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.

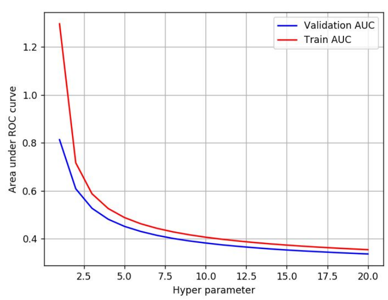
     categorical features

       - teacher_prefix
       - project_grade_category
       - school_state
       - clean_categories
       - clean_subcategories

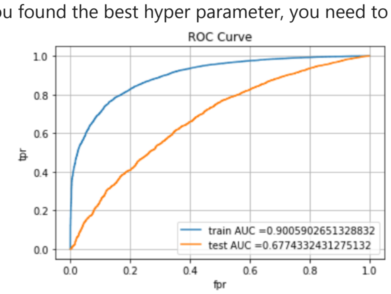
     numerical features

       - 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 $M\underline{t} \in omialNB$ (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

2. Naive Bayes

1.1 Loading Data

In [110...

```
%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
# Tutorial about Python regular expressions: https://pymotw.com/2/re/

import pickle
from tqdm import tqdm
import os
from sklearn import preprocessing

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

In [67]:

```
data  = pd.read_csv('preprocessed_data2.csv', nrows=50000)
data.head(5)
```

Out[67]:

	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcategories	project_title	project_resource_summary	teac
0	mrs	in	grades_prek_2	literacy_language	esl_literacy	Educational Support for English Learners at Home	My students need opportunities to practice beg...	
1	mr	fl	grades_6_8	history_civics_health_sports	civics_government_teamsports	Wanted: Projector for Hungry Learners	My students need a projector to help with view...	
2	ms	az	grades_6_8	health_sports	health_wellness_teamsports	Soccer Equipment for AWESOME Middle School Stu...	My students need shine guards, athletic socks,...	
3	mrs	ky	grades_prek_2	literacy_language_math_science	literacy_mathematics	Techie Kindergarteners	My students need to engage in Reading and Math...	
4	mrs	tx	grades_prek_2	math_science	mathematics	Interactive Math Tools	My students need hands on practice in mathemat...	

In [68]:
```
print("Number of data points in train data", data.shape)
print('-'*50)
print("The attributes of data :", data.columns.values)
```

```
Number of data points in train data (50000, 11)
--------------------------------------------------
The attributes of data : ['teacher_prefix' 'school_state' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved'
 'essay' 'price']
```

In [69]:
```
# check if we have any nan values are there
print(data['project_title'].isnull().values.any())
print("number of nan values",data['project_title'].isnull().values.sum())
```

```
False
number of nan values 0
```

In [70]:
```
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
X.head(1)
```

Out[70]:

	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcategories	project_title	project_resource_summary	teacher_nu
0	mrs	in	grades_prek_2	literacy_language	esl_literacy	Educational Support for English Learners at Home	My students need opportunities to practice beg...	

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

In [71]:
```
# train test split
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, and project_title

In [72]:
```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)


vectorizer_1 = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer_1.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_essay_bow = vectorizer_1.transform(X_train['essay'].values)
X_cv_essay_bow = vectorizer_1.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer_1.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)
print("="*100)
```

```
(22445, 10) (22445,)
(11055, 10) (11055,)
(16500, 10) (16500,)
====================================================================================================
After vectorizations
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
====================================================================================================
```

In [73]:
```
vectorizer_2 = CountVectorizer(max_features=500)
vectorizer_2.fit(X_train['project_title'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_bow = vectorizer_2.transform(X_train['project_title'].values)
X_cv_title_bow = vectorizer_2.transform(X_cv['project_title'].values)
X_test_title_bow = vectorizer_2.transform(X_test['project_title'].values)

print("After vectorizations")
print(X_train_title_bow.shape, y_train.shape)
print(X_cv_title_bow.shape, y_cv.shape)
print(X_test_title_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
```

```
(22445, 500) (22445,)
(11055, 500) (11055,)
(16500, 500) (16500,)
====================================================================================================
```

1.4 Make Data Model Ready: encoding numerical, categorical features

1.4.1 encoding categorical features: School State

In [74]:

```
vectorizer_3 = CountVectorizer()
vectorizer_3.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 = vectorizer_3.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer_3.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer_3.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)
```

```
After vectorizations
(22445, 51) (22445,)
(11055, 51) (11055,)
(16500, 51) (16500,)
```

1.4.2 encoding categorical features: teacher_prefix

In [75]:

```
vectorizer_4 = CountVectorizer()
vectorizer_4.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer_4.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = vectorizer_4.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer_4.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)
```

```
After vectorizations
(22445, 4) (22445,)
(11055, 4) (11055,)
(16500, 4) (16500,)
```

1.4.3 encoding categorical features: project_grade_category

In [76]:

```
vectorizer_5 = CountVectorizer()
vectorizer_5.fit(X_train['project_grade_category'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer_5.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vectorizer_5.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer_5.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)
```

```
After vectorizations
(22445, 4) (22445,)
(11055, 4) (11055,)
(16500, 4) (16500,)
```

1.4.4 encoding categorical features: project_subject_categories

In [77]:

```
vectorizer_6 = CountVectorizer()
vectorizer_6.fit(X_train['project_subject_categories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_subject_categories = vectorizer_6.transform(X_train['project_subject_categories'].values)
X_cv_subject_categories = vectorizer_6.transform(X_cv['project_subject_categories'].values)
X_test_subject_categories = vectorizer_6.transform(X_test['project_subject_categories'].values)

print("After vectorizations")
print(X_train_subject_categories.shape, y_train.shape)
print(X_cv_subject_categories.shape, y_cv.shape)
print(X_test_subject_categories.shape, y_test.shape)
```

```
After vectorizations
(22445, 50) (22445,)
(11055, 50) (11055,)
(16500, 50) (16500,)
```

1.4.5 encoding categorical features: project_subject_subcategories

In [78]:

```
vectorizer_7 = CountVectorizer(max_features=100)
```

```
vectorizer_7.fit(X_train['project_subject_subcategories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_subject_subcategories = vectorizer_7.transform(X_train['project_subject_subcategories'].values)
X_cv_subject_subcategories = vectorizer_7.transform(X_cv['project_subject_subcategories'].values)
X_test_subject_subcategories = vectorizer_7.transform(X_test['project_subject_subcategories'].values)

print("After vectorizations")
print(X_train_subject_subcategories.shape, y_train.shape)
print(X_cv_subject_subcategories.shape, y_cv.shape)
print(X_test_subject_subcategories.shape, y_test.shape)
```

```
After vectorizations
(22445, 100) (22445,)
(11055, 100) (11055,)
(16500, 100) (16500,)
```

1.4.6 encoding numerical features: Price

In [79]:

```
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("="*100)
```

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

In [80]:

```
#we are defining this function to return arrray
def to_array(a):
  b = a.tolist()
  c = []
  for i in b:
    for  j in i:
      c.append(j)
  d = [i for i in range(len(c))]
  df = pd.DataFrame(list(zip(d, c)),columns = ['1','2'])
  e = df.drop(['1'], axis=1)
  array = e.to_numpy()

  return array
```

In [81]:

```
X_train_price_norm_array = to_array(X_train_price_norm)
X_cv_price_norm_array = to_array(X_cv_price_norm)
X_test_price_norm_array = to_array(X_test_price_norm)
```

1.4.7 Concatinating all the features

In [82]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_bow,X_train_title_bow, X_train_state_ohe,X_train_subject_categories,X_train_subject_subcategories, X_train_teac
X_cr = hstack((X_cv_essay_bow,X_cv_title_bow, X_cv_state_ohe,X_cv_subject_categories,X_cv_subject_subcategories ,X_cv_teacher_ohe, X_cv_grad
X_te = hstack((X_test_essay_bow,X_test_title_bow, X_test_state_ohe,X_test_subject_categories,X_test_subject_subcategories, X_test_teacher_oh

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)
print("="*100)
```

```
Final Data matrix
(22445, 5709) (22445,)
(11055, 5709) (11055,)
(16500, 5709) (16500,)
====================================================================================================
```

In [83]:

```
X_tr1 = X_tr.toarray()
#Normalize Data
X_tr1 = preprocessing.normalize(X_tr1)
X_tr2 = np.concatenate((X_tr1, X_train_price_norm_array), axis=1)
```

```
X_cr1 = X_cr.toarray()
#Normalize Data
X_cr1 = preprocessing.normalize(X_cr1)
X_cr2 = np.concatenate((X_cr1, X_cv_price_norm_array), axis=1)


X_te1 = X_te.toarray()
#Normalize Data
X_te1 = preprocessing.normalize(X_te1)
X_te2 = np.concatenate((X_te1, X_test_price_norm_array), axis=1)
```

In [84]:

```
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
```

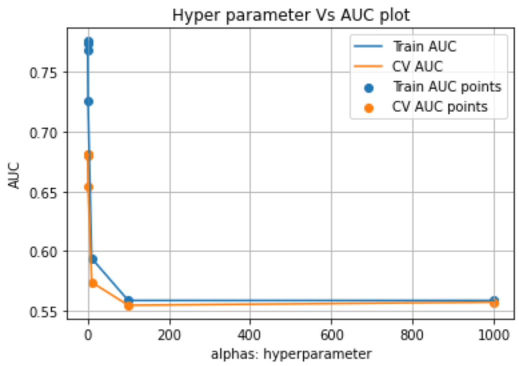
In [85]:

```
def Confusion_matrix(y_test, test_pred):
  # Confusion matrix for test data
  plt.figure()
  cm = confusion_matrix(y_test, test_pred)
  class_label = ["negative", "positive"]
  df_cm_test = pd.DataFrame(cm, index = class_label, columns = class_label)
  sns.heatmap(df_cm_test , annot = True, fmt = "d")
  plt.title("Confusiion Matrix for test data")
  plt.xlabel("Predicted Label")
  plt.ylabel("True Label")
  plt.show()
```

In [86]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
from sklearn.naive_bayes import MultinomialNB

NB = MultinomialNB()
#parameters = {'alphas':[0.0001*pow(10,i) for i in range(8)]}
alphas =  np.array([0.0001*pow(10,i) for i in range(8)])
parameters = {'alpha':alphas}
clf = RandomizedSearchCV(NB, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_tr2, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)

alphas_li = [row['params']['alpha'] for i, row in results.iterrows()]

results = results.drop(['params'], axis=1)


alphas_df= pd.DataFrame({'params':alphas_li})

frames = [results, alphas_df]

results = pd.concat(frames,axis=1)

results = results.sort_values(by = ['params'])

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']
alphas =  results['params']

plt.plot(alphas, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.plot(alphas, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

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


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

```
plt.show()

results.head()
```



Out[86]:

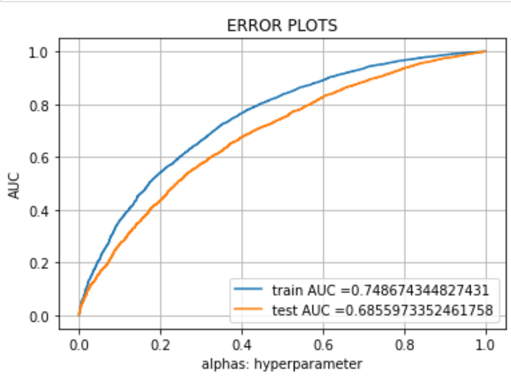
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	split0_test_score	split1_test_score	split2_test_score	mean_test_score	std_test_score
0	0.947855	0.545992	0.124796	0.001595	0.0001	0.681987	0.675024	0.686037	0.681016	0.004548
1	0.546391	0.002946	0.125135	0.002679	0.001	0.682472	0.675513	0.686167	0.681384	0.004417
2	0.543596	0.014811	0.125898	0.006619	0.01	0.682656	0.675740	0.686108	0.681501	0.004311
3	0.552372	0.008895	0.122491	0.001633	0.1	0.680470	0.673429	0.684053	0.679317	0.004413
4	0.553236	0.006184	0.124252	0.002827	1	0.655009	0.646988	0.661556	0.654518	0.005958

1.5.1.2 Testing the performance of the model on test data, plotting ROC Curves

In [89]:
```
#here we are choosing the best_alpha based on forloop results
best_alpha = clf.best_params_['alpha']
print(f'Best alpha is found to be {best_alpha}')
```

```
Best alpha is found to be 0.01
```

In [91]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

NB = MultinomialNB(alpha = best_alpha)
#neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
NB.fit(X_tr2, 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(NB, X_tr2)
y_test_pred = batch_predict(NB, X_te2)

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



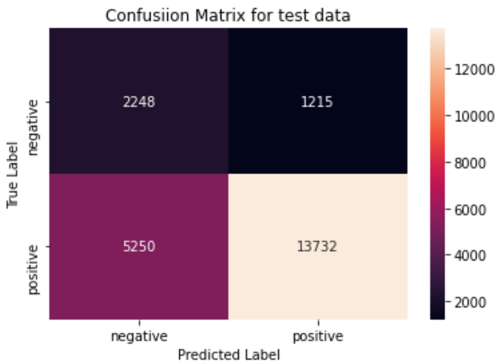
In [93]:
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very 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
```

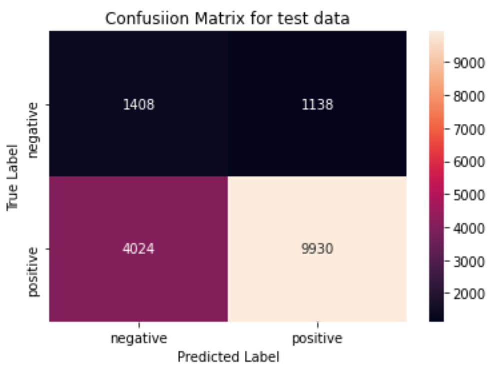
In [94]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
Confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
print("Test confusion matrix")
Confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.46960816686890905 for threshold 0.84
Train confusion matrix
```




```
Test confusion matrix
```



In [95]:

```
list_vectorizer = [vectorizer_1,vectorizer_2,vectorizer_3,vectorizer_4,vectorizer_5,vectorizer_6,vectorizer_7]
def top_20_features(transform,X_train,y_train,optimal_alpha):
  feature_name = []
  for i in transform:
    feature_name.extend(i.get_feature_names())

  feature_name.append('price')


  NB_optimal = MultinomialNB(alpha =optimal_alpha)
  NB_optimal.fit(X_train, y_train)

  # it gives Empirical log probability of features given a class (P(x_i|y))

  log_probability = NB_optimal.feature_log_prob_

  feature_probability_table = pd.DataFrame(log_probability, columns = feature_name)
  feature_prob_transpose = feature_probability_table.T

  print("Top 20 Negative Features:-\n",feature_prob_transpose[0].sort_values(ascending = False)[0:20])
  print("\n Top 20 Positive Features:-\n",feature_prob_transpose[1].sort_values(ascending = False)[0:20])
```

In [96]:

```
top_20_features(list_vectorizer,X_tr2,y_train,0.01)
```

```
Top 20 Negative Features:-
 to            -3.494245
and           -3.652191
the           -3.774983
students      -3.974735
of            -4.214012
in            -4.275673
my            -4.454045
are           -4.538571
they          -4.590924
```

```
their         -4.734337
will          -4.767109
that          -4.893931
for           -4.918809
is            -4.950923
have          -4.958382
my students   -4.971422
our           -4.980725
with          -4.983092
school        -5.037744
be            -5.175380
Name: 0, dtype: float64

 Top 20 Positive Features:-
 to            -3.497089
and           -3.654455
the           -3.757295
students      -3.945646
of            -4.223413
in            -4.310500
my            -4.450232
are           -4.561720
they          -4.622291
their         -4.720303
will          -4.730224
for           -4.868336
our           -4.928279
my students   -4.959564
that          -4.964342
is            -4.970082
have          -4.974108
with          -5.008706
school        -5.069478
be            -5.161296
Name: 1, dtype: float64
```

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

1.3 Make Data Model Ready: encoding eassay, and project_title with Tfidf as vectorizer

In [97]:

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

print("="*100)

vectorizer_1 = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer_1.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_essay_bow = vectorizer_1.transform(X_train['essay'].values)
X_cv_essay_bow = vectorizer_1.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer_1.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)
print("="*100)
```

```
(22445, 10) (22445,)
(11055, 10) (11055,)
(16500, 10) (16500,)
====================================================================================================
After vectorizations
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
====================================================================================================
```

In [98]:

```
vectorizer_2 = TfidfVectorizer(max_features=500)
vectorizer_2.fit(X_train['project_title'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_bow = vectorizer_2.transform(X_train['project_title'].values)
X_cv_title_bow = vectorizer_2.transform(X_cv['project_title'].values)
X_test_title_bow = vectorizer_2.transform(X_test['project_title'].values)

print("After vectorizations")
print(X_train_title_bow.shape, y_train.shape)
print(X_cv_title_bow.shape, y_cv.shape)
print(X_test_title_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 500) (22445,)
(11055, 500) (11055,)
```

```
(16500, 500) (16500,)
====================================================================================================
```

1.4.7 Concatinating all the features

In [99]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_bow,X_train_title_bow, X_train_state_ohe,X_train_subject_categories,X_train_subject_subcategories, X_train_teac
X_cr = hstack((X_cv_essay_bow,X_cv_title_bow, X_cv_state_ohe,X_cv_subject_categories,X_cv_subject_subcategories ,X_cv_teacher_ohe, X_cv_grad
X_te = hstack((X_test_essay_bow,X_test_title_bow, X_test_state_ohe,X_test_subject_categories,X_test_subject_subcategories, X_test_teacher_oh

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)
print("="*100)
```

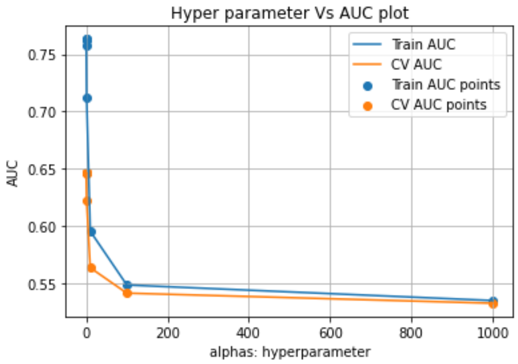
```
Final Data matrix
(22445, 5709) (22445,)
(11055, 5709) (11055,)
(16500, 5709) (16500,)
====================================================================================================
```

In [100…

```
X_tr1 = X_tr.toarray()
#Normalize Data
X_tr1 = preprocessing.normalize(X_tr1)
X_tr2 = np.concatenate((X_tr1, X_train_price_norm_array), axis=1)


X_cr1 = X_cr.toarray()
#Normalize Data
X_cr1 = preprocessing.normalize(X_cr1)
X_cr2 = np.concatenate((X_cr1, X_cv_price_norm_array), axis=1)


X_te1 = X_te.toarray()
#Normalize Data
X_te1 = preprocessing.normalize(X_te1)
X_te2 = np.concatenate((X_te1, X_test_price_norm_array), axis=1)
```

In [101…

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV

NB = MultinomialNB()
alphas =  np.array([0.0001*pow(10,i) for i in range(8)])
parameters = {'alpha':alphas}
clf = RandomizedSearchCV(NB, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_tr2, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)

alphas_li = [row['params']['alpha'] for i, row in results.iterrows()]

results = results.drop(['params'], axis=1)

alphas_df= pd.DataFrame({'params':alphas_li})

frames = [results, alphas_df]

results = pd.concat(frames,axis=1)

results = results.sort_values(by = ['params'])

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']
alphas =  results['params']

plt.plot(alphas, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.plot(alphas, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

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


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

results.head()
```



Out[101...

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	split0_test_score	split1_test_score	split2_test_score	mean_test_score	std_test_score
0	0.611533	0.016319	0.124663	0.001503	0.0001	0.652172	0.633897	0.652875	0.646315	0.008785
1	0.591540	0.005644	0.123307	0.000909	0.001	0.652421	0.633950	0.652850	0.646407	0.008810
2	0.590564	0.000622	0.124295	0.001074	0.01	0.653011	0.633945	0.652633	0.646529	0.008900
3	0.591205	0.006442	0.122157	0.000440	0.1	0.652245	0.632320	0.650585	0.645050	0.009027
4	0.593820	0.007279	0.125441	0.004302	1	0.628957	0.611316	0.625949	0.622074	0.007705

1.5.1.2 Testing the performance of the model on test data, plotting ROC Curves

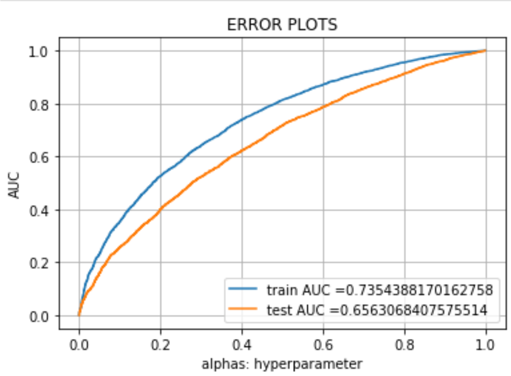
In [102...

```
#here we are choosing the best_alpha based on forloop results
best_alpha = clf.best_params_['alpha']
print(f'Best alpha found to be {best_alpha}')
```

```
Best alpha found to be 0.01
```

In [104...

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

NB = MultinomialNB(alpha = best_alpha)
#neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
NB.fit(X_tr2, 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(NB, X_tr2)
y_test_pred = batch_predict(NB, X_te2)

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



In [105...

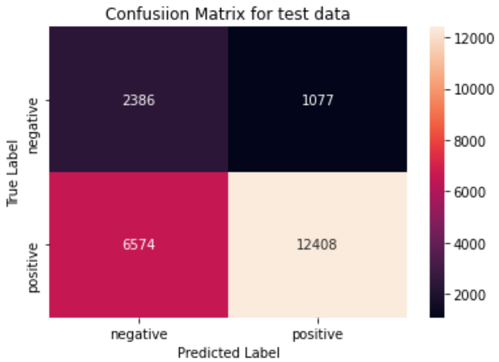
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    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
```

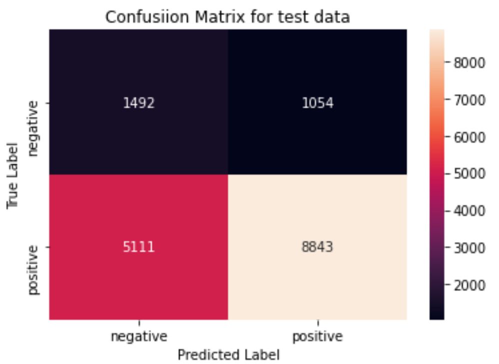
In [106…

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
Confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
print("Test confusion matrix")
Confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.4503786175775199 for threshold 0.847
Train confusion matrix
```




```
Test confusion matrix
```



In [109…

```
top_20_features(list_vectorizer,X_tr2,y_train,0.01)
```

```
Top 20 Negative Features:-
 performingarts                    -3.696321
warmth_care_hunger                 -3.902569
socialsciences                     -3.994793
specialneeds_visualarts            -4.146465
health_sports_warmth_care_hunger   -4.645724
literacy_language                  -4.755141
teamsports                         -4.806686
to                                 -4.892712
ca                                 -4.973409
and                                -5.051434
history_civics_literacy_language   -5.128339
the                                -5.165518
visualarts                         -5.209471
other_specialneeds                 -5.296020
appliedlearning                    -5.364747
students                           -5.367749
tx                                 -5.476709
of                                 -5.601691
in                                 -5.664642
health_wellness_teamsports         -5.675108
Name: 0, dtype: float64

 Top 20 Positive Features:-
 performingarts                    -3.664754
warmth_care_hunger                 -3.929461
socialsciences                     -4.040411
specialneeds_visualarts            -4.091937
health_sports_warmth_care_hunger   -4.503347
teamsports                         -4.881043
to                                 -4.895322
ca                                 -4.931833
literacy_language                  -4.934782
history_civics_literacy_language   -5.015992
and                                -5.052487
the                                -5.148065
```

```
visualarts                         -5.283528
students                           -5.338931
other_specialneeds                 -5.348073
appliedlearning                    -5.368277
health_wellness_other              -5.387831
health_wellness_teamsports         -5.567072
of                                 -5.611063
ny                                 -5.699153
Name: 1, dtype: float64
```

3. Summary

as mentioned in the step 5 of instructions

In [108…

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Hyper parameter", "AUC"]

x.add_row(["BOW", "NaiveBayes", 0.01, 0.685])
x.add_row(["TFIDF", "NaiveBayes", 0.01, 0.656])

print(x)
```

```
+------------+------------+-----------------+-------+
| Vectorizer |   Model    | Hyper parameter |  AUC  |
+------------+------------+-----------------+-------+
|    BOW     | NaiveBayes |       0.01      | 0.685 |
|   TFIDF    | NaiveBayes |       0.01      | 0.656 |
+------------+------------+-----------------+-------+
```