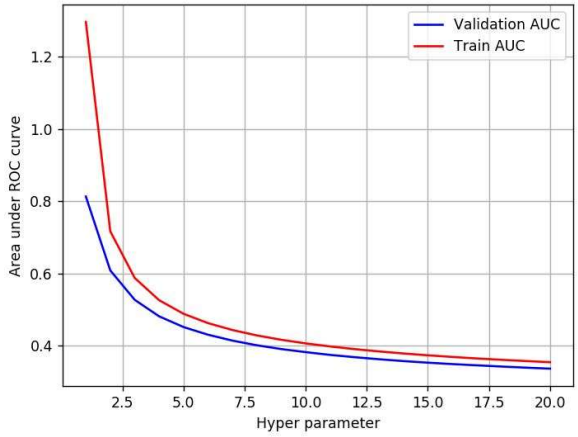
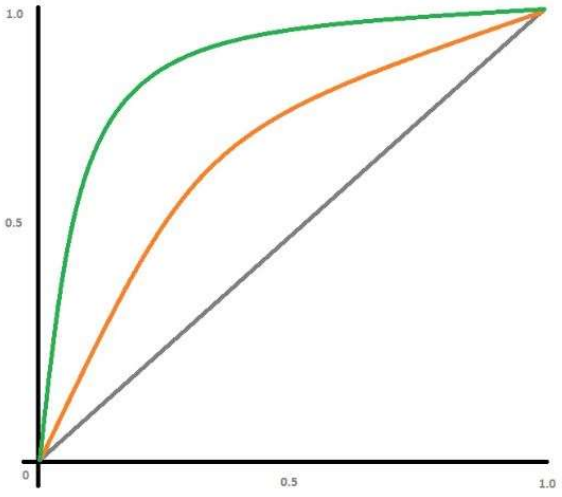
Assignment 6: Apply NB

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/)](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
W2V	Brute	10	0.78
TFIDFW2V	Brute	6	0.78

2. Naive Bayes

1.1 Loading Data

In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
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



import pickle
from tqdm import tqdm
import os

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

In [2]:
```
data = pd.read_csv('preprocessed_data.csv', nrows=100000)

## since we can see that project_title is not present in preprocessed_data so we


## 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


# 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'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', '
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itsel
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'th
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'oft
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than',
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "di
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn'
            'won', "won't", 'wouldn', "wouldn't"]

# Combining all the above stundents

def preprocess_text(text_data):
    preprocessed_text = []
    # tqdm is for printing the status bar
    for sentance in text_data:
        sent = decontracted(sentance)
        sent = sent.replace('\\r', ' ')
        sent = sent.replace('\\n', ' ')
        sent = sent.replace('\\"', ' ')
        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_text.append(sent.lower().strip())
    return preprocessed_text
project_data=pd.read_csv('train_data.csv', nrows=100000)
preprocessed_titles = preprocess_text(project_data['project_title'].values)

data['project_title']=preprocessed_titles
data.head(2)
```

Out[2]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projec
0	ca	mrs	grades_prek_2	
1	ut	ms	grades_3_5	

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

In [3]:
```
## separating data into X and Y such that x contains all set of features and Y co
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)

## splitting data into train, cross validate and test

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratif
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33,

print(X_train.shape)
print(X_test.shape)
print(X_cv.shape)
```

```
(44890, 9)
(33000, 9)
(22110, 9)
```

1.3 Make Data Model Ready: encoding eassay, and project_title

In [4]:
```
vectorizer_essay_bow = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=

## word2vec for essay
vectorizer_essay_bow.fit(X_train['essay'].values)

X_train_essay_bow = vectorizer_essay_bow.transform(X_train['essay'].values)
X_cv_essay_bow = vectorizer_essay_bow.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer_essay_bow.transform(X_test['essay'].values)


## word2vec for project_title
vectorizer_title_bow = CountVectorizer(min_df=5,ngram_range=(1,4), max_features=5
vectorizer_title_bow.fit(X_train['project_title'].values)

X_train_project_title_bow = vectorizer_title_bow.transform(X_train['project_title
X_cv_project_title_bow = vectorizer_title_bow.transform(X_cv['project_title'].val
X_test_project_title_bow = vectorizer_title_bow.transform(X_test['project_title']
```

In [5]:
```
print("Essay vectorizer")
print(X_train_essay_bow.shape)
print(X_cv_essay_bow.shape)
print(X_test_essay_bow.shape)
print('*'*50)
print("Title vectorizer")
print(X_train_project_title_bow.shape)
print(X_cv_project_title_bow.shape)
print(X_test_project_title_bow.shape)
```

```
Essay vectorizer
(44890, 5000)
(22110, 5000)
(33000, 5000)
**************************************************
Title vectorizer
(44890, 5000)
(22110, 5000)
(33000, 5000)
```

1.4 Make Data Model Ready: encoding numerical, categorical features

In [6]:
```
## vectorizing school_state
vectorizer_school_state= CountVectorizer(binary=True)
vectorizer_school_state.fit(X_train['school_state'].values)

X_train_state_ohe = vectorizer_school_state.transform(X_train['school_state'].va
X_cv_state_ohe = vectorizer_school_state.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer_school_state.transform(X_test['school_state'].value

## vectorizing teachers prefix
vectorizer_teacher_prefix_cat= CountVectorizer(binary=True)
vectorizer_teacher_prefix_cat.fit(X_train['teacher_prefix'])

X_train_teacher_prefix_ohe=vectorizer_teacher_prefix_cat.transform(X_train['teach
X_cv_teacher_prefix_ohe=vectorizer_teacher_prefix_cat.transform(X_cv['teacher_pre
X_test_teacher_prefix_ohe=vectorizer_teacher_prefix_cat.transform(X_test['teacher

## vectorizing project_grade_category

vectorizer_project_grade_cat= CountVectorizer(binary=True)
vectorizer_project_grade_cat.fit(X_train['project_grade_category'])

X_train_project_grade_category_ohe=vectorizer_project_grade_cat.transform(X_train
X_cv_project_grade_category_ohe=vectorizer_project_grade_cat.transform(X_cv['proj
X_test_project_grade_category_ohe=vectorizer_project_grade_cat.transform(X_test['


## vectorizing clean_categories
vectorizer_categories= CountVectorizer(binary=True)
vectorizer_categories.fit(X_train['clean_categories'])

X_train_clean_categories_ohe=vectorizer_categories.transform(X_train['clean_categ
X_cv_clean_categories_ohe=vectorizer_categories.transform(X_cv['clean_categories'
X_test_clean_categories_ohe=vectorizer_categories.transform(X_test['clean_categor

## vectorizing clean_subcategories
vectorizer_sub_cat= CountVectorizer(binary=True)
vectorizer_sub_cat.fit(X_train['clean_subcategories'])

X_train_clean_subcategories_ohe=vectorizer_sub_cat.transform(X_train['clean_subca
X_cv_clean_subcategories_ohe=vectorizer_sub_cat.transform(X_cv['clean_subcategori
X_test_clean_subcategories_ohe=vectorizer_sub_cat.transform(X_test['clean_subcate
```

In [7]:
```
## after vectorizing cheking output for any categorical feature say clean subcate

print("After vectorizations")
print(X_train_clean_subcategories_ohe.shape, y_train.shape)
print(X_cv_clean_subcategories_ohe.shape, y_cv.shape)
print(X_test_clean_subcategories_ohe.shape, y_test.shape)
print(vectorizer_categories.get_feature_names())
```

```
After vectorizations
(44890, 30) (44890,)
(22110, 30) (22110,)
(33000, 30) (33000,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy
_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
```

In [8]:
```
## tfidf vectorizer for essay

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_essay_tfidf = TfidfVectorizer(min_df=10)
vectorizer_essay_tfidf.fit(X_train['essay'])

essay_tfidf_train = vectorizer_essay_tfidf.transform(X_train['essay'])
essay_tfidf_test = vectorizer_essay_tfidf.transform(X_test['essay'])
essay_tfidf_cv = vectorizer_essay_tfidf.transform(X_cv['essay'])

print("Shape of train, test and CV are : ")
print(essay_tfidf_train.shape)
print(essay_tfidf_cv.shape)
print(essay_tfidf_test.shape)
```

```
Shape of train, test and CV are :
(44890, 11724)
(22110, 11724)
(33000, 11724)
```

In [9]:
```
vectorizer_title_tfidf = TfidfVectorizer(min_df=5)
vectorizer_title_tfidf.fit(X_train['project_title'].values)

X_train_project_title_tfidf = vectorizer_title_tfidf.transform(X_train['project_t
X_cv_project_title_tfidf = vectorizer_title_tfidf.transform(X_cv['project_title']
X_test_project_title_tfidf = vectorizer_title_tfidf.transform(X_test['project_tit

print("Shape of train, test and CV are : ")
print(X_train_project_title_tfidf.shape)
print(X_cv_project_title_tfidf.shape)
print(X_test_project_title_tfidf.shape)
```

```
Shape of train, test and CV are :
(44890, 2974)
(22110, 2974)
(33000, 2974)
```

Normalizing numerical feature

In [10]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

# normalizing price feature
normalizer.fit(X_train['price'].values.reshape(1,-1))

## after normalizing using reshape (1,-1) again reshaping back to (-1,1) for fixi
X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(-1,1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(-1,1))
X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(-1,1))

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



## normalizing privously_submitted_project feature

normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.res

## after normalizing using reshape (1,-1) again reshaping back to (-1,1) for fixi
prev_no_projects_train = normalizer.transform(X_train['teacher_number_of_previous
prev_no_projects_cv = normalizer.transform(X_cv['teacher_number_of_previously_pos
prev_no_projects_test = normalizer.transform(X_test['teacher_number_of_previously

print("*"*50)
print("After vectorizations")
print(prev_no_projects_train.shape, y_train.shape)
print(prev_no_projects_cv.shape, y_cv.shape)
print(prev_no_projects_test.shape, y_test.shape)
```

```
After vectorizations
(44890, 1) (44890,)
(33000, 1) (33000,)
(22110, 1) (22110,)
**************************************************
After vectorizations
(44890, 1) (44890,)
(22110, 1) (22110,)
(33000, 1) (33000,)
```

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

concatinating features as per given SET 1 and SET 2 :

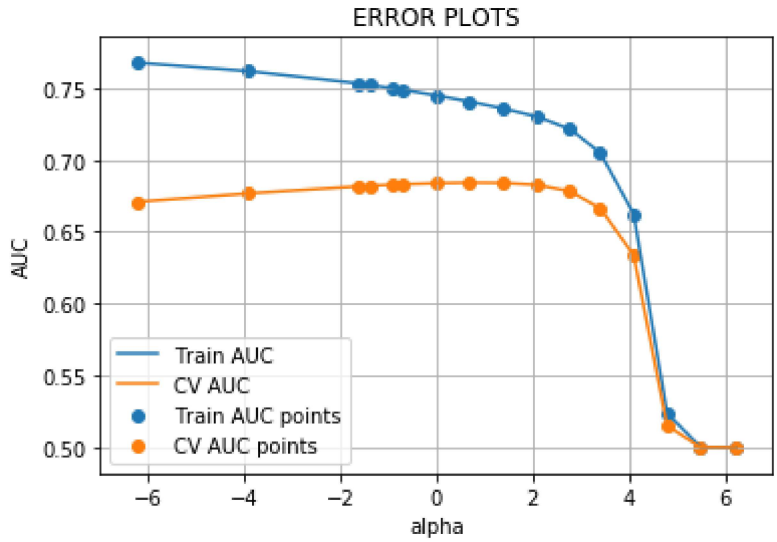
In [11]:
```
## set 1 :  categorical, numerical features + preprocessed_eassay (BOW)

from scipy.sparse import hstack

X_tr_1 = hstack((X_train_essay_bow,X_train_project_title_bow,prev_no_projects_tra
                 X_train_state_ohe,X_train_teacher_prefix_ohe,X_train_project_gra
                 X_train_clean_categories_ohe,X_train_clean_subcategories_ohe,X_t


X_cv_1= hstack((X_cv_essay_bow,X_cv_project_title_bow,prev_no_projects_cv,X_cv_st
                X_cv_project_grade_category_ohe,X_cv_clean_categories_ohe,X_cv_cl

X_te_1= hstack((X_test_essay_bow,X_test_project_title_bow,prev_no_projects_test,X
                X_test_project_grade_category_ohe,X_test_clean_categories_ohe,\
                X_test_clean_subcategories_ohe,X_test_price_norm)).tocsr()

print("Final Data matrix for set 1")
print(X_tr_1.shape, y_train.shape)
print(X_cv_1.shape, y_cv.shape)
print(X_te_1.shape, y_test.shape)
print("*"*100)

## set 2 :  categorical, numerical features + preprocessed_eassay (TFIDF)

from scipy.sparse import hstack

X_tr_2= hstack((essay_tfidf_train,X_train_project_title_tfidf,prev_no_projects_tr
                X_train_teacher_prefix_ohe,X_train_project_grade_category_ohe,X_t
                X_train_clean_subcategories_ohe,X_train_price_norm)).tocsr()

X_cv_2= hstack((essay_tfidf_cv,X_cv_project_title_tfidf,prev_no_projects_cv,X_cv_
                X_cv_project_grade_category_ohe,X_cv_clean_categories_ohe,X_cv_cl

X_te_2= hstack((essay_tfidf_test,X_test_project_title_tfidf,prev_no_projects_test
                X_test_teacher_prefix_ohe,X_test_project_grade_category_ohe,X_tes
                X_test_clean_subcategories_ohe,X_test_price_norm)).tocsr()

print("Final Data matrix for set 2")
print(X_tr_2.shape, y_train.shape)
print(X_cv_2.shape, y_cv.shape)
print(X_te_2.shape, y_test.shape)
```

```
Final Data matrix for set 1
(44890, 10101) (44890,)
(22110, 10101) (22110,)
(33000, 10101) (33000,)
****************************************************************************
********************
Final Data matrix for set 2
(44890, 14799) (44890,)
(22110, 14799) (22110,)
(33000, 14799) (33000,)
```

Finding best hyper parameter, AUC and confusion matrix for SET 1

In [12]:
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability esti
    # not the predicted outputs

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

    return y_data_pred
```

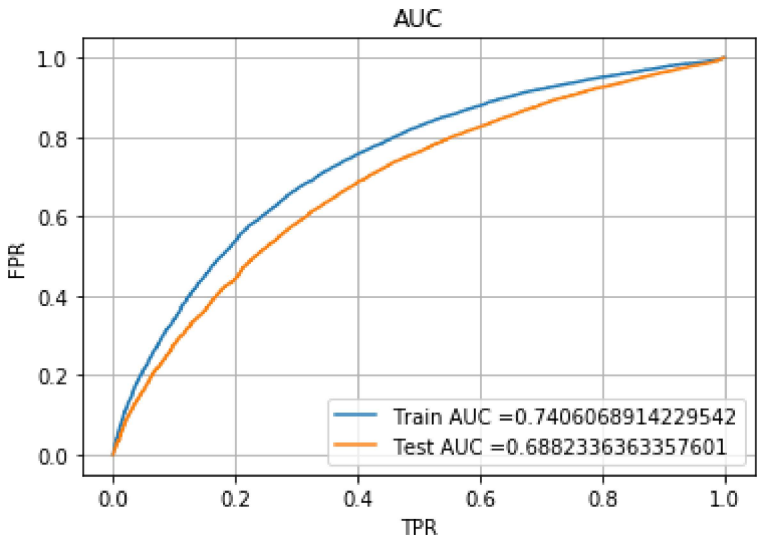
In [13]:
```
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
import math

train_auc, cv_auc, log_alpha = [],[],[]

alpha = [0.002,0.02,0.2,0.25,0.4,0.5,1,2,4,8,16,30,60,120,240,500]

for i in alpha:
    clf = MultinomialNB(alpha=i,class_prior=[0.5,0.5])
    clf.fit(X_tr_1, y_train)

    y_train_pred = batch_predict(clf, X_tr_1)
    y_cv_pred = batch_predict(clf, X_cv_1)

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

for a in alpha:
    b = math.log(a)
    log_alpha.append(b)

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

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

plt.legend()
plt.xlabel("alpha")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Observation

From above we can see that best alpha is 2

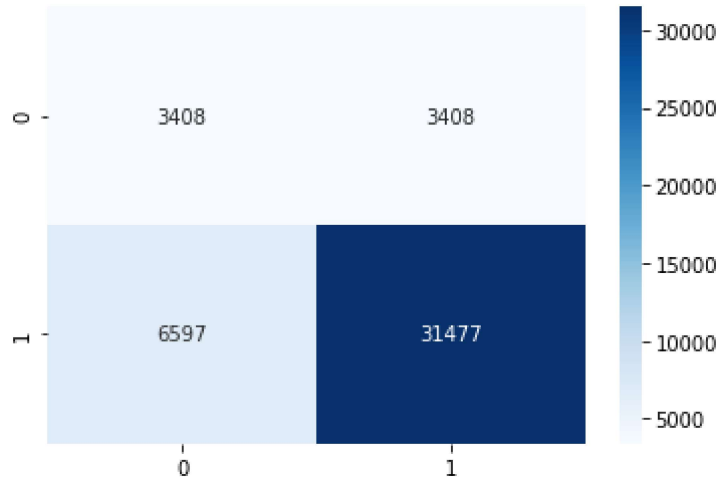
In [14]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.htm

clf_bow = MultinomialNB(alpha =2)
clf_bow.fit(X_tr_1, y_train)


y_train_pred = batch_predict(clf_bow, X_tr_1)
y_test_pred = batch_predict(clf_bow, X_te_1)

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("TPR")
plt.ylabel("FPR")
plt.title("AUC")
plt.grid()
plt.show()
```



In [15]:
```
## finding best threshold value
## and later using this value to calculate confusion matrix


def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    print("best threshold is : ", np.round(t,4))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

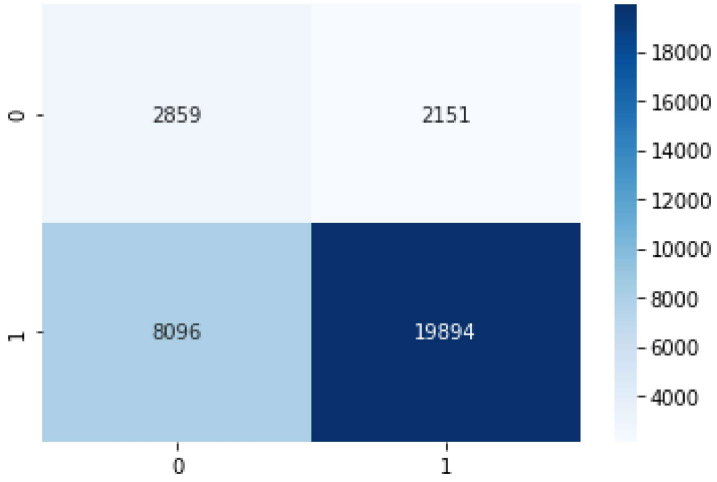
In [16]:
```
from sklearn.metrics import confusion_matrix

array_1=confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr,

import seaborn as sns
import matplotlib.pyplot as plt

sns.heatmap(array_1, annot=True,fmt="d",cmap='Blues')
plt.show()
```

```
best threshold is :  0.2096
```



In [17]:
```
array_2=confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, te

sns.heatmap(array_2, annot=True,fmt="d",cmap='Blues')
plt.show()
```

```
best threshold is :  0.7491
```



finding top 20 features for set 1

In [18]:
```
## getting feature counts and saving into other variables and further use this va

a=vectorizer_categories.get_feature_names()
b=vectorizer_sub_cat.get_feature_names()
c=vectorizer_school_state.get_feature_names()
d=vectorizer_project_grade_cat.get_feature_names()
e=vectorizer_teacher_prefix_cat.get_feature_names()
f=vectorizer_essay_bow.get_feature_names()
g=vectorizer_title_bow.get_feature_names()
h=vectorizer_essay_tfidf.get_feature_names()
i=vectorizer_title_tfidf.get_feature_names()
```

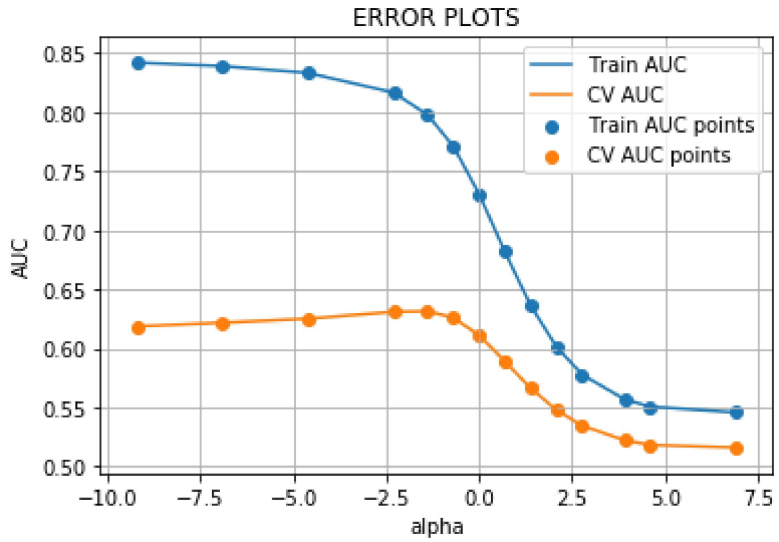
In [19]:
```
## converting all features into a single list
nb = MultinomialNB(alpha = 0.2,class_prior=[0.5,0.5]) #Multinominal Naive Bayes.
nb.fit(X_tr_1, y_train)

from itertools import chain
feature_names_bow = []
feature_names_bow = list(chain(a,b,c,d,e,["Price","teacher_number_of_previously_p
```

In [20]:
```
def feature_importance(clf,feature_nm):

    negative_class = clf.feature_log_prob_[0, :]
    positive_class = clf.feature_log_prob_[1, :]

    feature_impt_positive = pd.Series(positive_class, feature_nm).sort_values(asc
    feature_impt_negative = pd.Series(negative_class, feature_nm).sort_values(asc

    return (feature_impt_positive.head(10),feature_impt_negative.head(10))

Set_1_features_positive,Set_1_features_negative= feature_importance(nb,feature_na
print("Top 10 Positive features of Set 1\n",Set_1_features_positive,"\nTop 10 neg
```

```
Top 10 Positive features of Set 1
 spectrum disorder      -3.240363
relax                  -4.384520
meet needs students    -4.701178
kit                    -4.755298
cell                   -4.780807
supports               -4.996223
my students walk       -5.044206
technology available   -5.049279
middle school          -5.075982
it amazing             -5.096917
dtype: float64
Top 10 negative features of set 1 are
 spectrum disorder      -3.263290
relax                  -4.352229
kit                    -4.684050
meet needs students    -4.707684
cell                   -4.821957
my students walk       -4.997096
it amazing             -5.007636
technology available   -5.044532
supports               -5.053393
grade students         -5.057269
dtype: float64
```

Finding best hyper parameter, AUC and confusion matrix for SET 2

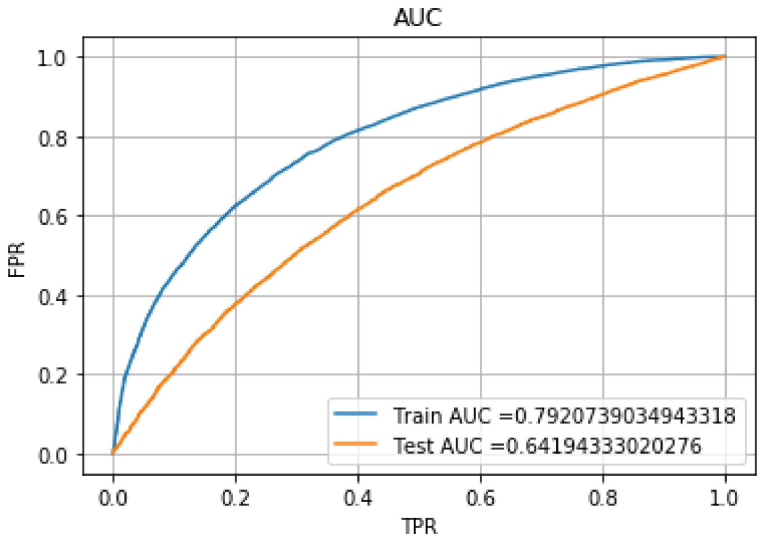
In [21]:
```
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
import math

train_auc, cv_auc, log_alpha = [],[],[]

alpha = [0.0001,0.001,0.01,0.1,0.25,0.5,1,2,4,8,16,50,100,1000]

for i in alpha:
    clf = MultinomialNB(alpha=i,class_prior=[0.5,0.5])
    clf.fit(X_tr_2, y_train)

    y_train_pred = batch_predict(clf, X_tr_2)
    y_cv_pred = batch_predict(clf, X_cv_2)

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

for a in alpha:
    b = math.log(a)
    log_alpha.append(b)

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

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

plt.legend()
plt.xlabel("alpha")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Observation

From above we can see that best alpha is 0.2

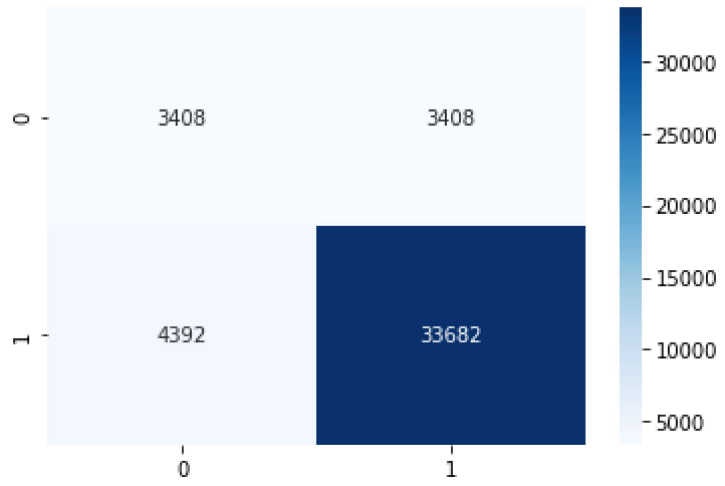
In [30]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.htm

clf_bow = MultinomialNB(alpha =0.2)
clf_bow.fit(X_tr_2, y_train)


y_train_pred = batch_predict(clf_bow, X_tr_2)
y_test_pred = batch_predict(clf_bow, X_te_2)

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("TPR")
plt.ylabel("FPR")
plt.title("AUC")
plt.grid()
plt.show()
```



In [23]:
```
## finding best threshold value
## and later using this value to calculate confusion matrix


def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    print("best threshold is : ", np.round(t,4))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

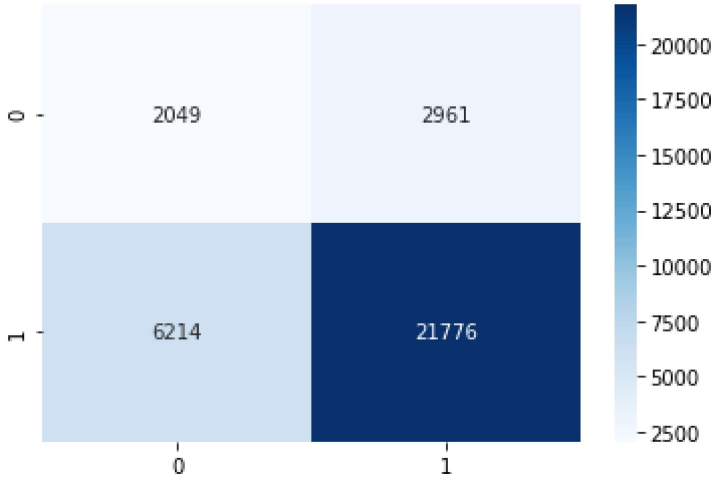
In [24]:
```
array_3=confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr,

sns.heatmap(array_3, annot=True,fmt="d",cmap='Blues')
plt.show()
```

```
best threshold is :  0.7729
```



In [25]:
```
array_4=confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, te

sns.heatmap(array_4, annot=True,fmt="d",cmap='Blues')
plt.show()
```

```
best threshold is :  0.8176
```



Finding top 20 features from SET 2

In [26]:
```
## getting feature counts and saving into other variables and further use this va

a=vectorizer_categories.get_feature_names()
b=vectorizer_sub_cat.get_feature_names()
c=vectorizer_school_state.get_feature_names()
d=vectorizer_project_grade_cat.get_feature_names()
e=vectorizer_teacher_prefix_cat.get_feature_names()
f=vectorizer_essay_bow.get_feature_names()
g=vectorizer_title_bow.get_feature_names()
h=vectorizer_essay_tfidf.get_feature_names()
i=vectorizer_title_tfidf.get_feature_names()

nb = MultinomialNB(alpha = 0.2,class_prior=[0.5,0.5]) #Multinominal Naive Bayes.
nb.fit(X_tr_2, y_train)


from itertools import chain
feature_names_tfidf = []
feature_names_tfidf = list(chain(a,b,c,d,e,["Price","teacher_number_of_previously


def feature_importance(clf,feature_nm):

    negative_class = clf.feature_log_prob_[0, :]
    positive_class = clf.feature_log_prob_[1, :]

    feature_impt_positive = pd.Series(positive_class, feature_nm).sort_values(asc
    feature_impt_negative = pd.Series(negative_class, feature_nm).sort_values(asc

    return (feature_impt_positive.head(10),feature_impt_negative.head(10))

Set_2_features_positive,Set_2_features_negative = feature_importance(nb,feature_n
print("Top 10 Positive features of set 2\n\n",Set_2_features_positive,"\nTop 10 N
```

```
Top 10 Positive features of set 2

 zooming        -2.913456
weebles        -3.235315
word           -3.549928
workstations   -3.634131
workout        -3.819977
world          -3.885751
words          -3.950733
workbooks      -3.987029
yearbook       -4.062419
yes            -4.261348
dtype: float64
Top 10 Negative features of set 2
 zooming        -2.918666
weebles        -3.320163
word           -3.614722
workstations   -3.794445
workout        -3.815445
world          -3.851703
words          -3.915218
```

```
workbooks      -4.039920
yes            -4.281672
yearbook       -4.295527
dtype: float64
```

Summary

In [31]:
```
## displaying final output
from prettytable import PrettyTable

x_pretty_table = PrettyTable()
x_pretty_table.field_names = ["Model Type","Vectorizer", "Hyper Parameter - alpha

x_pretty_table.add_row(["Naive Bayes","BOW",2,0.74,0.69])
x_pretty_table.add_row([ "Naive Bayes", "TFIDF",0.2,0.79,0.64])
print(x_pretty_table)
```

```
+-------------+------------+-------------------------+-----------+----------+
|  Model Type | Vectorizer | Hyper Parameter - alpha | Train-AUC | Test-AUC |
+-------------+------------+-------------------------+-----------+----------+
| Naive Bayes |    BOW     |            2            |    0.74   |   0.69   |
| Naive Bayes |   TFIDF    |           0.2           |    0.79   |   0.64   |
+-------------+------------+-------------------------+-----------+----------+
```