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
```

In [2]:

```
dft = pd.read_csv('train_data.csv')
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 (109248, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix'
'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]:

```
dft.columns
```

Out[4]:

```
Index(['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_approve
d'],
      dtype='object')
```

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/49702492/40840
39
dft['Date'] = pd.to_datetime(dft['project_submitted_datetime'])
dft.drop('project_submitted_datetime', axis=1, inplace=True)
dft.sort_values(by=['Date'], inplace=True)


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


dft.head(2)
```

Out[5]:

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

text processing

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)
dft.head(2)
```

Out[6]:

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

In [7]:

```
# 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 [8]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', '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"]
```

Preprocessing of project_subject_categories

In [9]:

```
catogories = list(dft['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47
301924/4084039

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

In [10]:

```
sub_catogories = list(dft['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47
301924/4084039

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

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", "Warmt
h", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "M
ath & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace
 it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"M
ath & Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spa
ces
        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]))
```

In [11]:

```
#Preprocessing of project_grade_category
print(dft['project_grade_category'][:20])
```

```
55660     Grades PreK-2
76127        Grades 3-5
51140     Grades PreK-2
473       Grades PreK-2
41558        Grades 3-5
29891        Grades 3-5
81565        Grades 3-5
79026        Grades 3-5
23374     Grades PreK-2
86551        Grades 3-5
49228     Grades PreK-2
72638       Grades 9-12
7176      Grades PreK-2
70898        Grades 3-5
102755       Grades 3-5
72593     Grades PreK-2
35006        Grades 3-5
100222       Grades 3-5
5145         Grades 3-5
48237       Grades 9-12
Name: project_grade_category, dtype: object
```

In [12]:

```
d= list(dft['project_grade_category'].values)
# remove special characters from list of strings 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]))
```

Test - Train Split

In [13]:

```
# train test split

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(dft, dft['project_is_approved'], te
st_size=0.33, stratify = dft['project_is_approved'])
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, strat
ify=y_train)
```

In [14]:

```
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, stratify= y_train,tes
t_size = 0.33)
print(y_train.value_counts())
print(y_test.value_counts())
print(y_cv.value_counts())
```

```
1    27882
0     4975
Name: project_is_approved, dtype: int64
1    30593
0     5459
Name: project_is_approved, dtype: int64
1    13733
0     2451
Name: project_is_approved, dtype: int64
```

In [15]:

```
X_train.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)
```

Text preprocessing of train,test and cv

In [16]:

```
# Combining all the above

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

```
100%|████████████████████████████████████████████████████████████████████
█████| 32857/32857 [00:18<00:00, 1803.09it/s]
```

In [17]:

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

```
100%|█████████████████████████████████████████████████████████████████████
█████| 36052/36052 [00:21<00:00, 1700.25it/s]
```

In [18]:

```
from tqdm import tqdm
preprocessed_essays_cv = []
# 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)
    preprocessed_essays_cv.append(sent.lower().strip())
```

```
100%|█████████████████████████████████████████████████████████████████████
█████| 16184/16184 [00:08<00:00, 1861.15it/s]
```

In [19]:

```
preprocessed_titles_train = []

for titles in tqdm(X_train["project_title"]):
    title = decontracted(titles)
    title = title.replace('\\r', ' ')
    title = title.replace('\\"', ' ')
    title = title.replace('\\n', ' ')
    title = re.sub('[^A-Za-z0-9]+', ' ', title)
    title = ' '.join(f for f in title.split() if f not in stopwords)
    preprocessed_titles_train.append(title.lower().strip())
```

```
100%|█████████████████████████████████████████████████████████████████████
████| 32857/32857 [00:00<00:00, 42503.58it/s]
```

In [20]:

```
preprocessed_titles_test = []

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

```
100%|██████████████████████████████████████████████████████████████████
████| 36052/36052 [00:00<00:00, 40551.62it/s]
```

In [21]:

```
preprocessed_titles_cv = []

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

```
100%|██████████████████████████████████████████████████████████████████
████| 16184/16184 [00:00<00:00, 37908.68it/s]
```

One Hot Encode - Clean Categories of Projects

In [22]:

```
# 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
vectorizer1 = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False,
binary=True)
vectorizer1.fit(X_train['clean_categories'].values)
# firstly convert fit the train data into the vectoriaer then it learn hte vocablery
# we use the fitted CountVectorizer to convert the text to vector
X_train_cat = vectorizer1.transform(X_train['clean_categories'].values)
X_cv_cat = vectorizer1.transform(X_cv['clean_categories'].values)
X_test_cat = vectorizer1.transform(X_test['clean_categories'].values)
print(vectorizer1.get_feature_names())
```

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

In [23]:

```
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
(32857, 9) (32857,)
(16184, 9) (16184,)
(36052, 9) (36052,)
==========================================================================
==========================
```

In [24]:

```
# 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
vectorizer2 = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=Fa
lse, binary
=True)
vectorizer2.fit(X_train['clean_subcategories'].values)
# firstly convert fit the train data into the vectoriaer then it learn hte vocablery
# we use the fitted CountVectorizer to convert the text to vector
X_train_subcat = vectorizer2.transform(X_train['clean_subcategories'].values)
X_cv_subcat = vectorizer2.transform(X_cv['clean_subcategories'].values)
X_test_subcat = vectorizer2.transform(X_test['clean_subcategories'].values)
print(vectorizer2.get_feature_names())
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvemen
t', 'Extracurricular', 'Civics_Government', 'ForeignLanguages', 'Nutrition
Education', 'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingArts',
'CharacterEducation', 'TeamSports', 'Other', 'College_CareerPrep', 'Musi
c', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL',
'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'A
ppliedSciences', 'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Lit
eracy']
```

In [25]:

```
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
(32857, 30) (32857,)
(16184, 30) (16184,)
(36052, 30) (36052,)
==========================================================================
==========================
```

In [26]:

```
#first convert to dict
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
]))# sor it
print(sorted_school_state_dict)
# 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
vectorizer3 = CountVectorizer(vocabulary=list(sorted_school_state_dict.keys()), lowerca
se=False, binary=True)
vectorizer3.fit(dft['school_state'].values)
# firstly convert fit the train data into the vectoriaer then it learn hte vocablery
# we use the fitted CountVectorizer to convert the text to vector
X_train_school_state = vectorizer3.transform(X_train['school_state'].values)
X_cv_school_state = vectorizer3.transform(X_cv['school_state'].values)
X_test_school_state = vectorizer3.transform(X_test['school_state'].values)
print(vectorizer3.get_feature_names())
```

```
{'VT': 80, 'WY': 98, 'ND': 143, 'MT': 245, 'RI': 285, 'SD': 300, 'NE': 30
9, 'DE': 343, 'AK': 345, 'NH': 348, 'WV': 503, 'ME': 505, 'HI': 507, 'DC':
516, 'NM': 557, 'KS': 634, 'IA': 666, 'ID': 693, 'AR': 1049, 'CO': 1111,
'MN': 1208, 'OR': 1242, 'KY': 1304, 'MS': 1323, 'NV': 1367, 'MD': 1514, 'C
T': 1663, 'TN': 1688, 'UT': 1731, 'AL': 1762, 'WI': 1827, 'VA': 2045, 'A
Z': 2147, 'NJ': 2237, 'OK': 2276, 'WA': 2334, 'MA': 2389, 'LA': 2394, 'O
H': 2467, 'MO': 2576, 'IN': 2620, 'PA': 3109, 'MI': 3161, 'SC': 3936, 'G
A': 3963, 'IL': 4350, 'NC': 5091, 'FL': 6185, 'NY': 7318, 'TX': 7396, 'C
A': 15388}
['VT', 'WY', 'ND', 'MT', 'RI', 'SD', 'NE', 'DE', 'AK', 'NH', 'WV', 'ME',
'HI', 'DC', 'NM', 'KS', 'IA', 'ID', 'AR', 'CO', 'MN', 'OR', 'KY', 'MS', 'N
V', 'MD', 'CT', 'TN', 'UT', 'AL', 'WI', 'VA', 'AZ', 'NJ', 'OK', 'WA', 'M
A', 'LA', 'OH', 'MO', 'IN', 'PA', 'MI', 'SC', 'GA', 'IL', 'NC', 'FL', 'N
Y', 'TX', 'CA']
```

In [27]:

```
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
(32857, 51) (32857,)
(16184, 51) (16184,)
(36052, 51) (36052,)
==========================================================================
==========================
```

In [28]:

```
# 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
vectorizer4 = CountVectorizer(vocabulary=list(sorted_project_grade_category_dict.keys
()),
lowercase=False, binary=True)
vectorizer4.fit(dft['clean_grade'].values)
# firstly convert fit the train data into the vectoriaer then it learn hte vocablery
# we use the fitted CountVectorizer to convert the text to vector
X_train_project_grade_category = vectorizer4.transform(X_train['clean_grade'].values)
X_cv_project_grade_category = vectorizer4.transform(X_cv['clean_grade'].values)
X_test_project_grade_category = vectorizer4.transform(X_test['clean_grade'].values)
print(vectorizer4.get_feature_names())
```

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

In [29]:

```
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
(32857, 4) (32857,)
(16184, 4) (16184,)
(36052, 4) (36052,)
==========================================================================
==========================
```

In [30]:

```
#https://stackoverflow.com/questions/42224700/attributeerror-float-object-has-no-attrib
ute-split
dft['teacher_prefix']=dft['teacher_prefix'].fillna(" ")# filll the null values with spa
ce
X_train['teacher_prefix'][:3]# dots is the problme for us
```

Out[30]:

```
84044      Ms.
50446      Ms.
101952    Mrs.
Name: teacher_prefix, dtype: object
```

In [31]:

```
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
]))
# 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
vectorizer5 = CountVectorizer(vocabulary=list(sorted_teacher_prefix_dict.keys()), lower
case=False,binary=True)
vectorizer5.fit(dft['teacher_prefix'].values.astype('U'))
# firstly convert fit the train data into the vectoriaer then it learn hte vocablery
# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_prefix = vectorizer5.transform(X_train['teacher_prefix'].values.astype(
'U'))
X_cv_teacher_prefix= vectorizer5.transform(X_cv['teacher_prefix'].values.astype('U'))
X_test_teacher_prefix = vectorizer5.transform(X_test['teacher_prefix'].values.astype(
'U'))
print(vectorizer5.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 trainfor
m
```

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

In [32]:

```
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
(32857, 5) (32857,)
(16184, 5) (16184,)
(36052, 5) (36052,)
==========================================================================
==========================
```

BOW

In [33]:

```
X_train_essay=preprocessed_essays_train
X_cv_essay=preprocessed_essays_cv
X_test_essay=preprocessed_essays_test

X_train_title=preprocessed_titles_train
X_cv_title=preprocessed_titles_cv
X_test_title=preprocessed_titles_test
# We are considering only the words which appeared in at least 10 documents(rows or pro
jects).
vectorizer6 = CountVectorizer(min_df=10)# its a countvectors used for convert text to v
ectors
vectorizer6.fit(X_train_essay)# that is learned from trainned data
# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer6.transform(X_train_essay)
X_cv_bow = vectorizer6.transform(X_cv_essay)
X_test_bow = vectorizer6.transform(X_test_essay)
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 alll are the same by using first fit and then transform
```

```
After vectorizations
(32857, 10246) (32857,)
(16184, 10246) (16184,)
(36052, 10246) (36052,)
==========================================================================
==========================
```

In [34]:

```
#title
vectorizer7 = CountVectorizer(min_df=10)
vectorizer7.fit(X_train_title)# that is learned from trainned data

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow_title = vectorizer7.transform(X_train_title)
X_cv_bow_title= vectorizer7.transform(X_cv_title)
X_test_bow_title = vectorizer7.transform(X_test_title)

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 alll are the same by using first fit and then transform
```

```
After vectorizations
(32857, 1599) (32857,)
(16184, 1599) (16184,)
(36052, 1599) (36052,)
==========================================================================
==========================
```

In [35]:

```
#tfidf titles
from sklearn.feature_extraction.text import TfidfVectorizer

# We are considering only the words which appeared in at least 10 documents(rows or pro
jects).
vectorizer8 = TfidfVectorizer(min_df=10)# its a countvectors used for convert text to v
ectors
vectorizer8.fit(X_train_title)# that is learned from trainned data

# we use the fitted CountVectorizer to convert the text to vector
X_train_tf_title = vectorizer8.transform(X_train_title)
X_cv_tf_title= vectorizer8.transform(X_cv_title)
X_test_tf_title = vectorizer8.transform(X_test_title)

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)
```

```
After vectorizations
(32857, 1599) (32857,)
(16184, 1599) (16184,)
(36052, 1599) (36052,)
==========================================================================
==========================
```

In [36]:

```
#tfidf essay
from sklearn.feature_extraction.text import TfidfVectorizer
# We are considering only the words which appeared in at least 10 documents(rows or pro
jects).
vectorizer9 = TfidfVectorizer(min_df=10)# its a countvectors used for convert text to v
ectors
vectorizer9.fit(X_train_essay)# that is learned from trainned data

# we use the fitted CountVectorizer to convert the text to vector
X_train_tf_essay = vectorizer9.transform(X_train_essay)
X_cv_tf_essay= vectorizer9.transform(X_cv_essay)
X_test_tf_essay = vectorizer9.transform(X_test_essay)

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)
```

```
After vectorizations
(32857, 10246) (32857,)
(16184, 10246) (16184,)
(36052, 10246) (36052,)
==========================================================================
==========================
```

Vectorizing Numerical features

In [37]:

```
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))
# we also have to do this in train,test and cv
# 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
```

Standardization

In [38]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikitlearn.org/stable/modules/generated/sklearn.pre
processing.StandardScaler.html
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn import preprocessing
price_scalar = MinMaxScaler()
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
# Now standardize the data with above maen and variance.
test_price_standar = price_scalar.transform(X_test['price'].values.reshape(-1, 1))
test_price_standar
# Now standardize the data with above maen and variance.
cv_price_standar = price_scalar.transform(X_cv['price'].values.reshape(-1, 1))
test_price_standar
```

Out[38]:

```
array([[0.01823308],
       [0.01015772],
       [0.01193202],
       ...,
       [0.04173805],
       [0.02975103],
       [0.01393235]])
```

In [39]:

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

```
(32857, 1) (32857,)
(36052, 1) (36052,)
(16184, 1) (16184,)
```

Standardized Previous_year_tecaher_projects train,test and cv

In [40]:

```
# previous_year_projects
price_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].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_prev_proj_standar = price_scalar.transform(X_train['teacher_number_of_previously_
posted_projects'].values.reshape(-1,1))
train_prev_proj_standar
# Now standardize the data with above maen and variance.
test_prev_proj_standar = price_scalar.transform(X_test['teacher_number_of_previously_po
sted_projects'].values.reshape(-1, 1))
test_prev_proj_standar
# 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.reshape(-1, 1))
cv_prev_proj_standar
```

Out[40]:

```
array([[0.        ],
       [0.00692841],
       [0.01385681],
       ...,
       [0.02309469],
       [0.02078522],
       [0.02078522]])
```

In [41]:

```
# shapes
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)
```

```
(32857, 1) (32857,)
(36052, 1) (36052,)
(16184, 1) (16184,)
```

standardize the Quantity column of the train,test and cv

In [42]:

```
price_scalar.fit(X_train['quantity'].values.reshape(-1,1)) # finding the mean and stand
ard 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_qnty_standar = price_scalar.transform(X_train['quantity'].values.reshape(-1, 1))
train_qnty_standar
# Now standardize the data with above maen and variance.
cv_qnty_standar = price_scalar.transform(X_cv['quantity'].values.reshape(-1, 1))
cv_qnty_standar
# Now standardize the data with above maen and variance.
test_qnty_standar = price_scalar.transform(X_test['quantity'].values.reshape(-1, 1))
test_qnty_standar
```

Out[42]:

```
array([[0.00161031],
       [0.00805153],
       [0.00322061],
       ...,
       [0.00161031],
       [0.00966184],
       [0.        ]])
```

In [43]:

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

```
(32857, 1) (32857,)
(36052, 1) (36052,)
(16184, 1) (16184,)
```

In [44]:

```
#Merge all features
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,
                       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)
```

```
(32857, 11944) (32857,)
```

In [45]:

```
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,
                    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)
```

```
(16184, 11944) (16184,)
```

In [46]:

```
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,
                      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)
```

```
(36052, 11944) (36052,)
```

In [47]:

```
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,
                       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)
```

```
(32857, 11944) (32857,)
```

In [48]:

```
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,
                    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)
```

```
(16184, 11944) (16184,)
```

In [49]:

```
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,
                      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)
```

```
(36052, 11944) (36052,)
```

Applying Naive Bayes(MultinomialNB) on BOW

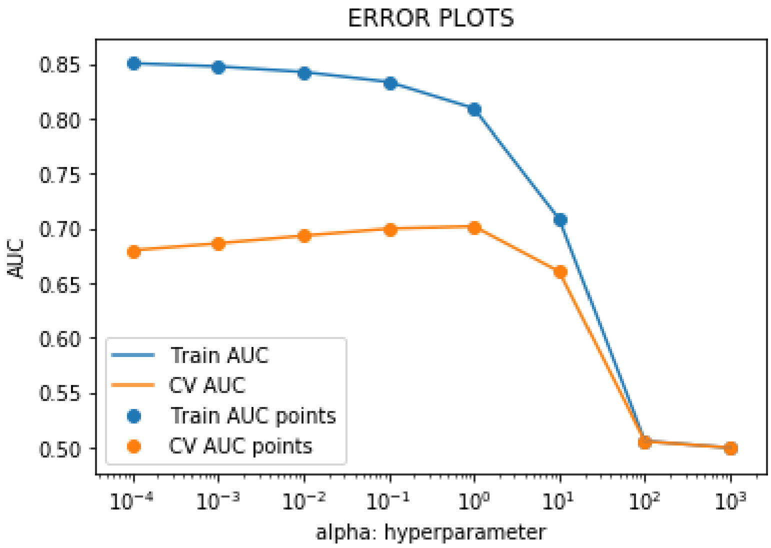
In [62]:

```
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
import math
train_auc = []
cv_auc = []
alpha =[0.0001,0.001,0.01,0.1,1,10,100,1000]


for i  in   tqdm(alpha):


    neigh = MultinomialNB(alpha=i)# takes the alpha 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 t
he positive class
# not the predicted outputs
    y_train_pred = neigh.predict_proba(X_set1_train)[:,1]#Return probability estimates
 for the set1x ,for the class label 1 or +ve.
    y_cv_pred = neigh.predict_proba(X_set1_cv)[:,1]#Return probability estimates for th
e setcvx,for the class label 1 or +ve .
# roc curve
#Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from predicti
on 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(alpha, train_auc, label='Train AUC')
plt.plot(alpha, cv_auc, label='CV AUC')
plt.xscale('log')# we take the log in the x axis
plt.scatter(alpha, train_auc, label='Train AUC points')
plt.scatter(alpha, cv_auc, label='CV AUC points')
plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
100%|██████████████████████████████████████████████████████████████████████████████████████████| 8/8 [00:01<00:00,  5.44it/s]
```



In [63]:

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

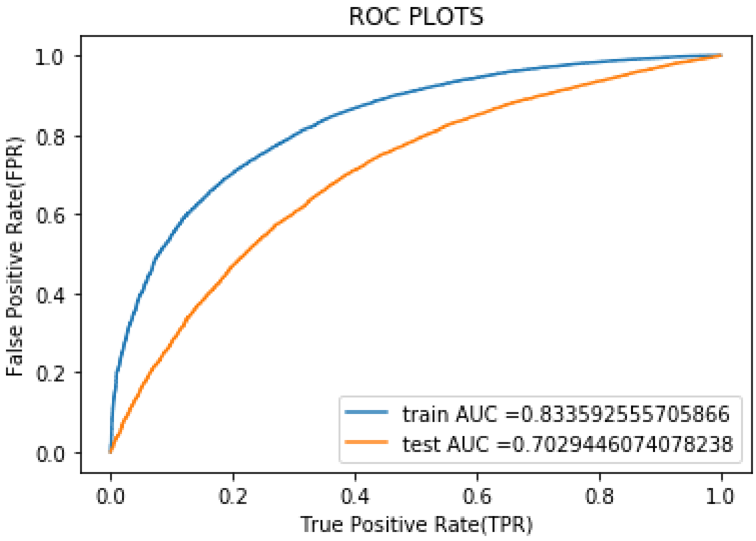
```
Maximum AUC score of cv is: 0.7020197041656755
Corresponding alpha value of cv is: 1

1
```

Fitting Model to Hyper-Parameter Curve

In [65]:

```
# https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklea
rn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
neigh = MultinomialNB(alpha=0.1)
neigh.fit(X_set1_train ,y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he 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()
print("="*100)
```



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

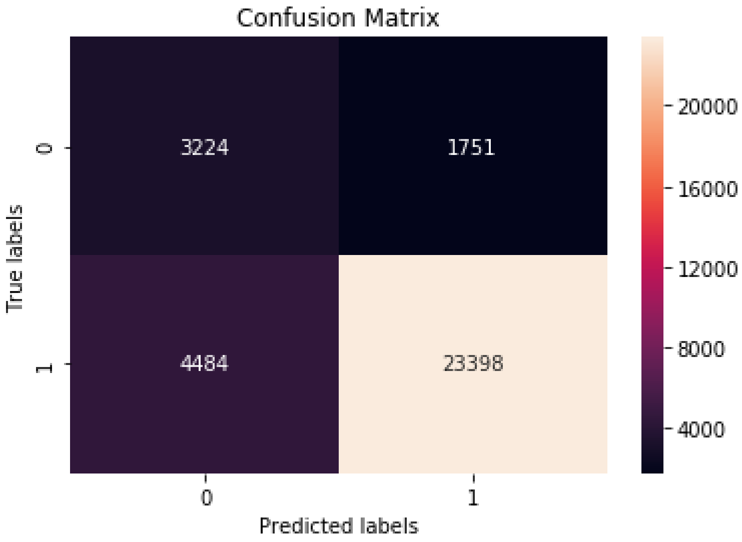
OBSERVATIONS: As we seen form the roc plot ,Model work good on the train data , also model works good on the test data, only a little bit overfitting

Confusion matrix :

In [66]:

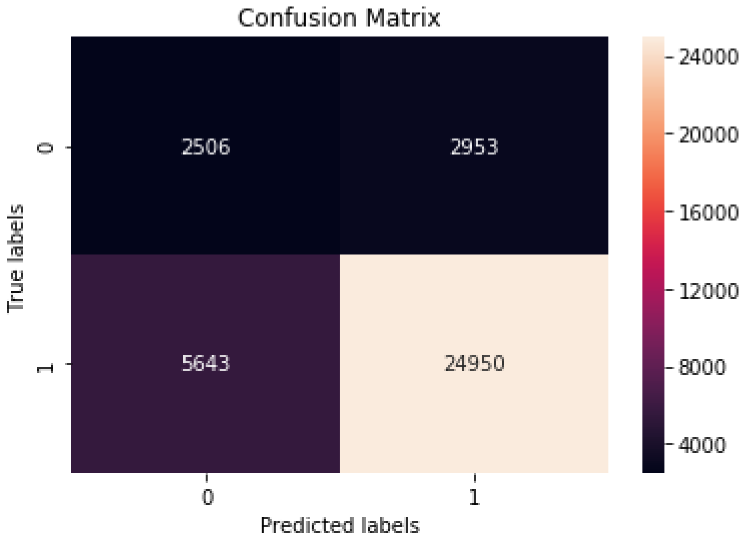
```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
import matplotlib.pyplot as plt
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_train, neigh.predict(X_set1_train )), annot=True, ax = a
x,fmt='g');

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```



In [67]:

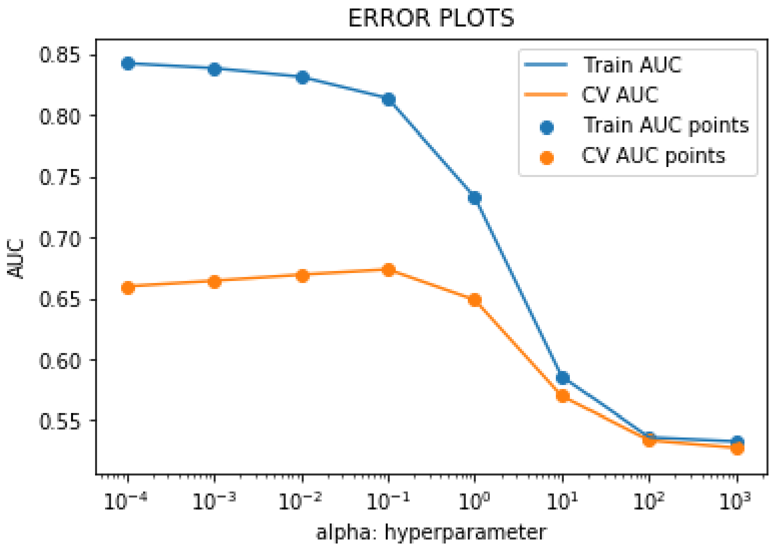
```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
import matplotlib.pyplot as plt
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, neigh.predict(X_set1_test )), annot=True, ax = ax,
fmt='g');
# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```



In [68]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.
y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence va
lues, or no
n-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.
"""
train_auc = []
cv_auc = []
alpha = [0.0001,0.001,0.01,0.1,1,10,100,1000]
for i in tqdm(alpha):
    neigh = MultinomialNB(alpha=i)# 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 t
he positive class
# not the predicted outputs
    y_train_pred = neigh.predict_proba(X_set2_train)[:,1]#Return probability estimates
 for the set1x ,for the class label 1 or +ve.
    y_cv_pred = neigh.predict_proba(X_set2_cv)[:,1]#Return probability estimates for th
e setcvx,for the class label 1 or +ve .
# roc curve
#Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from predicti
on 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(alpha, train_auc, label='Train AUC')
plt.plot(alpha, cv_auc, label='CV AUC')
plt.scatter(alpha, train_auc, label='Train AUC points')
plt.scatter(alpha, cv_auc, label='CV AUC points')
plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
100%|████████████████████████████████████████████████████████████
█████████████| 8/8 [00:02<00:00,  2.78it/s]
```

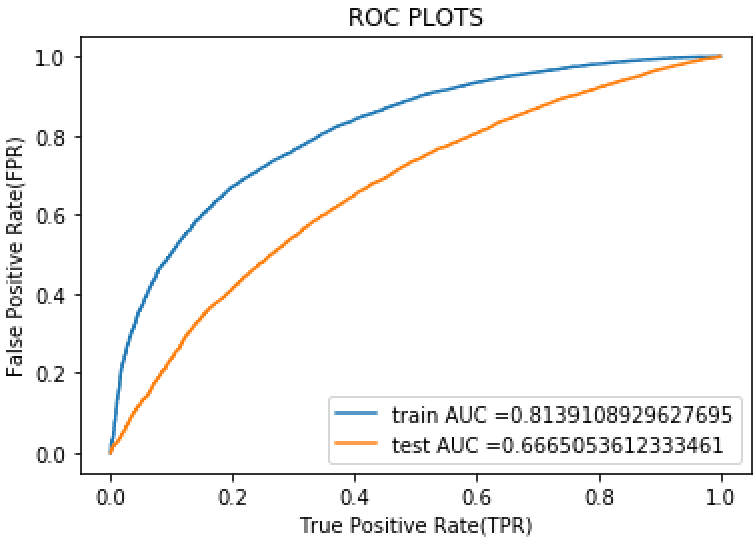


Fitting Model to Hyper-Parameter Curve:

In [69]:

```
# https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklea
rn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
neigh = MultinomialNB(alpha=0.1)
neigh.fit(X_set2_train ,y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he 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.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("ROC PLOTS")
plt.show()
print("="*100)
```



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

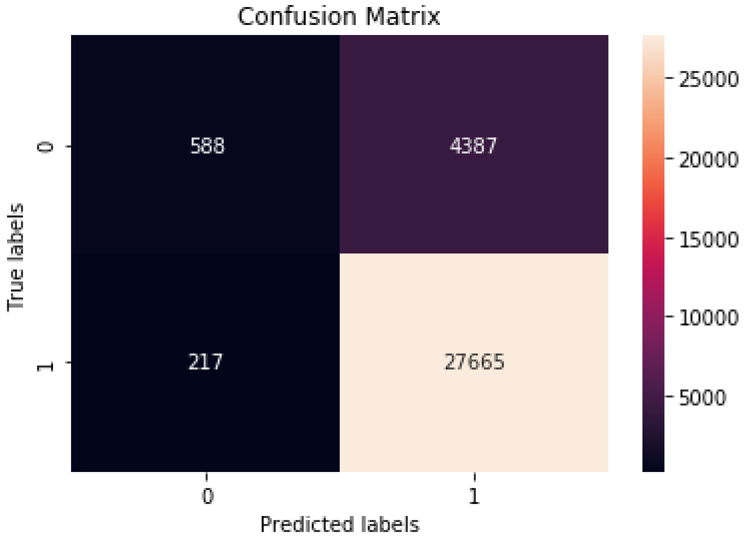
OBSERVATIONS: As we seen form the roc plot , only a little bit overfitting, but roc curve are not so good only 65 score.

Confusion matrix

In [70]:

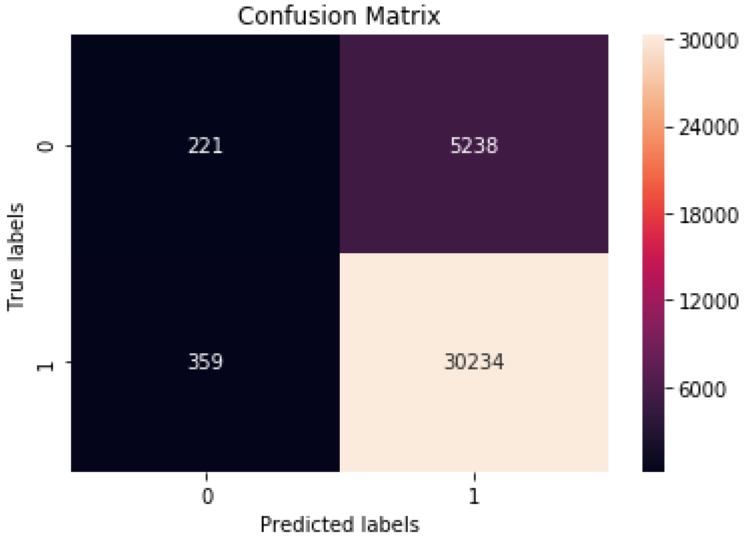
```
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_train, neigh.predict(X_set2_train )), annot=True, ax = a
x,fmt='g');

ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```



In [71]:

```
ax= plt.subplot()
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');
```



OBSERVATOINS: As we see from this confusion matrix ,True negatives are very less in this case because also in the original data it is very less , so bcz of this imbalance this work not good, dominating the negatives.

Top 10 features (negatives and positives)

In [80]:

```
#for BOW
nb = MultinomialNB(alpha=0.1)# takes the k from the i th list value
nb.fit(X_set1_train, y_train)# fit the model

# now make a dictionary of all the probabilities fo the weights
bow_features_probs = []
for a in range(11944):
    bow_features_probs.append(nb.feature_log_prob_[0,a] )

print(len(bow_features_probs))

bow_features_names = []
for a in vectorizer1.get_feature_names() :# clean categories
    bow_features_names.append(a)
for a in vectorizer2.get_feature_names() :# sub categoreis
    bow_features_names.append(a)
for a in vectorizer3.get_feature_names() :#schooll state
    bow_features_names.append(a)
for a in vectorizer4.get_feature_names() :# grade categoreis
    bow_features_names.append(a)
for a in vectorizer5.get_feature_names() :# teacher prefix
    bow_features_names.append(a)
for a in vectorizer6.get_feature_names(): #titles bow
    bow_features_names.append(a)
for a in vectorizer7.get_feature_names(): # essays bow
    bow_features_names.append(a)
print( len(bow_features_names))
```

```
11944
11944
```

In [81]:

```
#top 10 negatives
final_bow_features = pd.DataFrame({'feature_prob_estimates' : bow_features_probs, 'feat
ure_names': bow_features_names})
a =final_bow_features.sort_values(by = ['feature_prob_estimates'], ascending = False)
#print(final_bow_features.head(6))
a.head(10)
```

Out[81]:

	feature_prob_estimates	feature_names
10467	-2.988060	bay
9635	-4.090900	trusted
6924	-4.418089	planting
3313	-4.553203	enlarged
6920	-4.746311	plans
7813	-4.757874	resource
5986	-4.793209	minutes
7672	-4.966286	relatives
7246	-5.014401	programming
7711	-5.090380	remediate

In [87]:

```
#top 10 Positives
# now make a dictionary of all the probabilityies fo the weights
bow_features_probs_pos = []
for a in range(11944):
    bow_features_probs_pos.append(nb.feature_log_prob_[1,a] )# negative feature probabilities
#len(bow_features_probs)
final_bow_features = pd.DataFrame({'feature_prob_estimates_pos' : bow_features_probs_pos,'feature_names' : bow_features_names})
a =final_bow_features.sort_values(by = ['feature_prob_estimates_pos'], ascending = False)
a.head(10)
```

Out[87]:

	feature_prob_estimates_pos	feature_names
10467	-2.984085	bay
9635	-4.131952	trusted
6924	-4.489194	planting
3313	-4.512943	enlarged
7813	-4.780662	resource
6920	-4.831161	plans
5986	-4.857289	minutes
7246	-4.997282	programming
7672	-5.018591	relatives
7711	-5.130304	remediate

In [89]:

```
nb = MultinomialNB(alpha=1)# takes the k from the i th list value
nb.fit(X_set2_train, y_train)# fit the model
# now make a dictionary of all the probabilityies fo the weights
tf_features_probs = []
for a in range(11944):# loop till (shape of data)
 tf_features_probs.append(nb.feature_log_prob_[0,a] )# negative feature probabilities
#len(bow_features_probs)
 tf_features_names = []
for a in vectorizer1.get_feature_names() :# clean categories
 tf_features_names.append(a)
for a in vectorizer2.get_feature_names() :# sub categoreis
 tf_features_names.append(a)
for a in vectorizer3.get_feature_names() :#schooll state
 tf_features_names.append(a)
for a in vectorizer4.get_feature_names() :# grade categoreis
 tf_features_names.append(a)
for a in vectorizer5.get_feature_names() :# teacher prefix
 tf_features_names.append(a)
len(tf_features_names)

for a in vectorizer8.get_feature_names(): #titles tf_idf
 tf_features_names.append(a)
for a in vectorizer9.get_feature_names(): # essays tf_idf
 tf_features_names.append(a)


# top 10 negatives
final_tf_features = pd.DataFrame({'feature_prob_estimates' : tf_features_probs, 'feature_names' :tf_features_names})
a =final_tf_features.sort_values(by = ['feature_prob_estimates'], ascending = False)
#print(final_bow_features.head(6))
a.head(10)
```

Out[89]:

	feature_prob_estimates	feature_names
11858	-3.680449	workstation
11857	-3.759293	workspaces
11888	-4.189996	writers
11887	-4.196406	writer
11886	-4.455560	write
11885	-4.797464	wristbands
11855	-4.797464	workshops
8868	-4.827713	promising
11856	-4.839386	workspace
11943	-4.862589	zumba

In [91]:

```
#top 10 Positives
# now make a dictionary of all the probabilityies fo the weights
bow_features_probs_pos = []
for a in range(11944):
 bow_features_probs_pos.append(nb.feature_log_prob_[1,a] )# negative feature probabilities
#len(bow_features_probs)
final_bow_features = pd.DataFrame({'feature_prob_estimates_pos' : bow_features_probs_pos,'feature_names' : bow_features_names})
a =final_bow_features.sort_values(by = ['feature_prob_estimates_pos'], ascending = False)
a.head(10)
```

Out[91]:

	feature_prob_estimates_pos	feature_names
11858	-3.419909	visuals
11857	-3.668446	visual
11888	-3.850239	whiteboards
11887	-4.046308	whiteboard
11886	-4.283589	white
11943	-4.655345	zone
8868	-4.715152	stimulation
11856	-4.738352	virtual
11855	-4.784321	view
11885	-4.784321	while

Conclusion

In [92]:

```
# 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",1, 70])
tb.add_row(["Tf-Idf", "Auto", 0.1, 67])
print(tb.get_string(titles = "KNN - Observations"))
#print(tb)
```

```
+------------+-------+----------------+-----+
| Vectorizer | Model | HyperParameter | AUC |
+------------+-------+----------------+-----+
|    BOW     |  Auto |       1        |  70 |
|   Tf-Idf   |  Auto |      0.1       |  67 |
+------------+-------+----------------+-----+
```