Assignment 8: 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

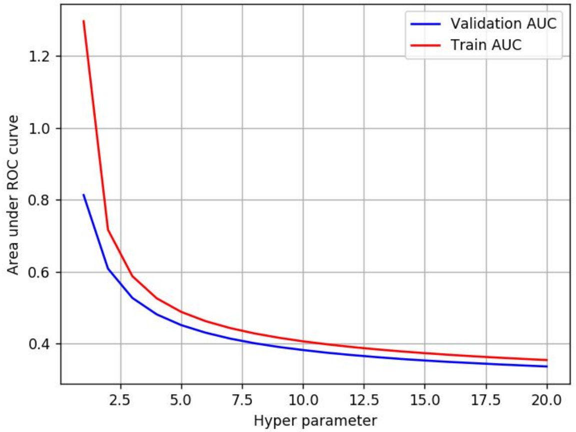
      essay
        while encoding essay, try to experiment with the max_features and n_grams parameter of vectorizers and see if it increases AUC score.

      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

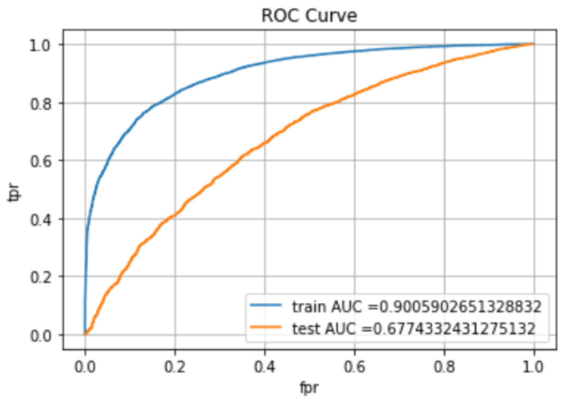
- For hyper parameter tuning using k-fold cross validation(use GridsearchCV or RandomsearchCV)/simple cross validation data (write for loop to iterate over hyper parameter values)
- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown 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

1. Naive Bayes

1.1 Loading Data

In [1]:
```
import numpy as np
import pandas as pd
from tqdm import tqdm
```

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

In [3]:
```
data.head(3)
```

Out[3]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categories
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

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

In [4]:
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

In [5]:
```
Y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
X.head(1)
```

Out[5]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories
0	ca	mrs	grades_prek_2	53	math_science	appliedsciences health_lifescience

In [6]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state = 0)
X_train, X_cv, Y_train, Y_cv = train_test_split(X_train, Y_train, test_size=0.3)
```

In [7]:
```
print(len(X_train))
print(len(X_cv))
print(len(X_test))
```

```
53531
22942
32775
```

In [8]:
```
data.columns
```

Out[8]:
```
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')
```

1.3 Make Data Model Ready: encoding essay

In [9]:
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

Using Bag Of Words

In [78]:
```
## NOTE: change ngram and max features and check if auc score increases
from sklearn.feature_extraction.text import CountVectorizer

vectorizer1 = CountVectorizer(min_df=10, ngram_range = (1,4))
vectorizer1.fit(X_train['essay'])
train_essay_bow = vectorizer1.transform(X_train['essay'])
cv_essay_bow = vectorizer1.transform(X_cv['essay'])
test_essay_bow = vectorizer1.transform(X_test['essay'])
print(train_essay_bow.shape)
print(cv_essay_bow.shape)
print(test_essay_bow.shape)
print(Y_train.shape, Y_cv.shape, Y_test.shape)
```

```
(53531, 183089)
(22942, 183089)
(32775, 183089)
(53531,) (22942,) (32775,)
```

Using TDIDF

In [79]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer8 = TfidfVectorizer(min_df=10, ngram_range=(1,4))
vectorizer8.fit(X_train['essay'].values)
train_essay_tfidf = vectorizer8.transform(X_train['essay'].values)
cv_essay_tfidf = vectorizer8.transform(X_cv['essay'].values)
test_essay_tfidf = vectorizer8.transform(X_test['essay'].values)
print(train_essay_tfidf.shape)
print(cv_essay_tfidf.shape)
print(test_essay_tfidf.shape)
print(Y_train.shape, Y_cv.shape, Y_test.shape)
```

```
(53531, 183089)
(22942, 183089)
(32775, 183089)
(53531,) (22942,) (32775,)
```

1.4 Make Data Model Ready: encoding numerical, categorical features

In [13]:
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
```

```
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
# categorical features
# - teacher_prefix
# - project_grade_category
# - school_state
# - clean_categories
# - clean_subcategories
# numerical features
# - price
# - teacher_number_of_previously_posted_projects
```

Categorical features

In [80]:

```
vectorizer2 = CountVectorizer()
vectorizer2.fit(X_train['school_state'].values)
train_state = vectorizer2.transform(X_train['school_state'].values)
cv_state = vectorizer2.transform(X_cv['school_state'].values)
test_state = vectorizer2.transform(X_test['school_state'].values)
print(train_state.shape)
print(cv_state.shape)
print(test_state.shape)
print(vectorizer2.get_feature_names())
```

```
(53531, 51)
(22942, 51)
(32775, 51)
['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']
```

In [81]:

```
vectorizer3 = CountVectorizer()
vectorizer3.fit(X_train['teacher_prefix'].values)
train_teacher_prefix = vectorizer3.transform(X_train['teacher_prefix'].values)
cv_teacher_prefix = vectorizer3.transform(X_cv['teacher_prefix'].values)
test_teacher_prefix = vectorizer3.transform(X_test['teacher_prefix'].values)
print(train_teacher_prefix.shape)
print(cv_teacher_prefix.shape)
```

```
print(test_teacher_prefix.shape)
print(vectorizer3.get_feature_names())
```

```
(53531, 5)
(22942, 5)
(32775, 5)
['dr', 'mr', 'mrs', 'ms', 'teacher']
```

In [82]:

```
vectorizer4 = CountVectorizer()
vectorizer4.fit(X_train['clean_categories'].values)
train_cat = vectorizer4.transform(X_train['clean_categories'].values)
cv_cat = vectorizer4.transform(X_cv['clean_categories'].values)
test_cat = vectorizer4.transform(X_test['clean_categories'].values)
print(train_cat.shape)
print(cv_cat.shape)
print(test_cat.shape)
print(vectorizer4.get_feature_names())
```

```
(53531, 9)
(22942, 9)
(32775, 9)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'music_arts',
'specialneeds', 'warmth']
```

In [83]:

```
vectorizer5 = CountVectorizer()
vectorizer5.fit(X_train['clean_subcategories'].values)
train_subcat = vectorizer5.transform(X_train['clean_subcategories'].values)
cv_subcat = vectorizer5.transform(X_cv['clean_subcategories'].values)
test_subcat = vectorizer5.transform(X_test['clean_subcategories'].values)
print(train_subcat.shape)
print(cv_subcat.shape)
print(test_subcat.shape)
print(vectorizer5.get_feature_names())
```

```
(53531, 30)
(22942, 30)
(32775, 30)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep', 'communityservice',
'earlydevelopment', 'economics', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy', 'foreignlanguage
s', 'gym_fitness', 'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'math
ematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socialsciences', 'specialneed
s', 'teamsports', 'visualarts', 'warmth']
```

In [84]:

```
vectorizer6 = CountVectorizer()
vectorizer6.fit(X_train['project_grade_category'].values)
train_grade = vectorizer6.transform(X_train['project_grade_category'].values)
```

```
cv_grade = vectorizer6.transform(X_cv['project_grade_category'].values)
test_grade = vectorizer6.transform(X_test['project_grade_category'].values)
print(train_grade.shape)
print(cv_grade.shape)
print(test_grade.shape)
print(vectorizer6.get_feature_names())
```

```
(53531, 4)
(22942, 4)
(32775, 4)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
```

Numerical Features

In [85]:

```
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import StandardScaler

previous_project_scalar = StandardScaler()
previous_project_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)) # finding the
print(f"Mean : {previous_project_scalar.mean_[0]}, Standard deviation : {np.sqrt(previous_project_scalar.var_[0])}")

train_prPos_norm = previous_project_scalar.transform(X_train['teacher_number_of_previously_posted_projects'].values.resh
cv_prPos_norm = previous_project_scalar.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-
test_prPos_norm = previous_project_scalar.transform(X_test['teacher_number_of_previously_posted_projects'].values.resha

print("After vectorizations")
print(train_prPos_norm.shape, Y_train.shape)
print(cv_prPos_norm.shape, Y_cv.shape)
print(test_prPos_norm.shape, Y_test.shape)
```

```
Mean : 11.028002465860903, Standard deviation : 27.542241880184424
After vectorizations
(53531, 1) (53531,)
(22942, 1) (22942,)
(32775, 1) (32775,)
```

In [86]:

```
from sklearn.preprocessing import StandardScaler

price_scalar = StandardScaler()
price_scalar.fit(X_train['price'].values.reshape(-1,1)) # finding the mean and standard deviation of this data
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

train_scaler_price = price_scalar.transform(X_train['price'].values.reshape(-1, 1))
cv_scaler_price = price_scalar.transform(X_cv['price'].values.reshape(-1, 1))
test_scaler_price = price_scalar.transform(X_test['price'].values.reshape(-1, 1))
```

```
print("After vectorizations")
print(train_scaler_price.shape, Y_train.shape)
print(cv_scaler_price.shape, Y_train.shape)
print(test_scaler_price.shape, Y_test.shape)
```

```
Mean : 297.92860548093626, Standard deviation : 365.4169653955776
After vectorizations
(53531, 1) (53531,)
(22942, 1) (53531,)
(32775, 1) (32775,)
```

Eliminating negative values

In [87]:
```
train_scaler_price = np.where(train_scaler_price<0, 0, train_scaler_price)
cv_scaler_price = np.where(cv_scaler_price<0, 0, cv_scaler_price)
test_scaler_price = np.where(test_scaler_price<0, 0, test_scaler_price)

train_prPos_norm = np.where(train_prPos_norm<0, 0, train_prPos_norm)
cv_prPos_norm = np.where(cv_prPos_norm<0, 0, cv_prPos_norm)
test_prPos_norm = np.where(test_prPos_norm<0, 0, test_prPos_norm)

print(train_scaler_price)
```

```
[[0.]
 [0.]
 [0.]
 ...
 [0.]
 [0.]
 [0.]]
```

1.5 Applying Naive Bayes on BOW

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 instructions

In [88]:
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
```

```
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

Stacking text, categorical and numerical features

In [112…

```
from scipy.sparse import hstack
X_tr = []
X_cv_stack = []
X_te = []
X_tr = hstack((train_essay_bow, train_state, train_teacher_prefix, train_grade, train_scaler_price, train_cat, train_su
X_cv_stack = hstack((cv_essay_bow, cv_state, cv_teacher_prefix, cv_grade, cv_scaler_price, cv_cat, cv_subcat)).tocsr()
X_te = hstack((test_essay_bow, test_state, test_teacher_prefix, test_grade, test_scaler_price, test_cat, test_subcat)).

X_tr = X_tr.tocsr()
X_cv_stack = X_cv_stack.tocsr()
X_te = X_te.tocsr()
print("Final Data matrix")
print(X_tr.shape, Y_train.shape)
print(X_cv_stack.shape, Y_cv.shape)
print(X_te.shape, Y_test.shape)
```

```
Final Data matrix
(53531, 183189) (53531,)
(22942, 183189) (22942,)
(32775, 183189) (32775,)
```

In [90]:

```
import math

## range of hyperparameter values
alpha = [0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,100]
log_alpha = list(map(lambda x : math.log10(x), alpha))
print(log_alpha)
```

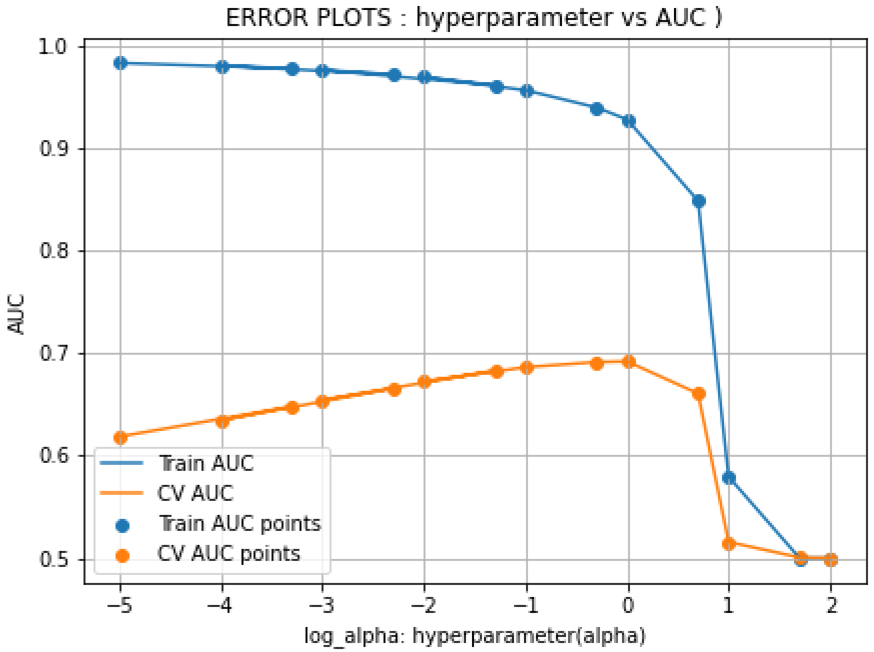
```
[-5.0, -3.3010299956639813, -4.0, -2.3010299956639813, -3.0, -1.3010299956639813, -2.0, -1.0, -0.3010299956639812, 0.0,
0.6989700043360189, 1.0, 1.6989700043360187, 2.0]
```

In [91]:

```
from sklearn.naive_bayes import MultinomialNB
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []

for i in tqdm(alpha):
```

```
    neigh = MultinomialNB(alpha = i, class_prior = [0.5,0.5])
    neigh.fit(X_tr, Y_train)

    Y_train_pred = neigh.predict_proba( X_tr)[:, 1]
    Y_cv_pred = neigh.predict_proba(X_cv_stack)[:, 1]

    train_auc.append(roc_auc_score(Y_train,Y_train_pred))
    cv_auc.append(roc_auc_score(Y_cv, Y_cv_pred))


plt.figure(figsize=(7,5))
plt.plot(log_alpha, train_auc, label='Train AUC')
plt.plot(log_alpha, cv_auc, label='CV AUC')

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

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

```
100%|██████████| 14/14 [00:01<00:00,  7.49it/s]
```



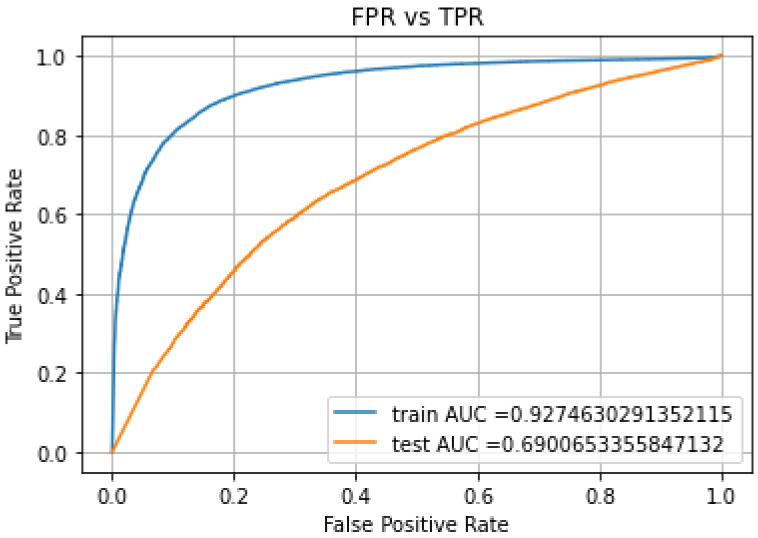
- low alpha values such as 0.00001 works very well on train data, whereas these values are not very efficient on cross validation data.
- Model works very well for both train and CV data with alpha values closer to 1.
- As alpha increases more than 1, the model seems not to be effective on both data.

Train model based on best hyper parameter value(alpha)

In [92]:
```
best_alpha = 1

from sklearn.metrics import roc_curve, auc


neigh = MultinomialNB(alpha = best_alpha, class_prior = [0.5,0.5])
neigh.fit(X_tr, Y_train)

y_train_pred = neigh.predict_proba(X_tr)[:, 1]
y_test_pred = neigh.predict_proba(X_te)[:, 1]

train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_test_pred)
```

```
auc1 = str(auc(train_fpr, train_tpr))
print(auc1)

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("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("FPR vs TPR")
plt.grid()
plt.show()
```

```
0.9274630291352115
```



train AUC = 0.93 test AUC = 0.69

In [93]:
```
test_auc1 = auc(test_fpr, test_tpr)
print(test_auc1)
```

```
0.6900653355847132
```

In [94]:
```
# 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
```

Confusion matrix on Train and Test set

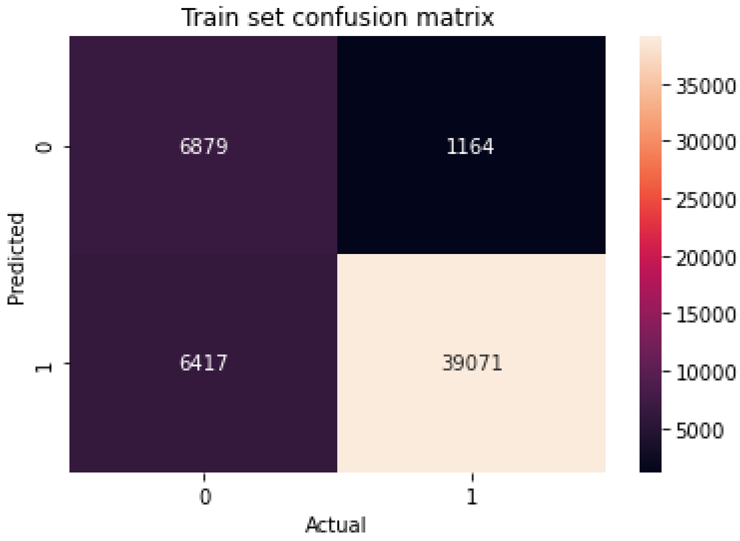
In [95]:

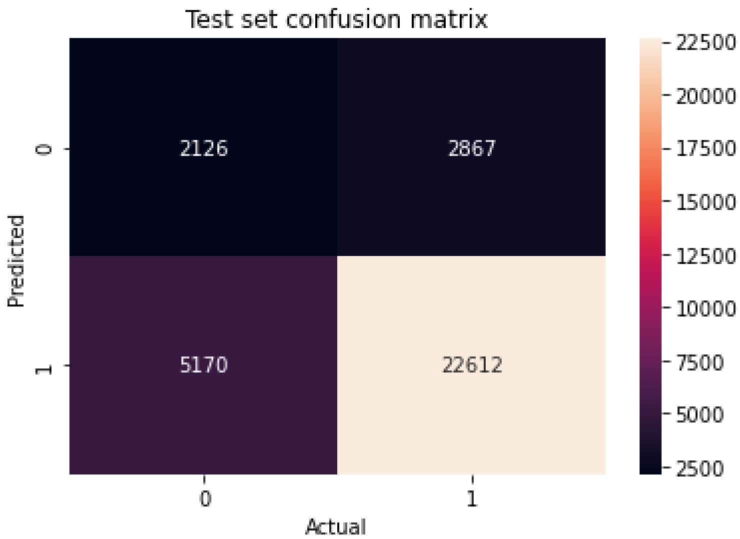
```
import numpy as np
print("="*100)
from sklearn.metrics import confusion_matrix
best_t1 = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(Y_train, predict_with_best_t(y_train_pred, best_t1)))
print("Test confusion matrix")
print(confusion_matrix(Y_test, predict_with_best_t(y_test_pred, best_t1)))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.734623683250366 for threshold 0.278
Train confusion matrix
[[ 6879  1164]
 [ 6417 39071]]
Test confusion matrix
[[ 2126  2867]
 [ 5170 22612]]
```

In [96]:

```
import seaborn as sns
# Heatmap for train set confusion matrix(Select K best)
heatmap_train = sns.heatmap(confusion_matrix(Y_train, predict_with_best_t(y_train_pred,best_t1)), annot=True, fmt="d")
plt.title("Train set confusion matrix")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()

# Heatmap for test set confusion matrix(Select K best)
heatmap_train = sns.heatmap(confusion_matrix(Y_test, predict_with_best_t(y_test_pred, best_t1)), annot=True, fmt="d")
plt.title("Test set confusion matrix")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```





In [97]:

```
# To find the top features of positive/negative class https://github.com/shashimanyam/NaiveBayes/blob/master/NAVIEBAYES
bow_features_probs = []
bow_features_neg = []

for a in range(len(neigh.feature_log_prob_[0,:])):
    bow_features_probs.append(neigh.feature_log_prob_[0,a] )
print(len(bow_features_probs))
```

```
print((neigh.feature_log_prob_)[0])
print((neigh.feature_log_prob_)[1])
```

```
183189
[-11.49813577 -12.88443013 -10.74436397 ...  -9.33385535  -8.35004013
 -10.59865216]
[-11.72333448 -13.40973343 -10.43165609 ...  -9.69129517  -8.5782248
  -9.97252561]
```

In [98]:

```
bow_features_names = []
for a in vectorizer1.get_feature_names(): # clean_categories
    bow_features_names.append(a)
for a in vectorizer2.get_feature_names(): # clean_sub_categories
    bow_features_names.append(a)
for a in vectorizer3.get_feature_names(): # school state
    bow_features_names.append(a)
for a in vectorizer4.get_feature_names(): # teacher_prefix
    bow_features_names.append(a)
for a in vectorizer5.get_feature_names(): # Grades
    bow_features_names.append(a)
for a in vectorizer6.get_feature_names(): # bow_essay
    bow_features_names.append(a)

bow_features_names.append("price")
print(len(bow_features_names))
```

```
183189
```

20 positive and negative features with high and low coeff scores from BOW:

In [99]:

```
top_ind_pos=np.argsort((neigh.feature_log_prob_)[1])[::1][0:20]
last_ind_pos=np.argsort((neigh.feature_log_prob_)[1])[::-1][0:20]
top_pos=np.take(bow_features_names,top_ind_pos)
last_pos=np.take(bow_features_names,last_ind_pos)

top_ind_neg=np.argsort((neigh.feature_log_prob_)[0])[::1][0:20]
last_ind_neg=np.argsort((neigh.feature_log_prob_)[0])[::-1][0:20]
top_neg=np.take(bow_features_names,top_ind_neg)
last_neg=np.take(bow_features_names,last_ind_neg)

print("Top positive features with high coeff:")
print(top_pos)
print("")
print("Top positive features with low coeff:")
print(last_pos)
```

```
print("")
print("Top negative features with high coeff:")
print(top_neg)
print("")
print("Top negative features with low coeff:")
print(last_neg)
```

```
Top positive features with high coeff:
['the demographic' 'life without constant'
 'life without constant connectivity' 'these supplies allow us'
 'want environment' 'without constant connectivity'
 'know life without constant' 'believer keeping focus' 'constantly solve'
 'believer keeping' 'fun engaging materials' 'begs' 'come low ses'
 'raise siblings' 'come work hard' 'minutes pe' 'beyond pencil paper'
 'products requested' 'items need help' 'letters words sentences']

Top positive features with low coeff:
['students' 'school' 'my' 'learning' 'classroom' 'the' 'they' 'not'
 'my students' 'learn' 'help' 'many' 'nannan' 'we' 'work' 'need' 'reading'
 'use' 'love' 'day']

Top negative features with high coeff:
['love wobble stools' 'carpet requesting' 'carpet reading' 'novels books'
 'every day love learning' 'with laptops students' 'the apple tv'
 'work sharing' 'the area school located' 'carpet meeting' 'carpet make'
 'looking best' 'carpet large' 'carpet it' 'carpet help students'
 'carpet give students' 'carpet give' 'carpet floor' 'carpet essential'
 'carpet daily']

Top negative features with low coeff:
['students' 'school' 'learning' 'my' 'classroom' 'not' 'learn' 'they'
 'help' 'the' 'my students' 'nannan' 'many' 'we' 'need' 'work' 'come'
 'love' 'skills' 'able']
```

1.6 Applying Naive Bayes on TFIDF

In [117…

```
from scipy.sparse import hstack
X_tr = []
X_cv_stack = []
X_te = []

X_tr = hstack((train_essay_tfidf, train_state, train_teacher_prefix, train_grade, train_scaler_price, train_cat, train_s
X_cv_stack = hstack((cv_essay_tfidf, cv_state, cv_teacher_prefix, cv_grade, cv_scaler_price, cv_cat, cv_subcat)).tocsr(
X_te = hstack((test_essay_tfidf, test_state, test_teacher_prefix, test_grade, test_scaler_price, test_cat, test_subcat)
```

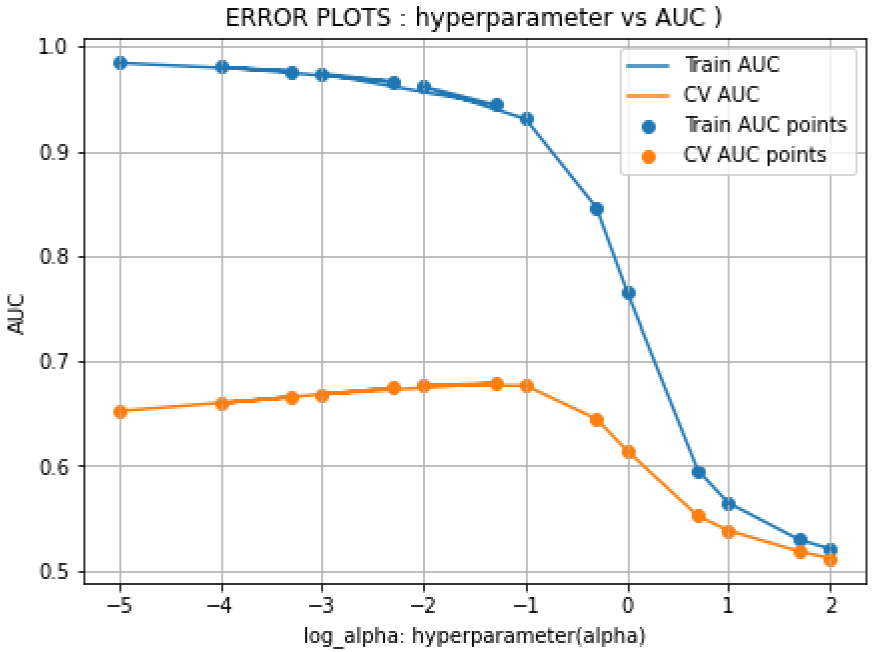
```
print(X_tr.shape , Y_train.shape)
print(X_cv_stack.shape , Y_cv.shape)
print(X_te.shape , Y_test.shape)
```

```
(53531, 183189) (53531,)
(22942, 183189) (22942,)
(32775, 183189) (32775,)
```

In [118…

```
from sklearn.naive_bayes import MultinomialNB
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []

for i in tqdm(alpha):
    neigh = MultinomialNB(alpha = i, class_prior = [0.5,0.5])
    neigh.fit(X_tr, Y_train)

    y_train_pred = neigh.predict_proba(X_tr)[:, 1]
    y_cv_pred = neigh.predict_proba(X_cv_stack)[:, 1]

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(Y_train,y_train_pred))
    cv_auc.append(roc_auc_score(Y_cv, y_cv_pred))

plt.figure(figsize=(7,5))
plt.plot(log_alpha, train_auc, label='Train AUC')
plt.plot(log_alpha, cv_auc, label='CV AUC')

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

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

```
100%|██████████| 14/14 [00:01<00:00,  7.13it/s]
```



- low alpha values such as 0.00001 works very well on train data, whereas these values are not very efficient on cross validation data.
- Model works very well for both train and CV data with alpha values closer to 0.1
- As alpha increases more than 0.1, the model seems not to be effective on both data.

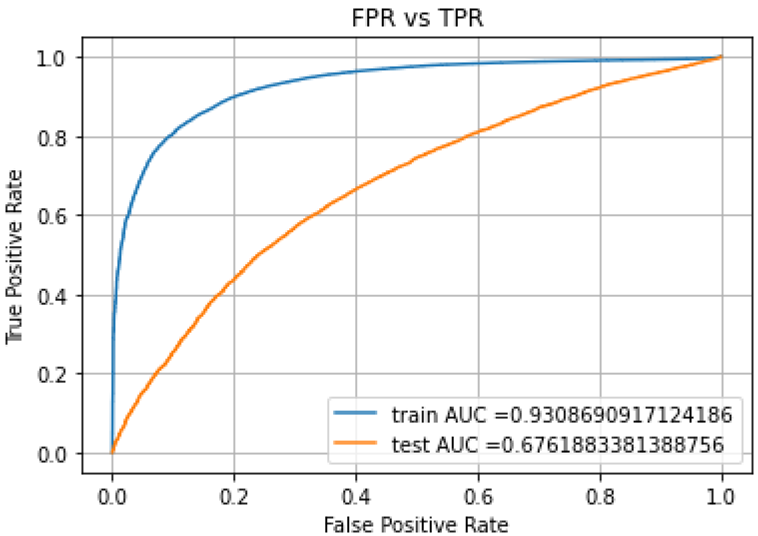
Train model based on best hyper parameter value(alpha)

In [119...
```
best_alpha2 = 0.1
```

In [120...
```
from sklearn.metrics import roc_curve, auc


neigh = MultinomialNB(alpha = best_alpha2, class_prior = [0.5,0.5])
neigh.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 = neigh.predict_proba(X_tr)[:, 1]
y_test_pred = neigh.predict_proba(X_te)[:, 1]

train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, y_train_pred)
```

```
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_test_pred)

auc2 = str(auc(train_fpr, train_tpr))
print(auc2)

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("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("FPR vs TPR")
plt.grid()
plt.show()
```

```
0.9308690917124186
```



In [121…

```
test_auc2 = auc(test_fpr, test_tpr)
print(test_auc2)
```

```
0.6761883381388756
```

Confusion matrix on Train and Test Data

In [124…

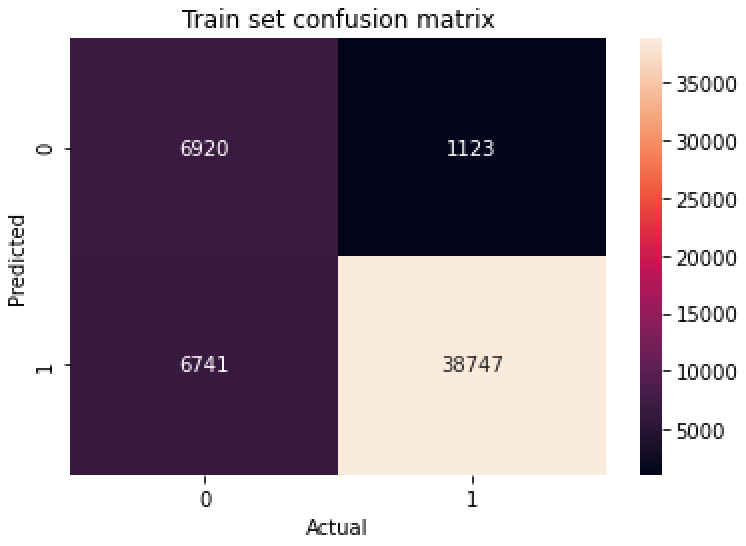
```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t2 = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
```

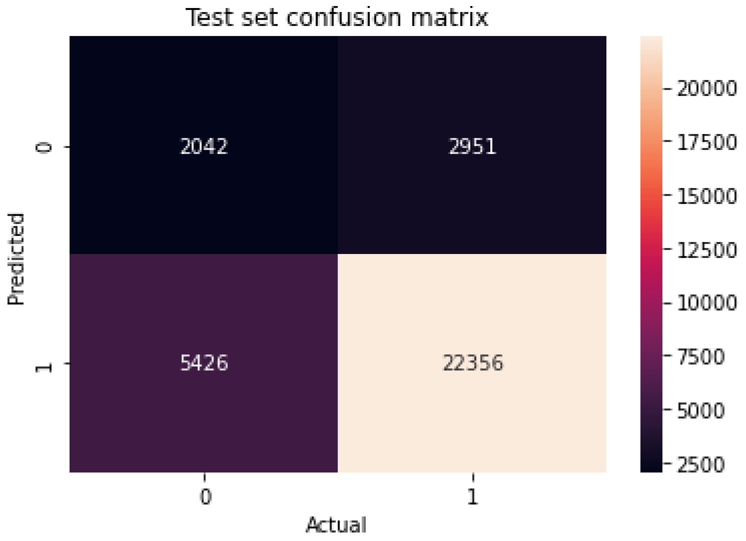
```
print(confusion_matrix(Y_train, predict_with_best_t(y_train_pred, best_t2)))
print("Test confusion matrix")
print(confusion_matrix(Y_test, predict_with_best_t(y_test_pred, best_t2)))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.7328739182364366 for threshold 0.493
Train confusion matrix
[[ 6920  1123]
 [ 6741 38747]]
Test confusion matrix
[[ 2042  2951]
 [ 5426 22356]]
```

In [125…

```
# Heatmap for train set confusion matrix(Select K best)
heatmap_train = sns.heatmap(confusion_matrix(Y_train, predict_with_best_t(y_train_pred,best_t2)), annot=True, fmt="d")
plt.title("Train set confusion matrix")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()

# Heatmap for test set confusion matrix(Select K best)
heatmap_train = sns.heatmap(confusion_matrix(Y_test, predict_with_best_t(y_test_pred, best_t2)), annot=True, fmt="d")
plt.title("Test set confusion matrix")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```





In [126…

```
tfidf_features_probs = []
for a in range(len(neigh.feature_log_prob_[0,:])):
    tfidf_features_probs.append(neigh.feature_log_prob_[0,a] )
print(len(tfidf_features_probs))

print((neigh.feature_log_prob_)[0])
print((neigh.feature_log_prob_)[1])
```

```
183189
[-11.59196456 -12.70756192 -10.97900552 ...  -6.75240268  -5.76588326
  -8.02825575]
[-11.74780679 -13.26743958 -10.66227718 ...  -7.03766115  -5.92385686
  -7.31924675]
```

In [127…

```
tfidf_features_names = []
for a in vectorizer8.get_feature_names(): # clean_categories
    tfidf_features_names.append(a)
for a in vectorizer2.get_feature_names(): # clean_sub_categories
    tfidf_features_names.append(a)
for a in vectorizer3.get_feature_names(): # school state
    tfidf_features_names.append(a)
for a in vectorizer4.get_feature_names(): # teacher_prefix
    tfidf_features_names.append(a)
for a in vectorizer5.get_feature_names(): # Grades
    tfidf_features_names.append(a)
for a in vectorizer6.get_feature_names(): # bow_essay
```

```
    tfidf_features_names.append(a)

tfidf_features_names.append("price")
print(len(tfidf_features_names))
```

```
183189
```

20 positive and negative features with high and low coeff scores from TFIDF:

In [128…

```
top_ind_pos=np.argsort((neigh.feature_log_prob_)[1])[::1][0:20]
last_ind_pos=np.argsort((neigh.feature_log_prob_)[1])[::-1][0:20]
top_pos=np.take(bow_features_names,top_ind_pos)
last_pos=np.take(bow_features_names,last_ind_pos)

top_ind_neg=np.argsort((neigh.feature_log_prob_)[0])[::1][0:20]
last_ind_neg=np.argsort((neigh.feature_log_prob_)[0])[::-1][0:20]
top_neg=np.take(bow_features_names,top_ind_neg)
last_neg=np.take(bow_features_names,last_ind_neg)

print("Top positive features with high coeff:")
print(top_pos)
print("")
print("Top positive features with low coeff:")
print(last_pos)
print("")
print("Top negative features with high coeff:")
print(top_neg)
print("")
print("Top negative features with low coeff:")
print(last_neg)
```

```
Top positive features with high coeff:
['without constant connectivity' 'know life without constant'
 'life without constant' 'life without constant connectivity'
 'know life without' 'not know life without' 'these supplies allow us'
 'constantly solve' 'constantly solve problems'
 'constantly solve problems make' 'include real life'
 'materials requesting give students' 'good attendance eager'
 'good attendance eager learn' 'show good attendance eager'
 'show good attendance' 'live show good attendance'
 'attendance eager learn my' 'students live show good' 'live show good']

Top positive features with low coeff:
['mrs' 'appliedsciences' 'history_civics' 'care_hunger' 'ms'
 'appliedlearning' 'other' 'literacy_language' 'performingarts'
 'parentinvolvement' 'care_hunger' 'ca' 'specialneeds' 'grades_6_8'
```

```
 'civics_government' 'math_science' 'health_sports' 'music' 'mr'
 'communityservice']

Top negative features with high coeff:
['living low socioeconomic' 'huge part learning' 'huge learning'
 'huge component' 'huge benefit students' 'huddle' 'hp printer'
 'hp chromebook' 'however teachers' 'however support' 'however provide'
 'however not let' 'however music department true'
 'however music department' 'however music' 'however many come'
 'however little' 'however excited' 'hours students' 'hours they']

Top negative features with low coeff:
['mrs' 'appliedsciences' 'history_civics' 'care_hunger'
 'literacy_language' 'ms' 'appliedlearning' 'performingarts' 'other'
 'parentinvolvement' 'care_hunger' 'civics_government' 'grades_6_8' 'ca'
 'specialneeds' 'math_science' 'communityservice' 'health_sports'
 'charactereducation' 'mr']
```

Summary

In [129…

```python
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyper Parameter", "Train AUC", "Test AUC"]

x.add_row(["BOW", "Naive Bayes", best_alpha, auc1, test_auc1])
x.add_row([" ", " ", " ", " ", " "])
x.add_row(["TFIDF", "Naive Bayes", best_alpha2, auc2, test_auc2])

print(x)
```

```
+------------+-------------+-----------------+--------------------+--------------------+
| Vectorizer |    Model    | Hyper Parameter |     Train AUC      |      Test AUC      |
+------------+-------------+-----------------+--------------------+--------------------+
|    BOW     | Naive Bayes |        1        | 0.9274630291352115 | 0.6900653355847132 |
|            |             |                 |                    |                    |
|   TFIDF    | Naive Bayes |       0.1       | 0.9308690917124186 | 0.6761883381388756 |
+------------+-------------+-----------------+--------------------+--------------------+
```