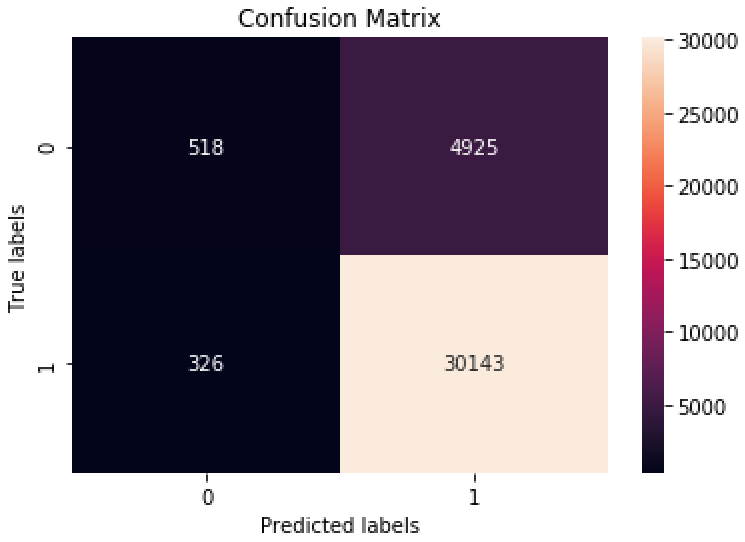
Confusion matrix

```
In [148]: import seaborn as sns
          import matplotlib.pyplot as plt

          ax= plt.subplot()

          def predict(proba,threshold,fpr,tpr):
              t=threshold[np.argmax(fpr*(1-tpr))]
              print("the maximun value of tpr*(1-fpr)",np.round(max(tpr*(1-fpr)),2) ,"for threshold",np.round(t,2))
              predictions = []
              for i in proba:
                  if i>=t:
                      predictions.append(1)
                  else:
                      predictions.append(0)
              return predictions


          sns.heatmap(confusion_matrix(y_test, neigh.predict(X_test2_new )), annot=True, ax = ax,fmt='g');
          ax.set_xlabel('Predicted labels');
          ax.set_ylabel('True labels');
          ax.set_title('Confusion Matrix');
```



In [168]:

```
import seaborn as sns
import matplotlib.pyplot as plt

ax= plt.subplot()

def predict(proba,threshold,fpr,tpr):
    t=threshold[np.argmax(fpr*(1-tpr))]
    print("the maximun value of tpr*(1-fpr)",np.round(max(tpr*(1-fpr)),2) ,"for threshold",np.round(t,2))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions


sns.heatmap(confusion_matrix(y_train, neigh.predict(X_train2_new )), annot=True, ax = ax,fmt='g');
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```



Observatoins: In train data as our best k iis one thats why fully pefcet train_data, but totaly overfitting this is.

3. Conclusions

In [154]:

```
# Please compare all your models using Prettytable library
# Please compare all your models using Prettytable library
#how to use pretty table http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

tb = PrettyTable()
tb.field_names= ("Vectorizer", "Model","HyperParameter" ,"AUC")
tb.add_row(["BOW", "Auto", 32, 71])
tb.add_row(["Tf-Idf", "Auto", 60, 84])
tb.add_row(["AVG-W2v", "Auto", 13, 75])
tb.add_row(["Tf-Idf W2v", "Auto", 8, 81])
tb.add_row(["Tf-Idf KBest", "Auto", 5, 85])
print(tb.get_string(titles = "KNN - Observations"))
#print(tb)
```

```
+--------------+-------+----------------+-----+
|  Vectorizer  | Model | HyperParameter | AUC |
+--------------+-------+----------------+-----+
|     BOW      |  Auto |       32       |  71 |
|    Tf-Idf    |  Auto |       60       |  84 |
|   AVG-W2v    |  Auto |       13       |  75 |
|  Tf-Idf W2v  |  Auto |       8        |  81 |
| Tf-Idf KBest |  Auto |       5        |  85 |
+--------------+-------+----------------+-----+
```

Performance of Model: So as we see from all our models, there are less true positives, and more true negatives. Simply its because of the k value. Also from AUC values, we can say that the model TF-Idf KBest is the best model, because it has the highest AUC value (0.85).