Assignment 6: Apply NB

1. Minimum data points need to be considered for people having 4GB RAM is **50k** and for 8GB RAM is **100k**
2. When you are using ramdomsearchcv or gridsearchcv you need not split the data into X_train,X_cv,X_test. As the above methods use kfold. The model will learn better if train data is more so splitting to X_train,X_test will suffice.
3. If you are writing for loops to tune your model then you need split the data into X_train,X_cv,X_test.
4. While splitting the data explore stratify parameter.
5. **Apply Multinomial NB on these feature sets**
    - Features that need to be considered

      essay
        while encoding essay, try to experiment with the max_features and n_grams parameter of vectorizers and see if it increases AUC score.

      categorical features
        - teacher_prefix
        - project_grade_category
        - school_state
        - clean_categories
        - clean_subcategories

      numerical features
        - price
        - teacher_number_of_previously_posted_projects

        while encoding the numerical features check this and this

    - Set 1: categorical, numerical features + preprocessed_eassay (BOW)
    - Set 2: categorical, numerical features + preprocessed_eassay (TFIDF)
6. **The hyper paramter tuning(find best alpha:smoothing parameter)**
    - Consider alpha values in range: 10^-5 to 10^2 like [0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,100]
    - Explore class_prior = [0.5, 0.5] parameter which can be present in MultinomialNB function(go through this) then check how results might change.
    - Find the best hyper parameter which will give the maximum AUC value
    - For hyper parameter tuning using k-fold cross validation(use GridsearchCV or RandomsearchCV)/simple cross validation data (write for loop to iterate over hyper parameter values)
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the

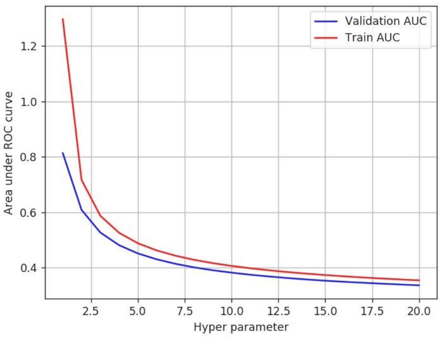
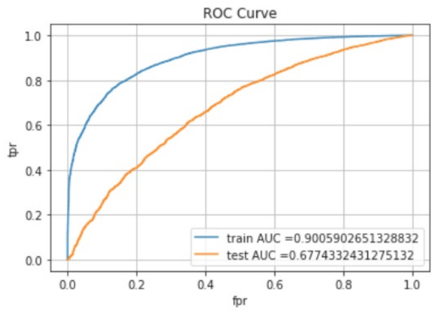


      figure
        -while plotting take log(alpha) on your X-axis so that it will be more readable
    - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC



      curve on both train and test.
    - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points

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

-plot the confusion matrix in heatmaps, while plotting the confusion matrix go through the link

7. find the top 20 features from either from feature Set 1 or feature Set 2 using values of feature$_{log}$ _ prob parameter of Mt ∈ omialNB (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html) and print **BOTH** positive as well as negative corresponding feature names.
   - go through the link
8. 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 |
+------------+--------+-----------------+------+
```

In [ ]:

1. Naive Bayes

In [2]:

```
%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 re

import pickle
from tqdm import tqdm
import os

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

1.1 Loading Data

In [3]:

```
import pandas
data = pandas.read_csv('preprocessed_data.csv', nrows = 50000)
data.head(5)
```

Out[3]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categories	cl
0	ca	mrs	grades_prek_2	53	1	math_science	
1	ut	ms	grades_3_5	4	1	specialneeds	

2	ca	mrs	grades_prek_2	10	1	literacy_language
3	ga	mrs	grades_prek_2	2	1	appliedlearning
4	wa	mrs	grades_3_5	2	1	literacy_language

In [4]:
```
# Seperating target class feature
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
```

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

In [5]:
```
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, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

1.3 Make Data Model Ready: encoding eassay

In [6]:
```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
#print(X_train.head(5))
```

```
(22445, 8) (22445,)
(11055, 8) (11055,)
(16500, 8) (16500,)
```

1.3.1 Encoding Text Feature EASSAY (BoW)

In [39]:
```
#Encoding text features(Essay)
vectorizer = CountVectorizer(min_df = 10, ngram_range = (1,3), max_features = 50000)
vectorizer.fit(X_train['essay'].values) #fit() to get the unique words
#transform unique words present count as BOW representation
x1 = vectorizer.get_feature_names()
X_train_essay_bow = vectorizer.transform(X_train['essay'].values)
X_cv_essay_bow = vectorizer.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
```

```
After vectorizations
(22445, 50000) (22445,)
(11055, 50000) (11055,)
(16500, 50000) (16500,)
```

In [ ]:
```
```

1.3.2 Encoding Text Feature EASSAY (tfidf)

In [65]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['essay'].values) #fit() to get the unique words
#transform unique words present count as tfidf representation

X_train_essay_tfidf = vectorizer.transform(X_train['essay'].values)
```

```
X_cv_essay_tfidf = vectorizer.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
```

```
After vectorizations
(22445, 8800) (22445,)
(11055, 8800) (11055,)
(16500, 8800) (16500,)
```

1.4 Make Data Model Ready: encoding numerical, categorical features(BoW)

1.4.1 Encoding categorical feature : Teacher_Prefix

In [42]:
```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data
x2 = vectorizer.get_feature_names()
# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
```

```
After vectorizations
(22445, 5) (22445,)
(11055, 5) (11055,)
(16500, 5) (16500,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
```

1.4.2 Encoding categorical feature : Project_grade_category

In [43]:
```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only on train data
x3 = vectorizer.get_feature_names()
# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
```

```
After vectorizations
(22445, 4) (22445,)
(11055, 4) (11055,)
(16500, 4) (16500,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
```

1.4.3 Encoding categorical feature : school_state

In [44]:
```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train data
x4 = vectorizer.get_feature_names()
# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.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(vectorizer.get_feature_names())
```

```
After vectorizations
(22445, 51) (22445,)
(11055, 51) (11055,)
(16500, 51) (16500,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la'
, 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or
', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'wy']
```

1.4.4 Encoding categorical feature : clean_categories

In [45]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on train data
x5 = vectorizer.get_feature_names()
# we use the fitted CountVectorizer to convert the text to vector
X_train_cleancat_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_cv_cleancat_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_cleancat_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_cleancat_ohe.shape, y_train.shape)
print(X_cv_cleancat_ohe.shape, y_cv.shape)
print(X_test_cleancat_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
```

```
After vectorizations
(22445, 9) (22445,)
(11055, 9) (11055,)
(16500, 9) (16500,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'music
_arts', 'specialneeds', 'warmth']
```

1.4.5 Encoding categorical feature : clean_subcategories

In [46]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_cleansubcat_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_cleansubcat_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_cleansubcat_ohe = vectorizer.transform(X_test['clean_subcategories'].values)
x6 = vectorizer.get_feature_names()
print("After vectorizations")
print(X_train_cleansubcat_ohe.shape, y_train.shape)
print(X_cv_cleansubcat_ohe.shape, y_cv.shape)
print(X_test_cleansubcat_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
```

```
After vectorizations
(22445, 30) (22445,)
(11055, 30) (11055,)
(16500, 30) (16500,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep', 'communityser
vice', 'earlydevelopment', 'economics', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy', 'f
oreignlanguages', 'gym_fitness', 'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'liter
ature_writing', 'mathematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'so
cialsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
```

1.4.6 Encoding numerical feature : price

In [14]:

```
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)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

1.4.7 Encoding numerical feature : teacher_number_of_previously_posted_projects

In [15]:

```
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['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

X_train_prevproj_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.resha
X_cv_prevproj_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1,
X_test_prevproj_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape



print("After vectorizations")
print(X_train_prevproj_norm.shape, y_train.shape)
print(X_cv_prevproj_norm.shape, y_cv.shape)
print(X_test_prevproj_norm.shape, y_test.shape)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

1.4.6 Concatenating all features

In [16]:

```
#set1:  categorical, numerical features + preprocessed_eassay (BOW)
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_bow, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_cleancat_ohe
X_cr = hstack((X_cv_essay_bow, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_cleancat_ohe, X_cv_cleansub
X_te = hstack((X_test_essay_bow, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_cleancat_ohe, X_t

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
```

```
Final Data matrix
(22445, 50101) (22445,)
(11055, 50101) (11055,)
(16500, 50101) (16500,)
```

In [17]:

```
#set2:  categorical, numerical features + preprocessed_eassay (TFIDF)
from scipy.sparse import hstack
X_tr_tf = hstack((X_train_essay_tfidf, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_cleanca
X_cr_tf = hstack((X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_cleancat_ohe, X_cv_cle
X_te_tf = hstack((X_test_essay_tfidf, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_cleancat_ohe

print("Final Data matrix")
print(X_tr_tf.shape, y_train.shape)
print(X_cr_tf.shape, y_cv.shape)
print(X_te_tf.shape, y_test.shape)
```

```
Final Data matrix
(22445, 8901) (22445,)
(11055, 8901) (11055,)
(16500, 8901) (16500,)
```

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 Naivebayes on Set1

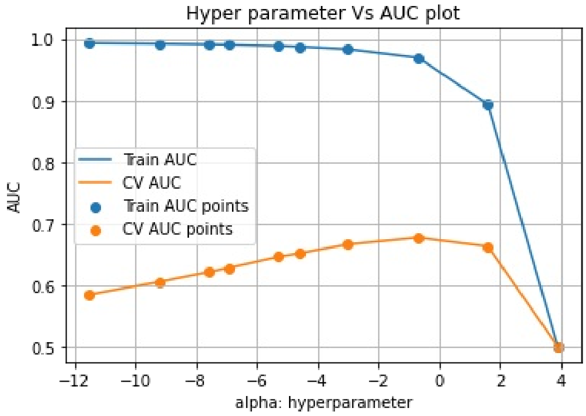
In [18]:
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs

    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%1000 = 49000
    # 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
```

1.5.2 RandomSearchCV for alpha smoothing

In [113…
```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
model = MultinomialNB(class_prior = [0.5, 0.5])
parameters = {'alpha':[0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,100]}
clf = RandomizedSearchCV(model, parameters, cv=3, scoring='roc_auc', return_train_score = True)
clf.fit(X_tr, y_train)

#neigh = KNeighborsClassifier(n_jobs=-1)
#parameters = {'n_neighbors':[3, 15, 25, 51, 101]}
#clf = RandomizedSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
#clf.fit(X_tr, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
#print(results.head(5))
results = results.sort_values(['param_alpha'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
alpha = results['param_alpha']
print('Alpha Values before log:', alpha)
# log of alpha values
K = []
for i in alpha:
    K.append(np.log(i))
print('Alpha Values after log:', K)
plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

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

plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```

```
Alpha Values before log: 4     1e-05
6    0.0001
7    0.0005
0     0.001
3     0.005
1      0.01
5      0.05
9       0.5
8         5
2        50
Name: param_alpha, dtype: object
Alpha Values after log: [-11.512925464970229, -9.210340371976182, -7.600902459542082, -6.907755278982137, -5.2983
```

17366548036, -4.605170185988091, -2.995732273553991, -0.6931471805599453, 1.6094379124341003, 3.912023005428146]



Out[113…

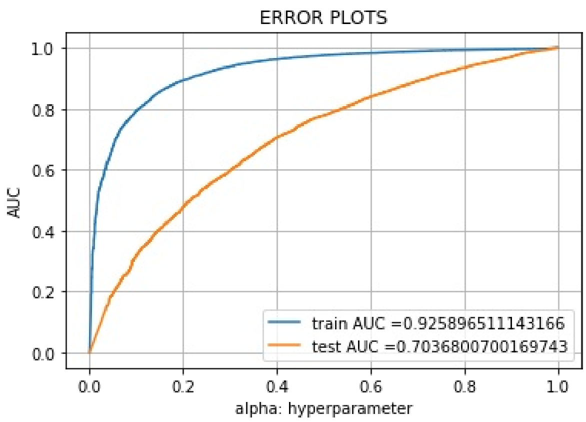
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_test_score	m
4	0.036333	0.000920	0.012321	0.000479	1e-05	{'alpha': 1e-05}	0.589102	0.580660	0.584003	
6	0.035678	0.000482	0.011319	0.000483	0.0001	{'alpha': 0.0001}	0.610910	0.605482	0.600363	
7	0.036063	0.000108	0.014225	0.004086	0.0005	{'alpha': 0.0005}	0.625479	0.619741	0.618330	
0	0.039335	0.004722	0.011991	0.000003	0.001	{'alpha': 0.001}	0.632599	0.626489	0.625542	
3	0.037000	0.001418	0.011691	0.000466	0.005	{'alpha': 0.005}	0.651204	0.642628	0.643859	

In [110…

```
# best alpha value
best_alpha = 5
```

In [161…

```
# run the model with best alpha and plotting error plot
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


mnb = MultinomialNB(alpha = best_alpha)
mnb.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(mnb, X_tr)
y_test_pred = batch_predict(mnb, X_te)
#y_train_pred = model.predict_proba(X_tr)
#y_test_pred = model.predict_proba(X_te)

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("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [24]:

```python
# reference notebook sample solutions from AAIC
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(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("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

1.5.3 Confusion Matrix

In [25]:

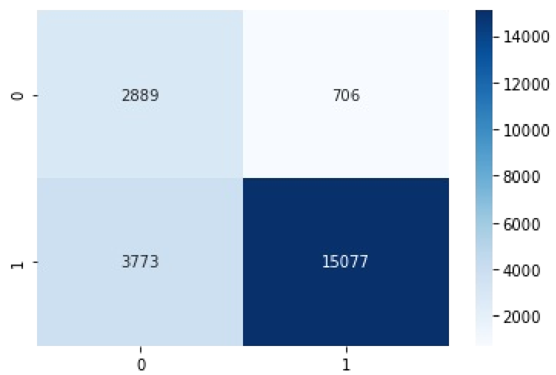
```python
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
cm_trn = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
print(cm_trn)
print("Test confusion matrix")
cm_test = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
print(cm_test)
```

```
the maximum value of tpr*(1-fpr) 0.6427650103481479 for threshold 1.0
Train confusion matrix
[[ 2889   706]
 [ 3773 15077]]
Test confusion matrix
[[ 1298  1344]
 [ 3050 10808]]
```

In [26]:

```python
# https://stackoverflow.com/questions/61748441/how-to-fix-the-values-displayed-in-a-confusion-matrix-in-exponenti
import seaborn as sns
import matplotlib.pyplot as plt

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

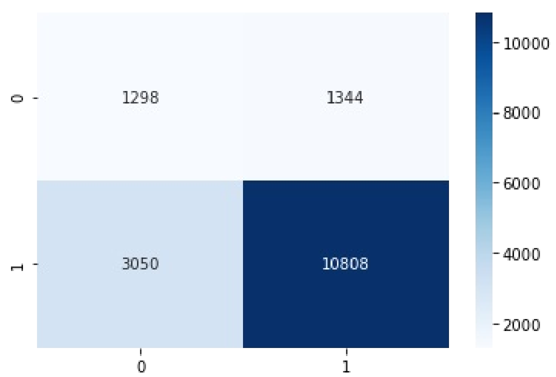
Out[26]: <AxesSubplot:>



In [28]:

```python
sns.heatmap(cm_test, annot = True, fmt="d", cmap='Blues')
```

Out[28]: <AxesSubplot:>

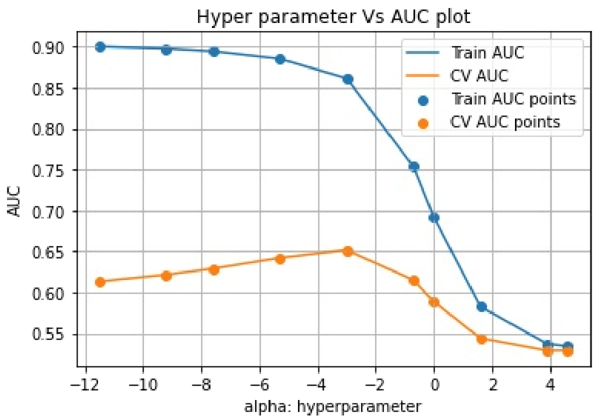


1.5.4 NaiveBayes on Set2

In [114…

```python
# RandomizedSearchCV for parameter tuning
from sklearn.model_selection import RandomizedSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
model = MultinomialNB(class_prior = [0.5, 0.5])
parameters = {'alpha':[0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,100]}
clf = RandomizedSearchCV(model, parameters, cv=3, scoring='roc_auc', return_train_score = True)
clf.fit(X_tr_tf, y_train)

#neigh = KNeighborsClassifier(n_jobs=-1)
#parameters = {'n_neighbors':[3, 15, 25, 51, 101]}
#clf = RandomizedSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
#clf.fit(X_tr, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
#print(results.head(5))
results = results.sort_values(['param_alpha'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
alpha = results['param_alpha']
print('Alpha Values before log:', alpha)
# log of alpha values
K = []
for i in alpha:
    K.append(np.log(i))
print('Alpha Values after log:', K)
plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

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

plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```

```
Alpha Values before log: 9     1e-05
7    0.0001
6    0.0005
2     0.005
1      0.05
8       0.5
0         1
5         5
3        50
4       100
Name: param_alpha, dtype: object
Alpha Values after log: [-11.512925464970229, -9.210340371976182, -7.600902459542082, -5.298317366548036, -2.9957
32273553991, -0.6931471805599453, 0.0, 1.6094379124341003, 3.912023005428146, 4.605170185988092]
```



Out[114…

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_test_score	m
9	0.019331	0.000473	0.007656	0.000464	1e-05	{'alpha': 1e-05}	0.617562	0.607446	0.614665	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	{'alpha':	[illegible]	[illegible]	[illegible]	

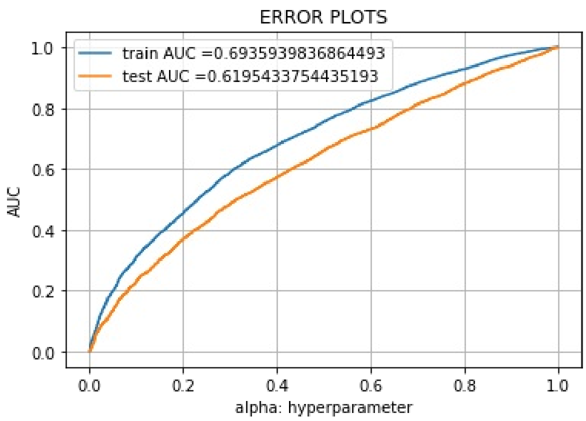
7	0.021182	0.001139	0.008334	0.000474	0.0001	{'alpha': 0.0001}	0.626482	0.614892	0.622878
6	0.019660	0.000944	0.007998	0.000816	0.0005	{'alpha': 0.0005}	0.634877	0.621788	0.630493
2	0.020330	0.001243	0.007990	0.000830	0.005	{'alpha': 0.005}	0.649096	0.633393	0.644074
1	0.020665	0.000941	0.007794	0.000584	0.05	{'alpha': 0.05}	0.658838	0.643108	0.653029

In [117…

```
# best alpha value
best_alpha = 1
```

In [119…

```
# run the model with best alpha and plotting error plot
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


mnb = MultinomialNB(alpha = best_alpha)
mnb.fit(X_tr_tf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(mnb, X_tr_tf)
y_test_pred = batch_predict(mnb, X_te_tf)
#y_train_pred = model.predict_proba(X_tr)
#y_test_pred = model.predict_proba(X_te)

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("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [105…

```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
cm_trn = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
print(cm_trn)
print("Test confusion matrix")
cm_test = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
print(cm_test)
```

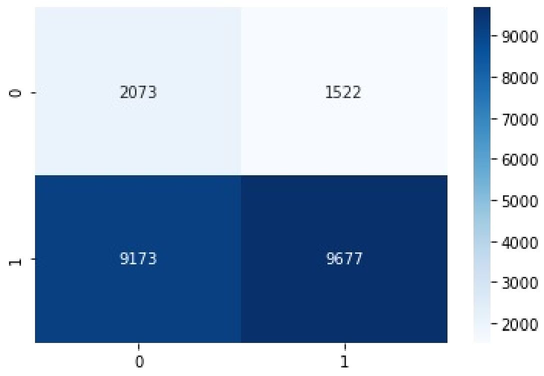
```
the maximum value of tpr*(1-fpr) 0.29602595706533164 for threshold 1.0
Train confusion matrix
[[2073 1522]
 [9173 9677]]
Test confusion matrix
[[1411 1231]
 [6723 7135]]
```

In [106…

```
# https://stackoverflow.com/questions/61748441/how-to-fix-the-values-displayed-in-a-confusion-matrix-in-exponenti
import seaborn as sns
```

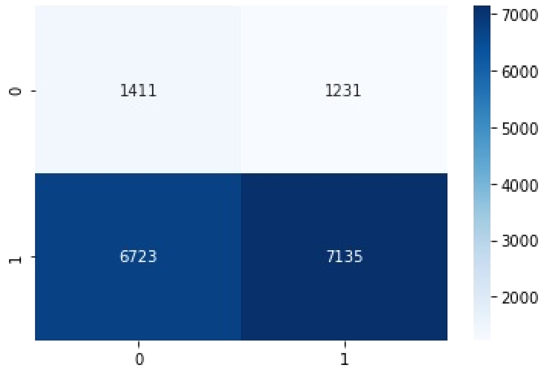
```
import matplotlib.pyplot as plt

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

Out[106… <AxesSubplot:>



In [107…
```
sns.heatmap(cm_test, annot = True, fmt="d", cmap='Blues')
```

Out[107… <AxesSubplot:>



2. Top featurename selection from neg and pos class on Set1

In [157…
```
#log prob of negative class label in sorted order(descending)
neg = np.argsort(mnb.feature_log_prob_[0])
neg[::-1]
#neg.shape
```

Out[157… array([8899, 8900, 8853, ..., 5950, 5951, 2128], dtype=int64)

In [158…
```
#log prob of positive class label in sorted order(descending)
pos = np.argsort(mnb.feature_log_prob_[1])
pos[::-1]
#pos.shape
```

Out[158… array([8899, 8900, 8853, ..., 4834, 8108, 4389], dtype=int64)

In [162…
```
# Merge all features as one list and obtaining top 20 features
# https://gist.github.com/rohan-paul/d51f5547bd7d5e416f792f6333da8163/revisions
from itertools import chain
all_features = list(chain(x1, x2, x3, x4, x5, x6))
all_features.extend(['price', 'teacher_number_of_previously_posted_projects'])
top_20_neg_class = np.take(all_features, neg[0:20])
print('Top 20 Neg Class Labels:', top_20_neg_class)
```

```
Top 20 Neg Class Labels: ['also focus' 'chair table' 'chair students' 'books may' 'attention need'
 'best meet needs' '60 minutes per' 'combining' 'chair classroom'
 'active involved' 'attention deficit disorder' 'attention deficit'
 'complex text' 'bands wobble' 'chromebooks would' 'cold' 'active members'
 'chronic' 'active move' 'cerebral palsy']
```

In [163…

```
top_20_pos_class = np.take(all_features, pos[0:20])
print('Top 20 Pos Class Labels:', top_20_pos_class)
```

```
Top 20 Pos Class Labels: ['best use' 'come work' 'books great' 'challenges school'
 'constantly searching' 'come mostly' 'class feel' 'constant'
 'able research' 'books make' 'art this' 'class first' 'cease'
 'celebrated' '75 students receive' 'child school' 'attach'
 'backgrounds academic' '95 students qualify' 'classrooms give']
```

3. Summary

In [112…

```
column = ['Vectorizer', 'Model', 'Hyperparameter', 'Train_AUC', 'Test_AUC']
```

In [120…

```
row1 = ['BoW', 'NaiveBayes', 5, 0.87, 0.69]
row2 = ['Tfidf', 'NaiveBayes', 5, 0.69, 0.61]
```

In [143…

```
summary = pd.DataFrame(data = [row1, row2], columns = column, index = None)
```

In [145…

```
import tabulate as tb
print(tb.tabulate(summary, headers='keys', tablefmt='psql'))
```

```
+----+--------------+------------+------------------+-------------+------------+
|    | Vectorizer   | Model      |   Hyperparameter |   Train_AUC |   Test_AUC |
|----+--------------+------------+------------------+-------------+------------|
|  0 | BoW          | NaiveBayes |                5 |        0.87 |       0.69 |
|  1 | Tfidf        | NaiveBayes |                5 |        0.69 |       0.61 |
+----+--------------+------------+------------------+-------------+------------+
```

In [ ]:

Processing math: 100%