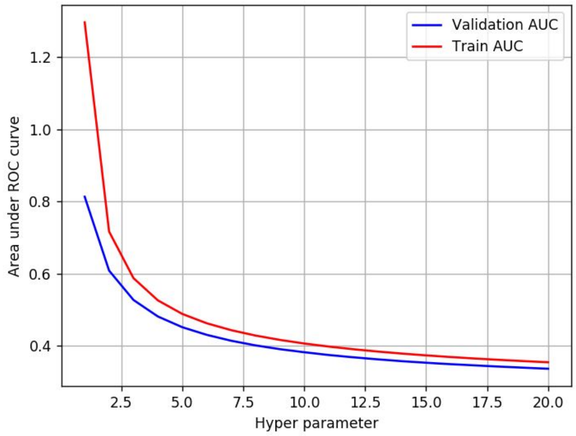
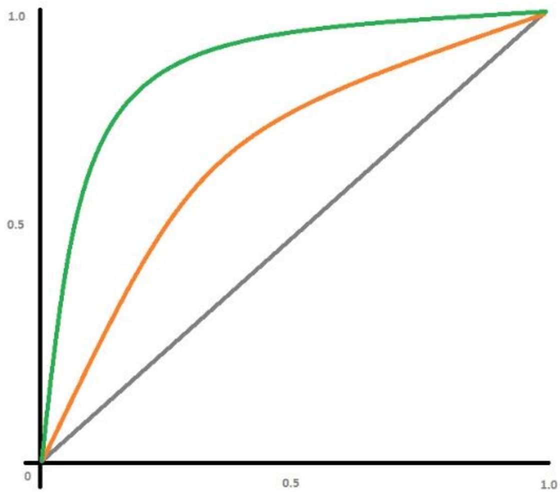
Assignment 6: Apply NB

1. **Apply Multinomial NB on these feature sets**
    - Set 1: categorical, numerical features + preprocessed_eassay (BOW)
    - Set 2: categorical, numerical features + preprocessed_eassay (TFIDF)
2. **The hyper paramter tuning(find best alpha:smoothing parameter)**
    - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
    - find the best hyper paramter using k-fold cross validation(use GridsearchCV or RandomsearchCV)/simple cross validation data (write for loop to iterate over hyper parameter values)
    -
3. **Representation of results**
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



    - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.



    - Along with plotting ROC curve, you need to print the confusion matrix (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points

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

4. fine the top 20 features from either from feature Set 1 or feature Set 2 using absolute values of \`feature_log_prob_ \` parameter of \`MultinomialNB\` (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html) and print their corresponding feature names
5. You need to summarize the results at the end of the notebook, summarize it in the table format

Vectorizer	Model	Hyper parameter	AUC
BOW	Brute	7	0.78
TFIDF	Brute	12	0.79

2. Naive Bayes

1.1 Loading 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 CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve,auc
import re

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]:

```
import pandas
data = pandas.read_csv('preprocessed_data.csv')
data.head(3)
```

Out[2]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_
0	ca	mrs	grades_prek_2	
1	ut	ms	grades_3_5	
2	ca	mrs	grades_prek_2	

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

In [3]:

```
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data, data['project_is_approved'],
test_size=0.33, stratify=data['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 [4]:

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

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)
```

```
1    41615
0     7426
Name: project_is_approved, dtype: int64
1    30593
0     5459
Name: project_is_approved, dtype: int64
1    20498
0     3657
Name: project_is_approved, dtype: int64
```

1.3 Make Data Model Ready: encoding eassay, and project_title

Applying Bow Essay

In [5]:

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

print("="*100)


vectorizer5 = CountVectorizer(min_df=10,ngram_range=(1,4)) #Considered words which appe
ared atleast 10 documents
vectorizer5.fit(X_train['essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_bowessay = vectorizer5.transform(X_train['essay'].values)
X_cv_bowessay = vectorizer5.transform(X_cv['essay'].values)
X_test_bowessay = vectorizer5.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_bowessay.shape, y_train.shape)
print(X_cv_bowessay.shape, y_cv.shape)
print(X_test_bowessay.shape, y_test.shape)
print("="*100)
#No of features is same for all train,test,cv
```

```
(49041, 8) (49041,)
(24155, 8) (24155,)
(36052, 8) (36052,)
==========================================================================
==========================
After vectorizations
(49041, 166493) (49041,)
(24155, 166493) (24155,)
(36052, 166493) (36052,)
==========================================================================
==========================
```

Applying TFIDF For Essay

In [15]:

```
# TFIDF Essay

vectorizer6 = TfidfVectorizer(min_df=10,ngram_range=(1,4))
vectorizer6.fit(X_train['essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_tfidfessay = vectorizer6.transform(X_train['essay'].values)
X_cv_tfidfessay = vectorizer6.transform(X_cv['essay'].values)
X_test_tfidfessay = vectorizer6.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_tfidfessay.shape, y_train.shape)
print(X_cv_tfidfessay.shape, y_cv.shape)
print(X_test_tfidfessay.shape, y_test.shape)
print("="*100)
#No of features is same for all train,test,cv
```

```
After vectorizations
(49041, 166493) (49041,)
(24155, 166493) (24155,)
(36052, 166493) (36052,)
==========================================================================
==========================
```

1.4 Make Data Model Ready: encoding numerical, categorical features

1.Clean_Categories convert category into vectors

In [7]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values)
# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_cat = vectorizer.transform(X_train['clean_categories'].values)
X_cv_clean_cat = vectorizer.transform(X_cv['clean_categories'].values)
X_test_clean_cat = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_clean_cat.shape, y_train.shape)
print(X_cv_clean_cat.shape, y_cv.shape)
print(X_test_clean_cat.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 9) (49041,)
(24155, 9) (24155,)
(36052, 9) (36052,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'lit
eracy_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
==========================================================================
==========================
```

2.clean_subcategories convert category into vector

In [8]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer

vectorizer1 = CountVectorizer()
vectorizer1.fit(X_train['clean_subcategories'].values)
# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_subcat = vectorizer1.transform(X_train['clean_subcategories'].values)
X_cv_clean_subcat = vectorizer1.transform(X_cv['clean_subcategories'].values)
X_test_clean_subcat = vectorizer1.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_clean_subcat.shape, y_train.shape)
print(X_cv_clean_subcat.shape, y_cv.shape)
print(X_test_clean_subcat.shape, y_test.shape)
print(vectorizer1.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 30) (49041,)
(24155, 30) (24155,)
(36052, 30) (36052,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_governmen
t', 'college_careerprep', 'communityservice', 'earlydevelopment', 'economi
cs', 'environmentalscience', 'esl', 'extracurricular', 'financialliterac
y', 'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_welln
ess', 'history_geography', 'literacy', 'literature_writing', 'mathematic
s', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performi
ngarts', 'socialsciences', 'specialneeds', 'teamsports', 'visualarts', 'wa
rmth']
==========================================================================
==========================
```

3.school_state convert categorical into vector

In [9]:

```
vectorizer2 = CountVectorizer()
vectorizer2.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer2.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer2.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer2.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer2.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 51) (49041,)
(24155, 51) (24155,)
(36052, 51) (36052,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi',
'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'm
o', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'o
k', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'w
i', 'wv', 'wy']
==========================================================================
==========================
```

4.project_grade_category convert categorical into vectors

In [10]:

```
vectorizer3 = CountVectorizer()
vectorizer3.fit(X_train['project_grade_category'].values) # fit has to happen only on t
rain data

# we use the fitted CountVectorizer to convert the text to vector
X_train_cleangrade = vectorizer3.transform(X_train['project_grade_category'].values)
X_cv_cleangrade = vectorizer3.transform(X_cv['project_grade_category'].values)
X_test_cleangrade = vectorizer3.transform(X_test['project_grade_category'].values)

print("After vectorizations")
print(X_train_cleangrade.shape, y_train.shape)
print(X_cv_cleangrade.shape, y_cv.shape)
print(X_test_cleangrade.shape, y_test.shape)
print(vectorizer3.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 4) (49041,)
(24155, 4) (24155,)
(36052, 4) (36052,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
==========================================================================
==========================
```

1. teacher_prefix convert categorical into vectors

In [11]:

```
vectorizer4 = CountVectorizer()
vectorizer4.fit(X_train['teacher_prefix'].values) # fit has to happen only on train dat
a

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher = vectorizer4.transform(X_train['teacher_prefix'].values)
X_cv_teacher = vectorizer4.transform(X_cv['teacher_prefix'].values)
X_test_teacher = vectorizer4.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher.shape, y_train.shape)
print(X_cv_teacher.shape, y_cv.shape)
print(X_test_teacher.shape, y_test.shape)
print(vectorizer4.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 5) (49041,)
(24155, 5) (24155,)
(36052, 5) (36052,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
==========================================================================
==========================
```

6.Price feature normalizing

In [12]:

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

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



print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
#print(normalizer.get_feature_names)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
==========================================================================
==========================
```

Concating all features

Set 1 : BOW

In [13]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack  #Combining all features
X_tr1 = hstack((X_train_clean_cat, X_train_clean_subcat, X_train_state_ohe, X_train_cle
angrade, X_train_teacher,X_train_bowessay)).tocsr()
X_cr1 = hstack((X_cv_clean_cat, X_cv_clean_subcat, X_cv_state_ohe, X_cv_cleangrade, X_c
v_teacher,X_cv_bowessay)).tocsr()
X_te1= hstack((X_test_clean_cat, X_test_clean_subcat, X_test_state_ohe, X_test_cleangra
de, X_test_teacher,X_test_bowessay)).tocsr()

print("Final Data matrix")
print(X_tr1.shape, y_train.shape)
print(X_cr1.shape, y_cv.shape)
print(X_te1.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(49041, 166592) (49041,)
(24155, 166592) (24155,)
(36052, 166592) (36052,)
==========================================================================
==========================
```

Set 2 : TFIDF

In [16]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr2 = hstack((X_train_clean_cat, X_train_clean_subcat, X_train_state_ohe, X_train_cle
angrade, X_train_teacher,X_train_tfidfessay)).tocsr()
X_cr2 = hstack((X_cv_clean_cat, X_cv_clean_subcat, X_cv_state_ohe, X_cv_cleangrade, X_c
v_teacher,X_cv_tfidfessay)).tocsr()
X_te2= hstack((X_test_clean_cat, X_test_clean_subcat, X_test_state_ohe, X_test_cleangra
de, X_test_teacher,X_test_tfidfessay)).tocsr()

print("Final Data matrix")
print(X_tr2.shape, y_train.shape)
print(X_cr2.shape, y_cv.shape)
print(X_te2.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(49041, 166592) (49041,)
(24155, 166592) (24155,)
(36052, 166592) (36052,)
==========================================================================
==========================
```

1.5 Appling NB on different kind of featurization as mentioned in the instructions

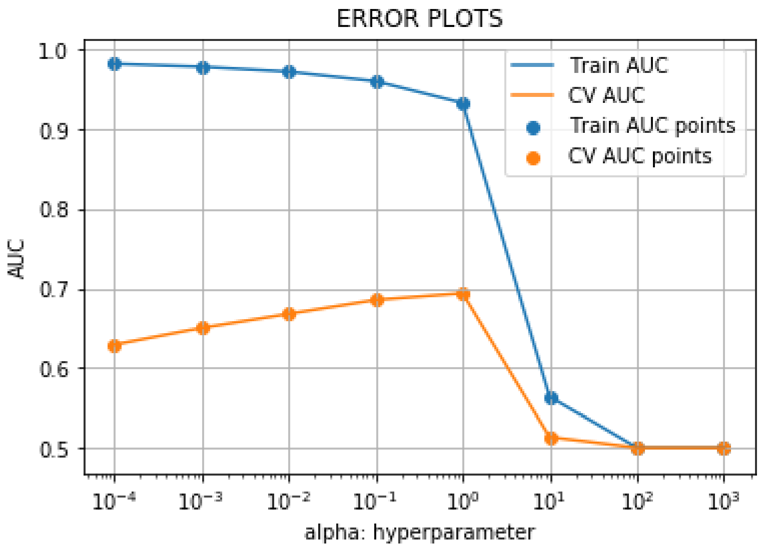
Apply NB on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

1.5.1 Appling Naive Bayes(MultinomialNB): BOW featurization

In [22]:

```
#NB for Bow

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
from sklearn.naive_bayes import MultinomialNB
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 non-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_ac = []
cv_ac = []
alpha = [0.0001,0.001,0.01,0.1,1,10,100,1000]
for i in tqdm(alpha):
    nom = MultinomialNB(alpha=i) #Iterating alpha for each i value
    nom.fit(X_tr1,y_train)  #fit the model
      # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimate
s of the positive class
    # not the predicted outputs
    y_train_pred = nom.predict_proba(X_tr1)[:,1] #Returns the probability estimates of
train set
    y_cv_pred = nom.predict_proba(X_cr1)[:,1]  #Returns the orobability estimates of cv
set
    train_ac.append(roc_auc_score(y_train,y_train_pred))  #Computing roc auc curve for
 train
    cv_ac.append(roc_auc_score(y_cv, y_cv_pred))  #Computing roc auc curve for cv

plt.plot(alpha, train_ac, label='Train AUC')
plt.plot(alpha, cv_ac, label='CV AUC')
plt.scatter(alpha, train_ac, label='Train AUC points')
plt.scatter(alpha, cv_ac, label='CV AUC points')
plt.legend()
plt.xscale('log')
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████████████████████████████████████████████████████████████
███████████████| 8/8 [00:03<00:00,  2.09it/s]
```



In [23]:

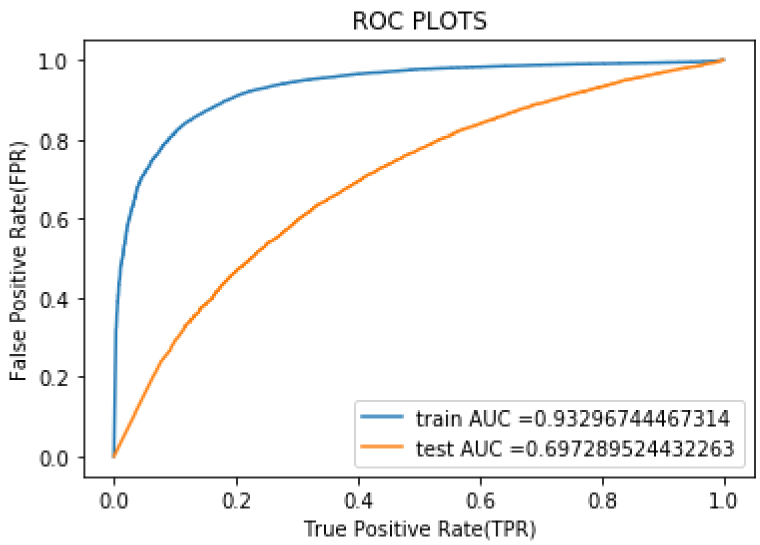
```
#Calculate max auc score
aucscore = [x for x in cv_ac]  # stored in list
val = alpha[aucscore.index(max(aucscore))] #Finding max score index and find alpha para
meter
print("Maximum AUC score of cv is:" + ' ' + str(max(aucscore)))
print("Corresponding alpha value of cv is:",val, '\n')
bestalpha=val
print(bestalpha)
```

```
Maximum AUC score of cv is: 0.6938985663860762
Corresponding alpha value of cv is: 1

1
```

Hyper Parameter alpha fitting to model

In [24]:

```
from sklearn.metrics import roc_curve, auc
neigh = MultinomialNB(alpha=1)
neigh.fit(X_tr1 ,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, tr_thresholds = roc_curve(y_train, neigh.predict_proba(X_tr1)[:,1
]) #Computing roc curve for train
test_fpr, test_tpr, tr_thresholds = roc_curve(y_test, neigh.predict_proba(X_te1)[:,1])
#Computing roc curve for test
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)
```

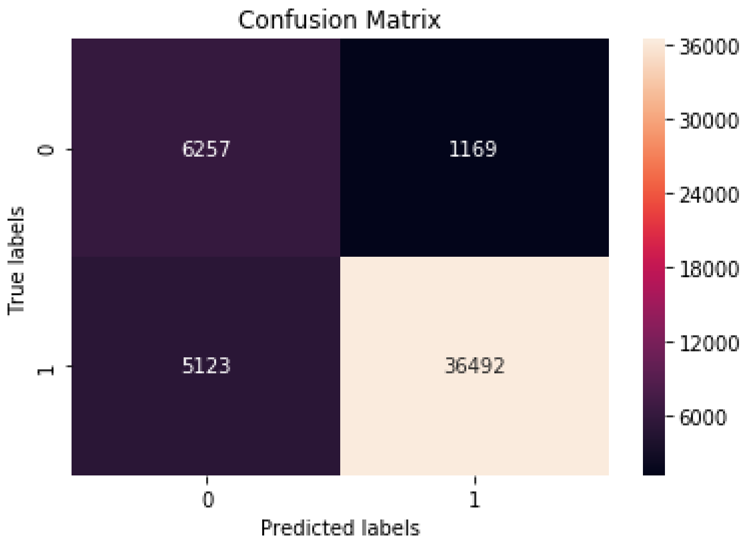



```
============================================================================
=========================
```

Confusion Matrix

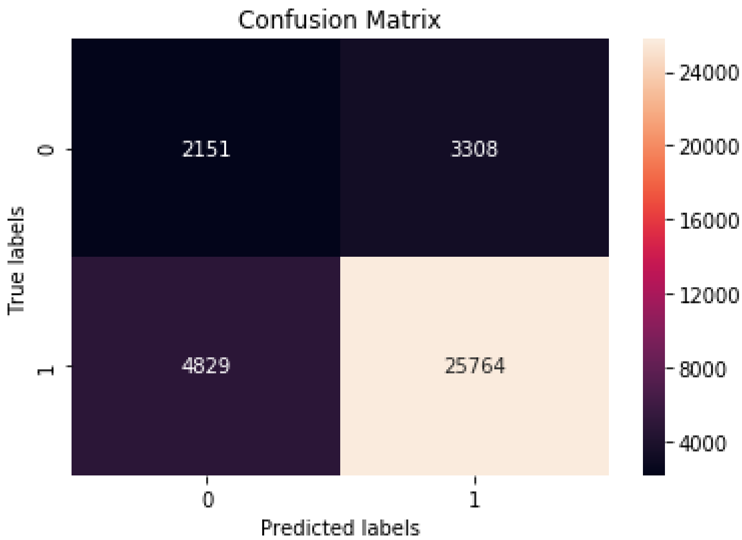
In [30]:

```
#Matrix for training data
import seaborn as sns
import matplotlib.pyplot as plt
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_train, neigh.predict(X_tr1)), annot=True, ax = ax,fmt=
'g');
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```



In [32]:

```
#Matrix for testing data
import seaborn as sns
import matplotlib.pyplot as plt
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, neigh.predict(X_te1)), annot=True, ax = ax,fmt='g'
);
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```



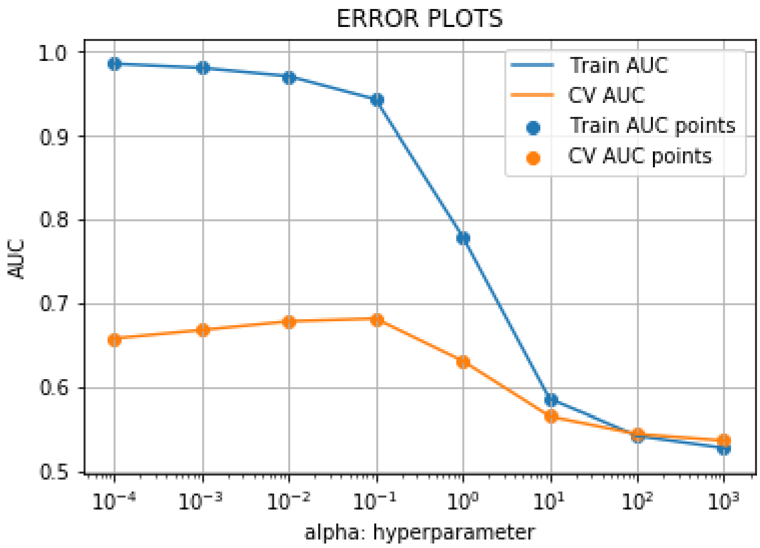
Observations : 1.In Naive Bayes BOW Featurization,as i observed based on roc plot, model is good in train and test data,but only little bit overfitting 2.Based upon hyper parameter tunning,in bow featurization best alpha value is 1 3.In confusion matrix,for imbalanced dataset true postives is more than false negative and false positives

1.5.1 Appling Naive Bayes(MultinomialNB): TFIDF featurization

In [33]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
from sklearn.naive_bayes import MultinomialNB
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 non-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_ac1 = []
cv_ac1 = []
alpha = [0.0001,0.001,0.01,0.1,1,10,100,1000]
for i in tqdm(alpha):
    nom1 = MultinomialNB(alpha=i)#Iterating alpha for each i value
    nom1.fit(X_tr2,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_pred1 = nom1.predict_proba(X_tr2)[:,1]#Returns the probability estimates of
train set
    y_cv_pred1 = nom1.predict_proba(X_cr2)[:,1]  #Returns the probability estimates of
 cv set
    train_ac1.append(roc_auc_score(y_train,y_train_pred1))  #Computing roc auc curve fo
r train
    cv_ac1.append(roc_auc_score(y_cv, y_cv_pred1))  #Computing roc auc curve for cv

plt.plot(alpha, train_ac1, label='Train AUC')
plt.plot(alpha, cv_ac1, label='CV AUC')
plt.scatter(alpha, train_ac1, label='Train AUC points')
plt.scatter(alpha, cv_ac1, label='CV AUC points')
plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████████████████████████████████████████████| 8/8 [00:02<00:00,  2.80it/s]
```



In [35]:

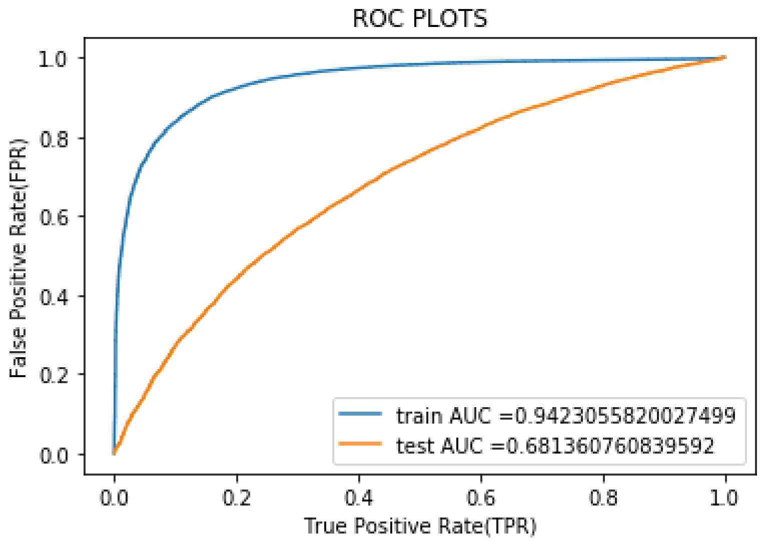
```
aucscore2 = [x for x in cv_ac1]
val2 = alpha[aucscore2.index(max(aucscore2))]
print("Maximum AUC score of cv is:" + ' ' + str(max(aucscore2)))
print("Corresponding alpha value of cv is:",val2, '\n')
bestalpha2=val2
print(bestalpha2)
```

```
Maximum AUC score of cv is: 0.6810158993482307
Corresponding alpha value of cv is: 0.1

0.1
```

Hyper Parameter alpha fitting to model

In [36]:

```
from sklearn.metrics import roc_curve, auc
neigh = MultinomialNB(alpha=0.1)
neigh.fit(X_tr2 ,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, tr_thresholds = roc_curve(y_train, neigh.predict_proba(X_tr2)[:,1
]) #Computing roc curve for train
test_fpr, test_tpr, tr_thresholds = roc_curve(y_test, neigh.predict_proba(X_te2)[:,1])
#Computing roc curve for test
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)
```

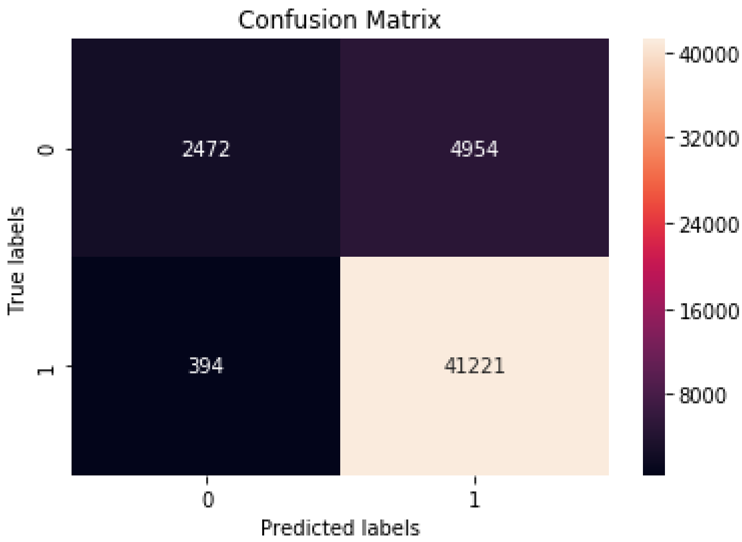



```
===========================================================================
==========================
```

Confusion Matrix

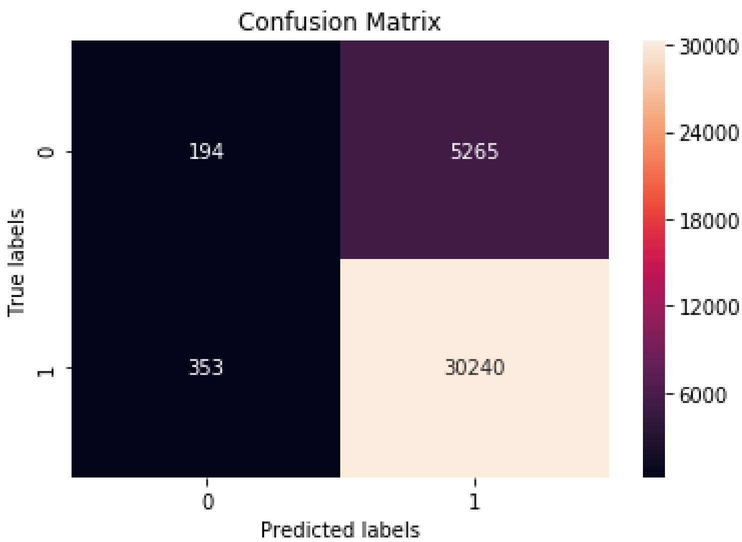
In [37]:

```
#Matrix for train data
import seaborn as sns
import matplotlib.pyplot as plt
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_train, neigh.predict(X_tr2)), annot=True, ax = ax,fmt=
'g');
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```



In [38]:

```
#Matrix for test
import seaborn as sns
import matplotlib.pyplot as plt
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, neigh.predict(X_te2)), annot=True, ax = ax,fmt='g'
);
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```



In [ ]:

```
Observations :
1.As i observed from roc plot,ROC curve for tfidf is less than roc curve for bow, it ha
s little bit overfitting
2.Based upon hyper parameter tunning,in tfidf featurization best alpha value is 0.1
3.In confusion matrix,very less true negative datapoints because on that data imbalance
is not work as much.
```

Top 10 Positive and Negative Features for BOW

In [59]:

```
#Top 10 features in Positive and negative for Bow

nbb = MultinomialNB(alpha=0.1) #Taking parameter as 0.1
nbb.fit(X_tr1,y_train) #Fit the model

bow_prob = []
for i in range(166592): #Iterating the loop
    bow_prob.append(nbb.feature_log_prob_[0,i]) #Calculate negative feature probabiliti
es

bow_feature = []
for i in vectorizer.get_feature_names() : #Categories
    bow_feature.append(i)
for i in vectorizer1.get_feature_names() : #sub Categories
    bow_feature.append(i)
for i in vectorizer2.get_feature_names() : #school state
    bow_feature.append(i)
for i in vectorizer3.get_feature_names() : #grade categories
    bow_feature.append(i)
for i in vectorizer4.get_feature_names() : #teacher prefix
    bow_feature.append(i)
for i in vectorizer5.get_feature_names() : #Essay
    bow_feature.append(i)


#top 10 negatives
finalfeatures = pd.DataFrame({'feature_prob_estimates' : bow_prob, 'feature_names'
: bow_feature})
a=finalfeatures.sort_values(by = ['feature_prob_estimates'], ascending = False)
a.head(10)
```

Out[59]:

	feature_prob_estimates	feature_names
129668	-3.676597	students
115914	-4.774282	school
73738	-5.088852	learning
89125	-5.128693	my
22950	-5.246464	classroom
94505	-5.427405	not
71780	-5.439249	learn
146233	-5.446493	they
59068	-5.473036	help
89786	-5.495444	my students

In [60]:

```
#Top 10 postives
#All probablities stored in list
bow_prob_pos = []
for i in range(166592):
    bow_prob_pos.append(nbb.feature_log_prob_[1,i])

finalfeatures_pos = pd.DataFrame({'feature_prob_estimates_positive' : bow_prob_pos,
'feature_names' : bow_feature})
a =finalfeatures_pos.sort_values(by = ['feature_prob_estimates_positive'], ascending =
False)
a.head(10)
```

Out[60]:

	feature_prob_estimates_positive	feature_names
129668	-3.667370	students
115914	-4.807328	school
89125	-5.122883	my
73738	-5.174551	learning
22950	-5.201345	classroom
143642	-5.422791	the
146233	-5.464058	they
94505	-5.466517	not
89786	-5.499499	my students
71780	-5.512554	learn

Top 10 Positive and Negative Features for TFIDF

In [64]:

```
#Top 10 features  negative for TFIDF

nbb = MultinomialNB(alpha=0.1)#Taking parameter as 0.1
nbb.fit(X_tr2,y_train)  #Fit the model

tfidf_prob = []

for i in range(166592): #Iterating the Loop
    tfidf_prob.append(nbb.feature_log_prob_[0,i])  #Calculate negative feature probabil
ities


tfidf_feature = []

for i in vectorizer.get_feature_names() : #Categories
    tfidf_feature.append(i)
for i in vectorizer1.get_feature_names() : #sub Categories
    tfidf_feature.append(i)
for i in vectorizer2.get_feature_names() : #school state
    tfidf_feature.append(i)
for i in vectorizer3.get_feature_names() : #grade categories
    tfidf_feature.append(i)
for i in vectorizer4.get_feature_names() : #teacher prefix
    tfidf_feature.append(i)
for i in vectorizer5.get_feature_names() : #Essay
    tfidf_feature.append(i)


#top 10 negatives
finalfeatures = pd.DataFrame({'feature_prob_estimates' : tfidf_prob, 'feature_names' :
tfidf_feature})
a =finalfeatures.sort_values(by = ['feature_prob_estimates'], ascending = False)
a.head(10)
```

Out[64]:

	feature_prob_estimates	feature_names
96	-3.765193	mrs
4	-3.931852	literacy_language
93	-3.986922	grades_prek_2
5	-3.989269	math_science
97	-4.084677	ms
90	-4.173063	grades_3_5
26	-4.410625	literacy
28	-4.411649	mathematics
27	-4.752619	literature_writing
91	-4.898983	grades_6_8

In [66]:

```
#Top 10 postives

#All probablities stored in list
tfidf_prob_pos = []
for i in range(166592):
    tfidf_prob_pos.append(nbb.feature_log_prob_[1,i])

finalfeatures_pos = pd.DataFrame({'feature_prob_estimates_positive' : tfidf_prob_pos,
'feature_names' : tfidf_feature})
a =finalfeatures_pos.sort_values(by = ['feature_prob_estimates_positive'], ascending =
False)
a.head(10)
```

Out[66]:

	feature_prob_estimates_positive	feature_names
96	-3.642965	mrs
4	-3.725345	literacy_language
93	-3.919194	grades_prek_2
5	-3.991176	math_science
97	-4.049358	ms
90	-4.076267	grades_3_5
26	-4.153818	literacy
28	-4.370050	mathematics
27	-4.590755	literature_writing
91	-4.875629	grades_6_8

Conculsion

In [68]:

```
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, 68])
print(tb.get_string(titles = "Naive Bayes - Observations"))
```

```
+------------+-------+----------------+-----+
| Vectorizer | Model | HyperParameter | AUC |
+------------+-------+----------------+-----+
|    BOW     |  Auto |       1        |  70 |
|   Tf-Idf   |  Auto |      0.1       |  68 |
+------------+-------+----------------+-----+
```

Observation: For Hyper parmater tunnning, Bow performs best AUC sCORE with hyper parameter '1' compared to TFIDF AUC score with hyper parameter '0.1'