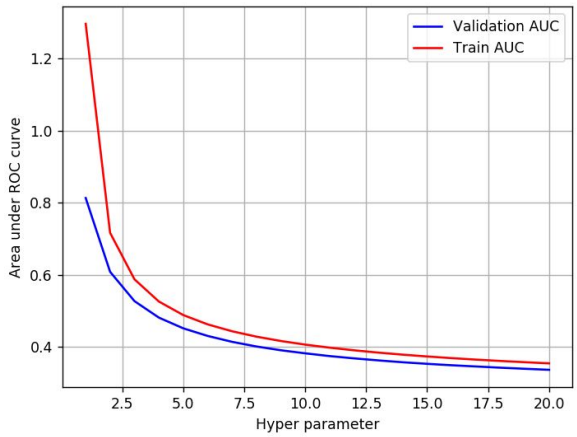
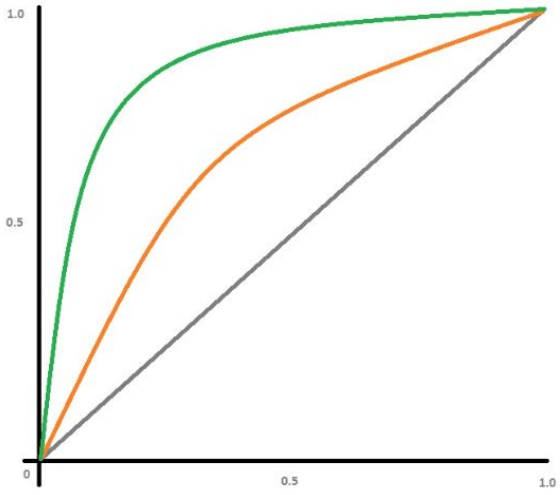
Assignment 6: Apply NB

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



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



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

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

4. fine the top 20 features from either from feature Set 1 or feature Set 2 using absolute values of `feature_log_prob_` parameter of `MultinomialNB` (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html (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]:
```
import pandas
data = pandas.read_csv('preprocessed_data.csv')
# data.shape
print(data.columns)
data.head(2)
```

```
Index(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'price'],
      dtype='object')
```

Out[1]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_project
0	ca	mrs	grades_prek_2	5
1	ut	ms	grades_3_5	

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

In [2]:
```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer

X = data.drop(['project_is_approved'],axis = 1)
Y = data[['project_is_approved']]

X_train,X_rem,Y_train,Y_rem = train_test_split(X,Y,test_size = 0.40,stratify =
X_cv,X_test,Y_cv,Y_test = train_test_split(X_rem,Y_rem,test_size = 0.50,strati
X.shape,X_train.shape,X_cv.shape,X_test.shape
```

Out[2]: ((109248, 8), (65548, 8), (21850, 8), (21850, 8))

1.3 Make Data Model Ready: encoding eassay, and project_title

In [3]:
```
from sklearn.feature_extraction.text import CountVectorizer

print(X_train.shape, Y_train.shape)
print(X_cv.shape, Y_cv.shape)
print(X_test.shape, Y_test.shape)

vectorizer = CountVectorizer(min_df = 10, ngram_range=(1,4), max_features= 500
vectorizer.fit(X_train['essay'].values)
```

```
(65548, 8) (65548, 1)
(21850, 8) (21850, 1)
(21850, 8) (21850, 1)
```

Out[3]:
```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=5000, min_df=10,
        ngram_range=(1, 4), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None)
```

In [5]:
```
X_train_essay_bow = vectorizer.transform(X_train['essay'].values)
X_test_essay_bow = vectorizer.transform(X_test['essay'].values)
X_cv_essay_bow = vectorizer.transform(X_cv['essay'].values)
X_essay_features = vectorizer.get_feature_names()

print(X_train_essay_bow.shape)
print(X_test_essay_bow.shape)
print(X_cv_essay_bow.shape)
```

```
(65548, 5000)
(21850, 5000)
(21850, 5000)
```

1.4 Make Data Model Ready: encoding numerical, categorical features

In [6]:
```
# encoding categorical features: School State

vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on tra

# 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)
X_state_features = vectorizer.get_feature_names()

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

```
After vectorizations
(65548, 51) (65548, 1)
(21850, 51) (21850, 1)
(21850, 51) (21850, 1)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'i
a', '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']
====================================================================================
======================
```

In [7]:
```
# encoding categorical features: teacher_prefix

vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on t

# 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)
X_teacher_features = vectorizer.get_feature_names()

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

```
After vectorizations
(65548, 5) (65548, 1)
(21850, 5) (21850, 1)
(21850, 5) (21850, 1)
['dr', 'mr', 'mrs', 'ms', 'teacher']
====================================================================================
======================
```

In [9]:
```
# encoding categorical features: clean_categories

vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on
# print(vectorizer.get_feature_names())
# we use the fitted CountVectorizer to convert the text to vector
X_train_category_ohe = vectorizer.transform(X_train['clean_categories'].values
X_cv_category_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_category_ohe = vectorizer.transform(X_test['clean_categories'].values)
X_category_features = vectorizer.get_feature_names()

print("After vectorizations")
print(X_train_category_ohe.shape, Y_train.shape)
print(X_cv_category_ohe.shape, Y_cv.shape)
print(X_test_category_ohe.shape, Y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(65548, 9) (65548, 1)
(21850, 9) (21850, 1)
(21850, 9) (21850, 1)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'litera
cy_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
====================================================================================
======================
```

In [10]:
```
# encoding categorical features: clean_subcategories

vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only
# print(vectorizer.get_feature_names())
# we use the fitted CountVectorizer to convert the text to vector
X_train_subcategory_ohe = vectorizer.transform(X_train['clean_subcategories'].
X_cv_subcategory_ohe = vectorizer.transform(X_cv['clean_subcategories'].values
X_test_subcategory_ohe = vectorizer.transform(X_test['clean_subcategories'].va
X_subcategory_features = vectorizer.get_feature_names()

print("After vectorizations")
print(X_train_subcategory_ohe.shape, Y_train.shape)
print(X_cv_subcategory_ohe.shape, Y_cv.shape)
print(X_test_subcategory_ohe.shape, Y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(65548, 30) (65548, 1)
(21850, 30) (21850, 1)
(21850, 30) (21850, 1)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'e
nvironmentalscience', 'esl', 'extracurricular', 'financialliteracy', 'foreign
languages', 'gym_fitness', 'health_lifescience', 'health_wellness', 'history_
geography', 'literacy', 'literature_writing', 'mathematics', 'music', 'nutrit
ioneducation', 'other', 'parentinvolvement', 'performingarts', 'socialscience
s', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
==========================================================================
======================
```

In [11]:
```
# encoding categorical features: project_grade_category

vectorizer = CountVectorizer()

vectorizer.fit(X_train['project_grade_category'].values) #

vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen o

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].val
X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].value
X_grade_features = vectorizer.get_feature_names()

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

```
After vectorizations
(65548, 4) (65548, 1)
(21850, 4) (21850, 1)
(21850, 4) (21850, 1)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
====================================================================================
======================
```

In [13]:
```
# encoding numerical features: Price

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)).res
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1))
X_price_features = ['price']


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
(65548, 1) (65548, 1)
(21850, 1) (21850, 1)
(21850, 1) (21850, 1)
====================================================================================
======================
```

1.5 Concatinating all the features

In [17]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_bow, X_train_state_ohe, X_train_teacher_ohe, X_tr
X_cr = hstack((X_cv_essay_bow, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_oh
X_te = hstack((X_test_essay_bow, X_test_state_ohe, X_test_teacher_ohe, X_test_
X_feature = X_essay_features + X_state_features + X_teacher_features + X_grade

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('Feature size:',len(X_feature))
print("="*100)
```

```
Final Data matrix
(65548, 5100) (65548, 1)
(21850, 5100) (21850, 1)
(21850, 5100) (21850, 1)
Feature size: 5100
====================================================================================
======================
```

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

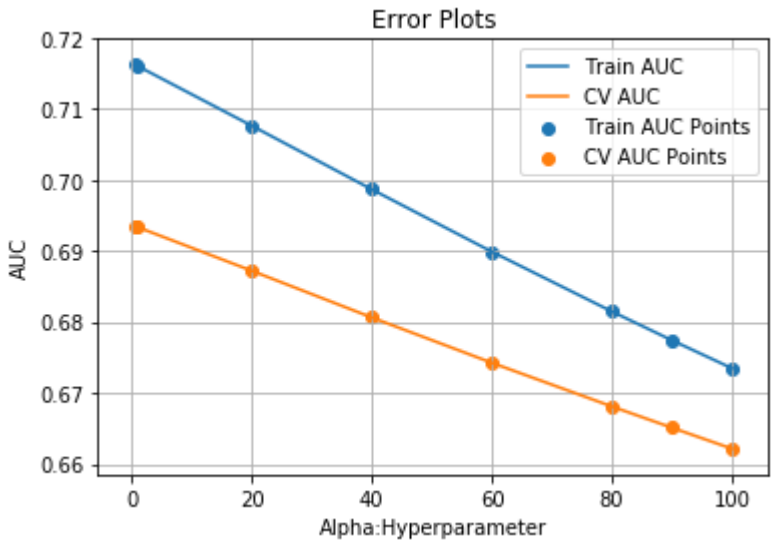
Apply NB on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

1.5.1 Appling NB: BOW featurization

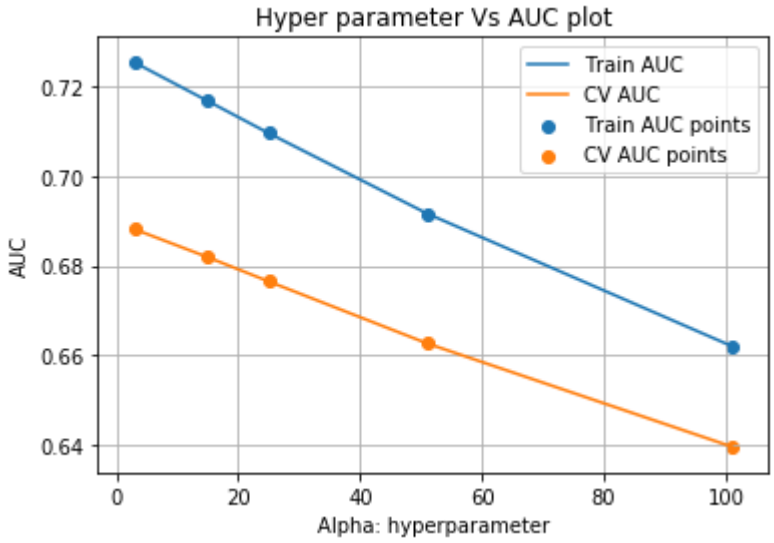
In [18]:
```
def batch_predict(cfg,data):
    loop = data.shape[0]-data.shape[0]%1000
    y_data_predict = []
    for i in range(0,loop,1000):
        y_data_predict.extend(cfg.predict_proba(data[i:i+1000])[:,1])
    if data.shape[0]%1000 !=0:
        y_data_predict.extend(cfg.predict_proba(data[loop:])[:,1])

    return y_data_predict
```

In [19]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from tqdm import tqdm

train_auc = []
cv_auc = []
K = [0.5,1,20,40,60,80,90,100]
# K = [0.1,0.3,0.5,0.7,0.9,1]
for i in tqdm(K):
    model = MultinomialNB(alpha = i)
    model.fit(X_tr,Y_train)

    Y_train_predict = batch_predict(model,X_tr)
    Y_cv_predict = batch_predict(model,X_cr)

    train_auc.append(roc_auc_score(Y_train,Y_train_predict))
    cv_auc.append(roc_auc_score(Y_cv,Y_cv_predict))

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('Error Plots')
plt.grid()
plt.show()
```

```
100%|██████████| 8/8 [00:04<00:00,  1.94it/s]
```



In [20]:
```
from sklearn.metrics import roc_curve, auc
import pandas as pd

model_bow = MultinomialNB()
parameters = {'alpha':[3, 15, 25, 51, 101]}

clf_bow = GridSearchCV(model_bow,parameters,cv = 3,n_jobs = -1,scoring = 'roc_
clf_bow.fit(X_tr,Y_train)

results = pd.DataFrame.from_dict(clf_bow.cv_results_)
# print(results.head())
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']

plt.plot(alpha, 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_st

plt.plot(alpha, 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,

plt.scatter(alpha, train_auc, label='Train AUC points')
plt.scatter(alpha, 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()
```

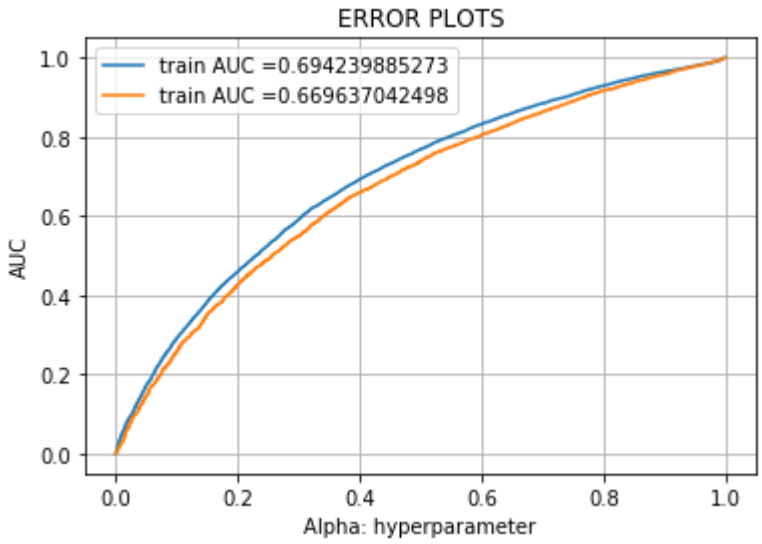


Out[20]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_alpha	params	rar
0	0.604672	0.069666	0.688104	0.725324	3	{'alpha': 3}	
1	0.598672	0.057000	0.681862	0.716727	15	{'alpha': 15}	
2	0.604004	0.068002	0.676453	0.709539	25	{'alpha': 25}	
3	0.593338	0.064333	0.662612	0.691598	51	{'alpha': 51}	
4	0.527003	0.035668	0.639552	0.661985	101	{'alpha': 101}	

In [21]:
```
best_alpha = 50
```

In [22]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.
from sklearn.metrics import roc_curve, auc

best_model_bow = MultinomialNB(alpha = best_alpha)
best_model_bow.fit(X_tr,Y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estim
# not the predicted outputs

y_train_pred = batch_predict(best_model_bow, X_tr)
y_test_pred = batch_predict(best_model_bow, 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_tp
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 [23]:
```
import numpy as np

# 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 hi
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold
    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 [24]:
```
from sklearn.metrics import confusion_matrix

best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(Y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(Y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 0.420612470667 for threshold 0.985
Train confusion matrix
[[ 6735  3190]
 [21146 34477]]
Test confusion matrix
[[ 2147  1162]
 [ 7199 11342]]
```

1.6 Applying Naive Bayes: TfIDF featurization

1.6.1 Make Data Model Ready: encoding eassay, and project_title

In [27]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

print(X_train.shape, Y_train.shape)
print(X_cv.shape, Y_cv.shape)
print(X_test.shape, Y_test.shape)

vectorizer = TfidfVectorizer(min_df = 10, ngram_range=(1,4), max_features= 500
vectorizer.fit(X_train['essay'].values)
```

```
(65548, 8) (65548, 1)
(21850, 8) (21850, 1)
(21850, 8) (21850, 1)
```

Out[27]:
```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=5000, min_df=10,
        ngram_range=(1, 4), norm='l2', preprocessor=None, smooth_idf=True,
        stop_words=None, strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None)
```

In [28]:
```
X_train_essay_tfidf = vectorizer.transform(X_train['essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv['essay'].values)

print(X_train_essay_tfidf.shape)
print(X_test_essay_tfidf.shape)
print(X_cv_essay_tfidf.shape)
```

```
(65548, 5000)
(21850, 5000)
(21850, 5000)
```

1.6.2 Concatinating all the features

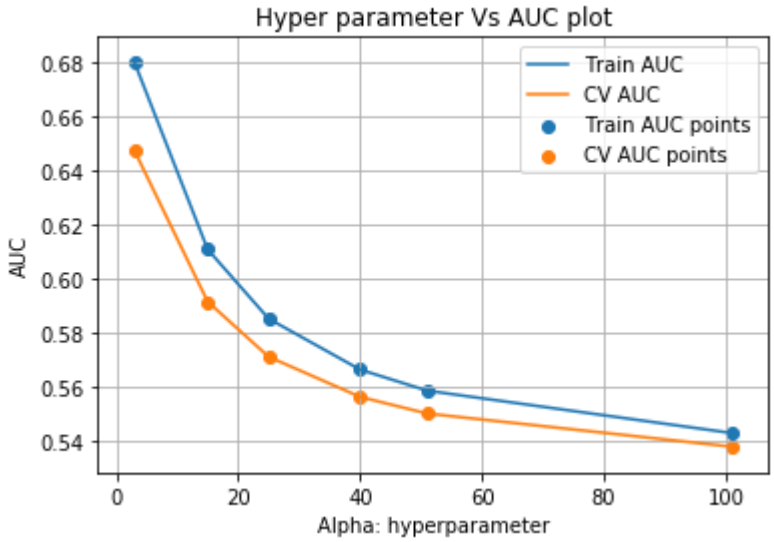
In [30]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039

X_tr_tfidf = hstack((X_train_essay_tfidf, X_train_state_ohe, X_train_teacher_o
X_cr_tfidf = hstack((X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_
X_te_tfidf = hstack((X_test_essay_tfidf, X_test_state_ohe, X_test_teacher_ohe,

print("Final Data matrix")
print(X_tr_tfidf.shape, Y_train.shape)
print(X_tr_tfidf.shape, Y_cv.shape)
print(X_tr_tfidf.shape, Y_test.shape)
print("="*100)
```

```
Final Data matrix
(65548, 5100) (65548, 1)
(65548, 5100) (21850, 1)
(65548, 5100) (21850, 1)
====================================================================================
======================
```

1.6.3 Appling NB: Tfidf featurization

In [31]:

```
model_tfidf = MultinomialNB()
parameters = {'alpha':[3, 15, 25, 40, 51, 101]}

clf_idf = GridSearchCV(model_tfidf,parameters,cv = 3,n_jobs = -1,scoring = 'ro
clf_idf.fit(X_tr_tfidf,Y_train)

results = pd.DataFrame.from_dict(clf_idf.cv_results_)
# print(results.head())
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']

plt.plot(alpha, 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_st

plt.plot(alpha, 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,

plt.scatter(alpha, train_auc, label='Train AUC points')
plt.scatter(alpha, 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()
```

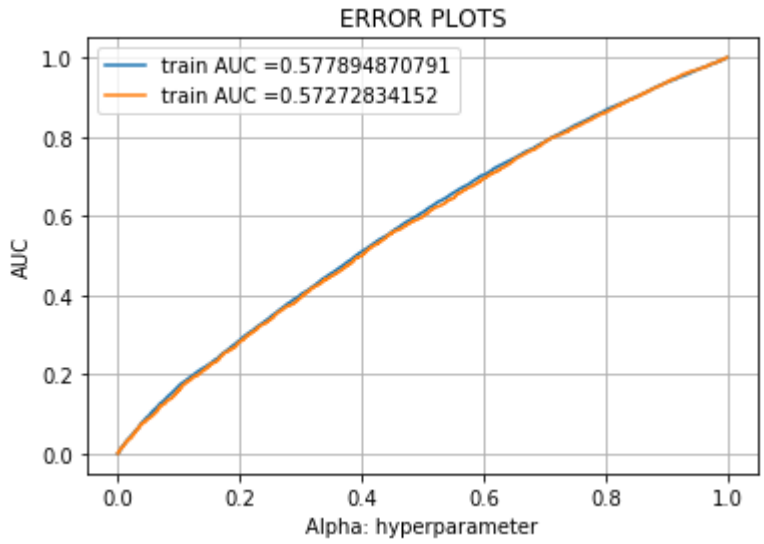


Out[31]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_alpha	params
0	0.667671	0.066001	0.647241	0.679978	3	{'alpha': 3}
1	0.645004	0.055666	0.591349	0.610700	15	{'alpha': 15}
2	0.769671	0.124002	0.570917	0.585029	25	{'alpha': 25}
3	0.875340	0.081334	0.556050	0.566161	40	{'alpha': 40}
4	0.734672	0.062667	0.549897	0.558366	51	{'alpha': 51}

In [32]:
```
best_alpha = 40
```

In [33]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.
from sklearn.metrics import roc_curve, auc

best_model_tfidf = MultinomialNB(alpha = best_alpha)
best_model_tfidf.fit(X_tr_tfidf,Y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estim
# not the predicted outputs

y_train_pred_tfidf = batch_predict(best_model_tfidf, X_tr_tfidf)
y_test_pred_tfidf = batch_predict(best_model_tfidf, X_te_tfidf)

train_fpr_tfidf, train_tpr_tfidf, tr_thresholds_tfidf = roc_curve(Y_train, y_t
test_fpr_tfidf, test_tpr_tfidf, te_thresholds_tfidf = roc_curve(Y_test, y_test

plt.plot(train_fpr_tfidf, train_tpr_tfidf, label="train AUC ="+str(auc(train_f
plt.plot(test_fpr_tfidf, test_tpr_tfidf, label="test AUC ="+str(auc(test_fpr_t
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



3. Summary

as mentioned in the step 5 of instructions

3.1 Top 20 Features of Set 1

In [76]:
```
neg_class_prob_sorted = abs(best_model_bow.feature_log_prob_[0, :]).argsort()[
pos_class_prob_sorted = abs(best_model_bow.feature_log_prob_[1, :]).argsort()

print('Printing top 20 Feature for Negative Class:\n', np.take(X_feature, neg_
print()
print('Printing top 20 Feature for Positive Class:\n',np.take(X_feature, pos_c
```

```
Printing top 20 Feature for Negative Class:
 ['dr' 'nd' 'vt' 'wy' 'chromebooks allow' 'de' 'these stools'
 'these chromebooks' 'the wobble' 'stools allow' 'graphing' 'balance balls'
 'reluctant readers' 'chairs allow' 'subscription' 'sturdy' 'bouncy bands'
 'listen stories' 'core muscles' 'sd']

Printing top 20 Feature for Positive Class:
 ['students' 'school' 'my' 'learning' 'classroom' 'the' 'they' 'not'
 'my students' 'learn' 'help' 'many' 'nannan' 'we' 'work' 'reading' 'need'
 'use' 'love' 'day']
```

3.2 Tabular Format

In [112]:
```
# https://pythonmatplotlibtips.blogspot.com/2018/11/matplotlib-only-table.html

fig = plt.figure()
ax = fig.add_subplot(111)

col_labels = ['Model','Hyper-Para','AUC']
row_labels = ['BOW','TfIDF']
table_vals = [['Brute', 50, '0.6690'],['Brute', 40, 0.5727]]
# Draw table
the_table = plt.table(cellText=table_vals,
                      colWidths=[0.1] * 3,
                      rowLabels=row_labels,
                      colLabels=col_labels,
                      loc='center')
the_table.auto_set_font_size(False)
the_table.set_fontsize(24)
the_table.scale(4, 4)

# Removing ticks and spines enables you to get the figure only with table
plt.tick_params(axis='x', which='both', bottom=False, top=False, labelbottom=F
plt.tick_params(axis='y', which='both', right=False, left=False, labelleft=Fal
for pos in ['right','top','bottom','left']:
    plt.gca().spines[pos].set_visible(False)
```

	Model	Hyper-Para	AUC
BOW	Brute	50	0.6690
TfIDF	Brute	40	0.5727