Assignment_8

March 1, 2022

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

import pandas as pd
import numpy as np
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc

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

import pickle
from tqdm import tqdm
import os

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


data  = pd.read_csv('preprocessed_data.csv')     # I have taken all the rows
print(data.shape)
data.head(5)
```

```
(109248, 9)
```

[1]:

	school_state	teacher_prefix	project_grade_category	\
0	ca	mrs	grades_prek_2	
1	ut	ms	grades_3_5	
2	ca	mrs	grades_prek_2	

```
3           ga            mrs          grades_prek_2
4           wa            mrs             grades_3_5

   teacher_number_of_previously_posted_projects  project_is_approved  \
0                                            53                    1
1                                             4                    1
2                                            10                    1
3                                             2                    1
4                                             2                    1

    clean_categories                clean_subcategories  \
0       math_science  appliedsciences health_lifescience
1       specialneeds                        specialneeds
2  literacy_language                            literacy
3    appliedlearning                   earlydevelopment
4  literacy_language                            literacy

                                               essay   price
0  i fortunate enough use fairy tale stem kits cl…  725.05
1  imagine 8 9 years old you third grade classroo…  213.03
2  having class 24 students comes diverse learner…  329.00
3  i recently read article giving students choice…  481.04
4  my students crave challenge eat obstacles brea…   17.74
```

0.1 Set 1: categorical, numerical features + preprocessed_eassay (BOW)

0.1.1 Splitting the data and encoding Essay

[2]:
```
y = data['project_is_approved'].values
x = data.drop(['project_is_approved'], axis=1)
x.head(1)

from sklearn.model_selection import train_test_split   #splitting the data into␣
 ↪train & test (No CV)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33,␣
 ↪stratify=y)

print(x_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)

vectorizer_essay = CountVectorizer(min_df=10,ngram_range=(1,4),␣
 ↪max_features=5000)
vectorizer_essay.fit(x_train['essay'].values)

x_train_essay_bow = vectorizer_essay.transform(x_train['essay'].values)
x_test_essay_bow = vectorizer_essay.transform(x_test['essay'].values)
```

```
print("After vectorizations")
print(x_train_essay_bow.shape, y_train.shape)
print(x_test_essay_bow.shape, y_test.shape)
```

```
(73196, 8) (73196,)
(36052, 8) (36052,)
After vectorizations
(73196, 5000) (73196,)
(36052, 5000) (36052,)
```

0.1.2 encoding categorical features: teacher_prefix

[3]:
```
vectorizer_teacher = CountVectorizer()
vectorizer_teacher.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 = vectorizer_teacher.transform(x_train['teacher_prefix'].values)
x_test_teacher = vectorizer_teacher.transform(x_test['teacher_prefix'].values)

print("After vectorizations")
print(x_train_teacher.shape, y_train.shape)
print(x_test_teacher.shape, y_test.shape)
print(vectorizer_teacher.get_feature_names())
```

```
After vectorizations
(73196, 5) (73196,)
(36052, 5) (36052,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
```

0.1.3 encoding categorical features: project_grade_category

[4]:
```
vectorizer_grade = CountVectorizer()
vectorizer_grade.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 = vectorizer_grade.transform(x_train['project_grade_category'].
 ↪values)
x_test_grade = vectorizer_grade.transform(x_test['project_grade_category'].
 ↪values)

print("After vectorizations")
print(x_train_grade.shape, y_train.shape)
print(x_test_grade.shape, y_test.shape)
print(vectorizer_grade.get_feature_names())
```

```
After vectorizations
(73196, 4) (73196,)
(36052, 4) (36052,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
```

0.1.4 encoding categorical features: school_state

[5]:
```
vectorizer_state = CountVectorizer()
vectorizer_state.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 = vectorizer_state.transform(x_train['school_state'].values)
x_test_state = vectorizer_state.transform(x_test['school_state'].values)

print("After vectorizations")
print(x_train_state.shape, y_train.shape)
print(x_test_state.shape, y_test.shape)
print(vectorizer_state.get_feature_names())
```

```
After vectorizations
(73196, 51) (73196,)
(36052, 51) (36052,)
['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']
```

0.1.5 encoding categorical features: clean_categories

[6]:
```
vectorizer_categ = CountVectorizer()
vectorizer_categ.fit(x_train['clean_categories'].values) # fit has to happen␣
 ↪only on train data

# we use the fitted CountVectorizer to convert the text to vector
x_train_categ = vectorizer_categ.transform(x_train['clean_categories'].values)
x_test_categ = vectorizer_categ.transform(x_test['clean_categories'].values)

print("After vectorizations")
print(x_train_categ.shape, y_train.shape)
print(x_test_categ.shape, y_test.shape)
print(vectorizer_categ.get_feature_names())
```

```
After vectorizations
(73196, 9) (73196,)
(36052, 9) (36052,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics',
'literacy_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
```

0.1.6 encoding categorical features: clean_subcategories

[7]:
```
vectorizer_sub_categ = CountVectorizer()
vectorizer_sub_categ.fit(x_train['clean_subcategories'].values) # fit has to␣
↪happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
x_train_sub_categ = vectorizer_sub_categ.
↪transform(x_train['clean_subcategories'].values)
x_test_sub_categ = vectorizer_sub_categ.transform(x_test['clean_subcategories'].
↪values)

print("After vectorizations")
print(x_train_sub_categ.shape, y_train.shape)
print(x_test_sub_categ.shape, y_test.shape)
print(vectorizer_sub_categ.get_feature_names())
```

```
After vectorizations
(73196, 30) (73196,)
(36052, 30) (36052,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics',
'environmentalscience', 'esl', 'extracurricular', 'financialliteracy',
'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_wellness',
'history_geography', 'literacy', 'literature_writing', 'mathematics', 'music',
'nutritioneducation', 'other', 'parentinvolvement', 'performingarts',
'socialsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
```

0.1.7 encoding numerical features: Price

[8]:
```
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].
# we can give columns as a list to resolve the problem
print(x_train[['price']].shape) # It will be 2D
normalizer.fit(x_train[['price']])

x_train_price = normalizer.transform(x_train[['price']].values)
x_test_price = normalizer.transform(x_test[['price']].values)

print("After vectorizations")
print(x_train_price.shape, y_train.shape)
print(x_test_price.shape, y_test.shape)
```

```
(73196, 1)
After vectorizations
```

```
(73196, 1) (73196,)
(36052, 1) (36052,)
```

0.1.8 encoding numerical features: teacher_number_of_previously_posted_projects

[9]:
```
normalizer = Normalizer()
print(x_train[['teacher_number_of_previously_posted_projects']].shape)
normalizer.fit(x_train[['teacher_number_of_previously_posted_projects']])

x_train_no_of_prev_proj = normalizer.
 ↪transform(x_train[['teacher_number_of_previously_posted_projects']].values)
x_test_no_of_prev_proj = normalizer.
 ↪transform(x_test[['teacher_number_of_previously_posted_projects']].values)

print("After vectorizations")
print(x_train_no_of_prev_proj.shape, y_train.shape)
print(x_test_no_of_prev_proj.shape, y_test.shape)
```

```
(73196, 1)
After vectorizations
(73196, 1) (73196,)
(36052, 1) (36052,)
```

0.1.9 Concatinating all the features (BOW)

[10]:
```
from scipy.sparse import hstack
x_tr_bow = hstack((x_train_essay_bow, x_train_teacher, x_train_grade,␣
 ↪x_train_state, x_train_categ, x_train_sub_categ, x_train_price,␣
 ↪x_train_no_of_prev_proj)).tocsr()
x_te_bow = hstack((x_test_essay_bow, x_test_teacher, x_test_grade,␣
 ↪x_test_state, x_test_categ, x_test_sub_categ, x_test_price,␣
 ↪x_test_no_of_prev_proj)).tocsr()
print("Final Data matrix")
print(x_tr_bow.shape, y_train.shape)
print(x_te_bow.shape, y_test.shape)
```

```
Final Data matrix
(73196, 5101) (73196,)
(36052, 5101) (36052,)
```

0.2 Set 2: categorical, numerical features + preprocessed_eassay (TF-IDF)

0.2.1 encoding essay

[11]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(min_df=10, max_features=5000)
vectorizer.fit(x_train['essay'])
```

```
x_train_essay_tfidf = vectorizer.transform(x_train['essay'])
x_test_essay_tfidf  = vectorizer.transform(x_test['essay'])

print(x_train_essay_tfidf.shape, y_train.shape)
print(x_test_essay_tfidf.shape, y_test.shape)
```

```
(73196, 5000) (73196,)
(36052, 5000) (36052,)
```

0.2.2 Concatinating all the features (TF-IDF)

[12]:
```
x_tr_tfidf = hstack((x_train_essay_tfidf, x_train_teacher, x_train_grade,␣
 ↪x_train_state, x_train_categ, x_train_sub_categ, x_train_price,␣
 ↪x_train_no_of_prev_proj)).tocsr()
x_te_tfidf = hstack((x_test_essay_tfidf, x_test_teacher, x_test_grade,␣
 ↪x_test_state, x_test_categ, x_test_sub_categ, x_test_price,␣
 ↪x_test_no_of_prev_proj)).tocsr()
print("Final Data matrix")
print(x_tr_tfidf.shape, y_train.shape)
print(x_te_tfidf.shape, y_test.shape)
```

```
Final Data matrix
(73196, 5101) (73196,)
(36052, 5101) (36052,)
```

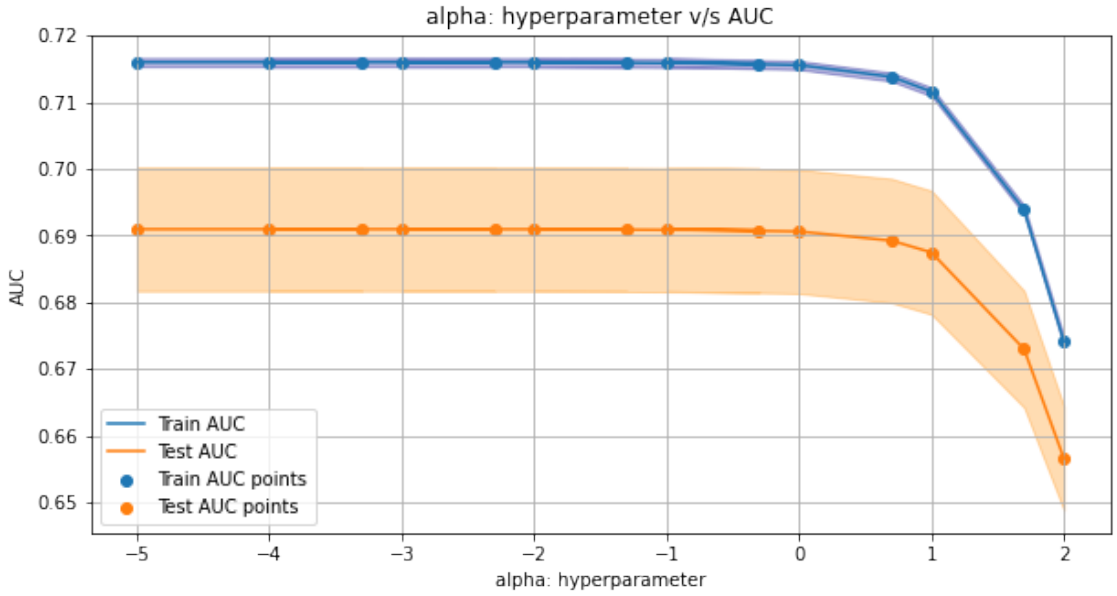
0.3 Appling Naive Bayes: BOW featurization

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

mnb_bow = MultinomialNB(class_prior=[0.5, 0.5])
parameters = {'alpha':[0.00001, 0.0005, 0.0001, 0.005, 0.001, 0.05, 0.01, 0.5,␣
 ↪0.1, 1, 5, 10, 50, 100]}
clf = GridSearchCV(mnb_bow, parameters, cv= 10,␣
 ↪scoring='roc_auc',verbose=1,return_train_score=True)
clf.fit(x_tr_bow,y_train)
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
test_auc = clf.cv_results_['mean_test_score']
test_auc_std= clf.cv_results_['std_test_score']
bestAlpha_1=clf.best_params_['alpha']
bestScore_1=clf.best_score_
print("BEST ALPHA: ",clf.best_params_['alpha']," BEST SCORE: ",clf.best_score_)
```

```
Fitting 10 folds for each of 14 candidates, totalling 140 fits
BEST ALPHA:  1e-05  BEST SCORE:  0.6909453019832686
```

[14]:
```
alphas = [0.00001, 0.0005, 0.0001, 0.005, 0.001, 0.05, 0.01, 0.5, 0.1, 1, 5,␣
 ↪10, 50, 100]
log_alphas = []

for a in tqdm(alphas):
    log_a = np.log10(a)

    log_alphas.append(log_a)

plt.figure(figsize = (10, 5))
plt.plot(log_alphas, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alphas,train_auc - train_auc_std,train_auc +␣
 ↪train_auc_std,alpha=0.3,color='darkblue')
plt.plot(log_alphas, test_auc, label='Test AUC')
plt.gca().fill_between(log_alphas,test_auc - test_auc_std,test_auc +␣
 ↪test_auc_std,alpha=0.3,color='darkorange')
plt.scatter(log_alphas, train_auc, label='Train AUC points')
plt.scatter(log_alphas, test_auc, label='Test AUC points')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("alpha: hyperparameter v/s AUC")
plt.grid()
plt.show()
```

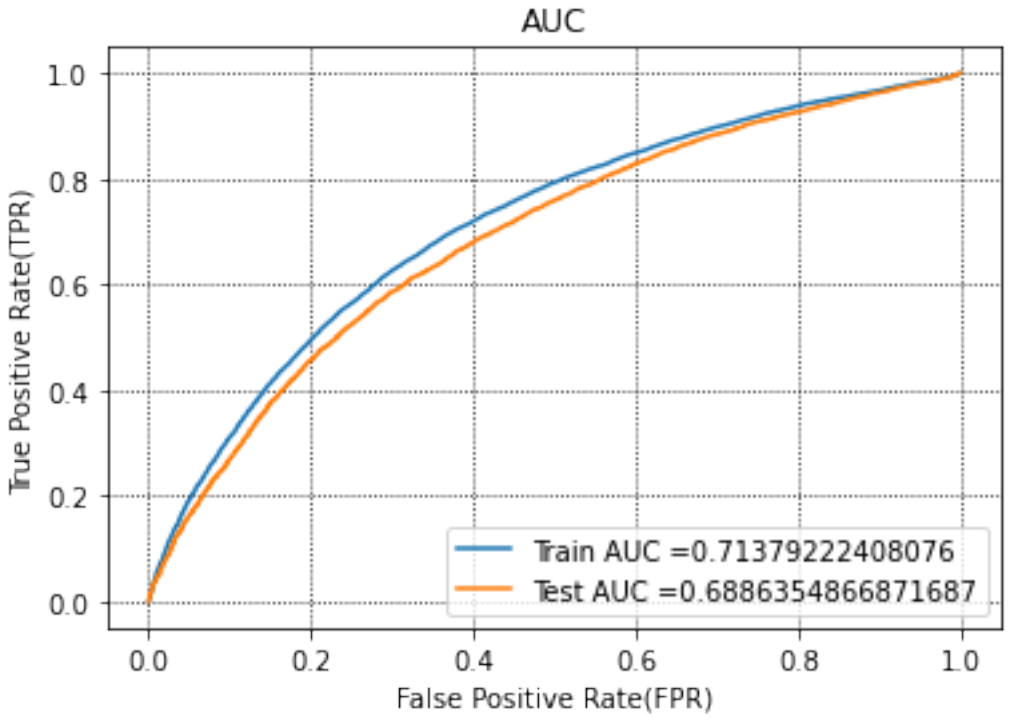
```
100%|
        | 14/14 [00:00<?, ?it/s]
```



0.3.1 TESTING WITH BEST HYPERPARAMETER VALUE ON SET 1

[15]:
```
#https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.
 ↪html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
mnb_bow_testModel = MultinomialNB(alpha = bestAlpha_1,class_prior=[0.5, 0.5])
mnb_bow_testModel.fit(x_tr_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability␣
 ↪estimates of the positive class
# not the predicted outputs
# y_train_pred = batch_predict(mnb_bow_testModel, x_train_onehot_bow)
y_train_pred=mnb_bow_testModel.predict_proba(x_tr_bow)[:,1]
# y_test_pred = batch_predict(mnb_bow_testModel, x_test_onehot_bow)
y_test_pred=mnb_bow_testModel.predict_proba(x_te_bow)[:,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)

ax = plt.subplot()

auc_set1_train=auc(train_fpr, train_tpr)
auc_set1_test=auc(test_fpr, test_tpr)


ax.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr,␣
 ↪train_tpr)))
ax.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
```

```
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("AUC")
plt.grid(b=True, which='major', color='k', linestyle=':')
ax.set_facecolor("white")
plt.show()
```



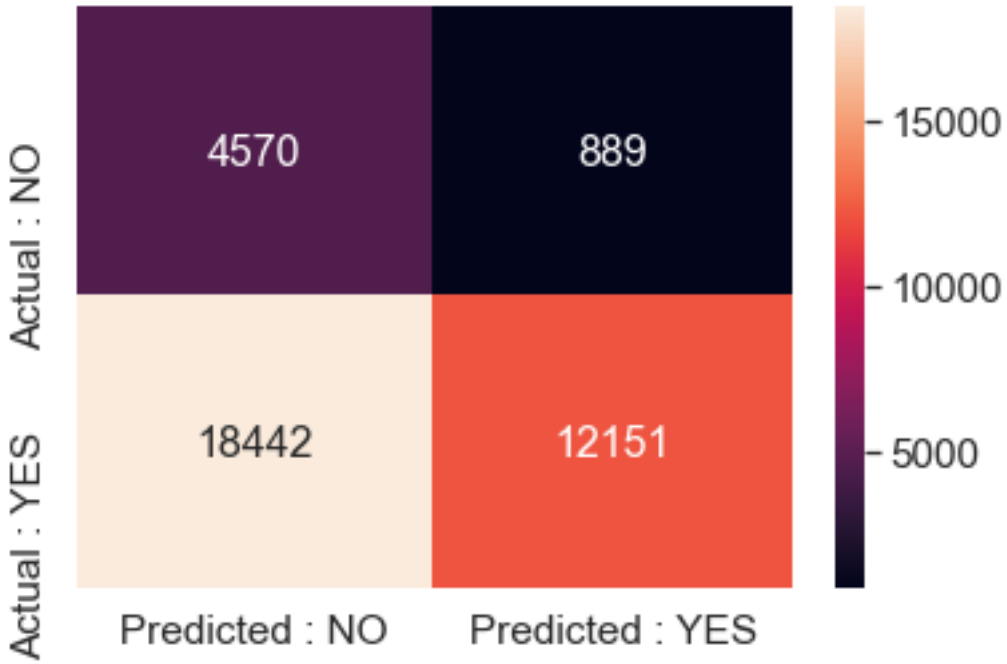
[16]:
```
def predict(proba, threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for␣
 ↪threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

0.3.2 Confusion matrix

[17]:
```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr,␣
 ↪test_tpr)))

conf_matr_df_test = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred,␣
 ↪tr_thresholds, test_fpr, test_tpr)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
xticklabels = ["Predicted : NO","Predicted : YES"]
yticklabels = ["Actual : NO","Actual : YES"]
sns.heatmap(conf_matr_df_test,xticklabels=xticklabels,yticklabels=yticklabels␣
 ↪,annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.4144264761491971 for threshold 0.989
[[ 4570   889]
 [18442 12151]]
the maximum value of tpr*(1-fpr) 0.4144264761491971 for threshold 0.989
```

[17]: <AxesSubplot:>



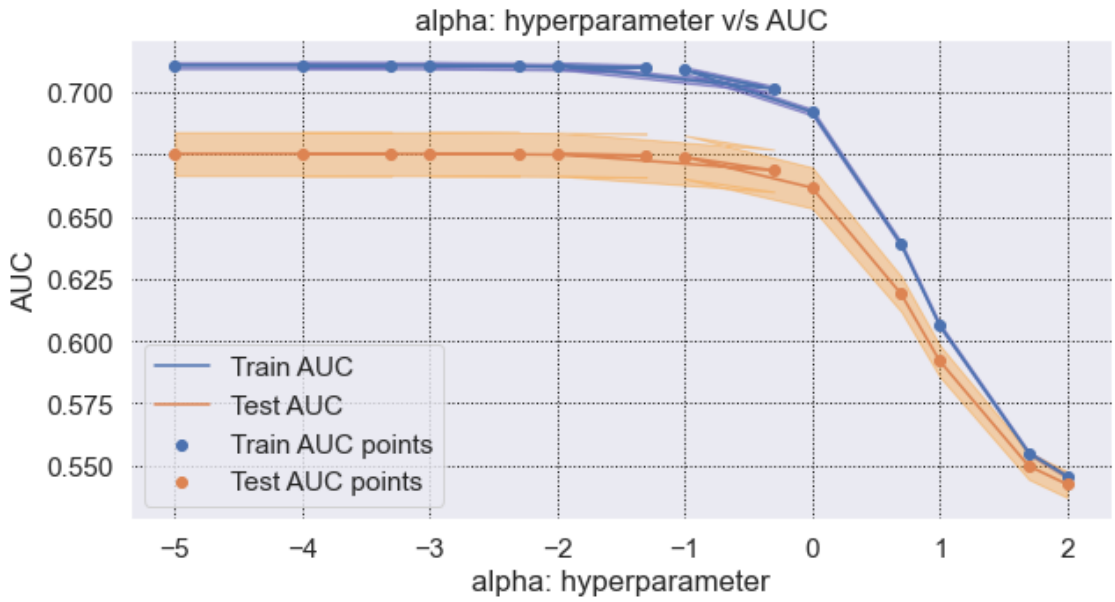
0.4 Appling Naive Bayes: TF_IDF featurization

[18]:
```
mnb_tfidf = MultinomialNB(class_prior=[0.5, 0.5])
parameters = {'alpha':[0.00001, 0.0005, 0.0001, 0.005, 0.001, 0.05, 0.01, 0.5,␣
 ↪0.1, 1, 5, 10, 50, 100]}
clf = GridSearchCV(mnb_tfidf, parameters, cv= 10,␣
 ↪scoring='roc_auc',verbose=1,return_train_score=True)
clf.fit(x_tr_tfidf,y_train)
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
test_auc = clf.cv_results_['mean_test_score']
test_auc_std= clf.cv_results_['std_test_score']
bestAlpha_2=clf.best_params_['alpha']
bestScore_2=clf.best_score_
print("BEST ALPHA: ",clf.best_params_['alpha']," BEST SCORE: ",clf.best_score_)
```

```
Fitting 10 folds for each of 14 candidates, totalling 140 fits
BEST ALPHA:  1e-05  BEST SCORE:  0.6753073664011305
```

[19]:
```
alphas = [0.00001, 0.0005, 0.0001, 0.005, 0.001, 0.05, 0.01, 0.5, 0.1, 1, 5,␣
 ↪10, 50, 100]
log_alphas = []

for a in tqdm(alphas):
    log_a = np.log10(a)

    log_alphas.append(log_a)

plt.figure(figsize = (10, 5))
plt.plot(log_alphas, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alphas,train_auc - train_auc_std,train_auc +␣
 ↪train_auc_std,alpha=0.3,color='darkblue')
plt.plot(log_alphas, test_auc, label='Test AUC')
plt.gca().fill_between(log_alphas,test_auc - test_auc_std,test_auc +␣
 ↪test_auc_std,alpha=0.3,color='darkorange')
plt.scatter(log_alphas, train_auc, label='Train AUC points')
plt.scatter(log_alphas, test_auc, label='Test AUC points')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("alpha: hyperparameter v/s AUC")
plt.grid(b=True, which='major', color='k', linestyle=':')
plt.show()
```

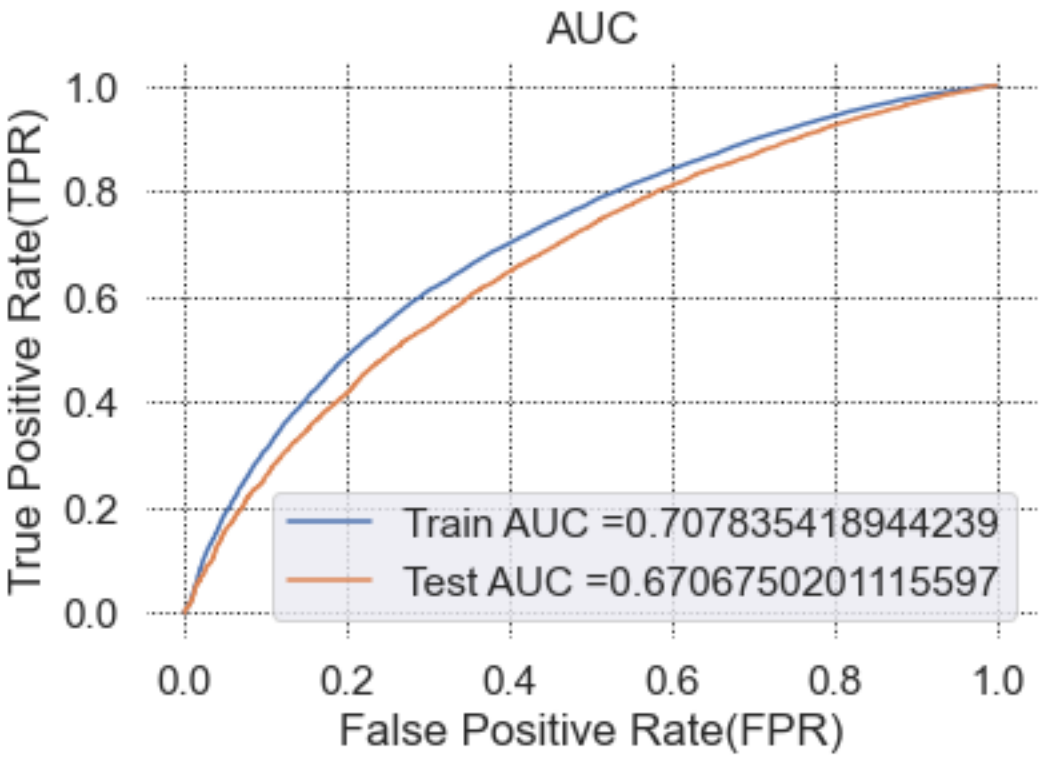
```
100%|
        | 14/14 [00:00<?, ?it/s]
```



0.4.1 TESTING WITH BEST HYPERPARAMETER VALUE ON SET 2¶

[20]:
```
mnb_bow_testModel = MultinomialNB(alpha = bestAlpha_2,class_prior=[0.5, 0.5])
mnb_bow_testModel.fit(x_tr_tfidf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability␣
 ↪estimates of the positive class
# not the predicted outputs
# y_train_pred = batch_predict(mnb_bow_testModel, x_train_onehot_bow)
y_train_pred=mnb_bow_testModel.predict_proba(x_tr_tfidf)[:,1]
# y_test_pred = batch_predict(mnb_bow_testModel, x_test_onehot_bow)
y_test_pred=mnb_bow_testModel.predict_proba(x_te_tfidf)[:,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)

ax = plt.subplot()

auc_set1_train=auc(train_fpr, train_tpr)
auc_set1_test=auc(test_fpr, test_tpr)


ax.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr,␣
 ↪train_tpr)))
ax.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
```

```
plt.title("AUC")
plt.grid(b=True, which='major', color='k', linestyle=':')
ax.set_facecolor("white")
plt.show()
```



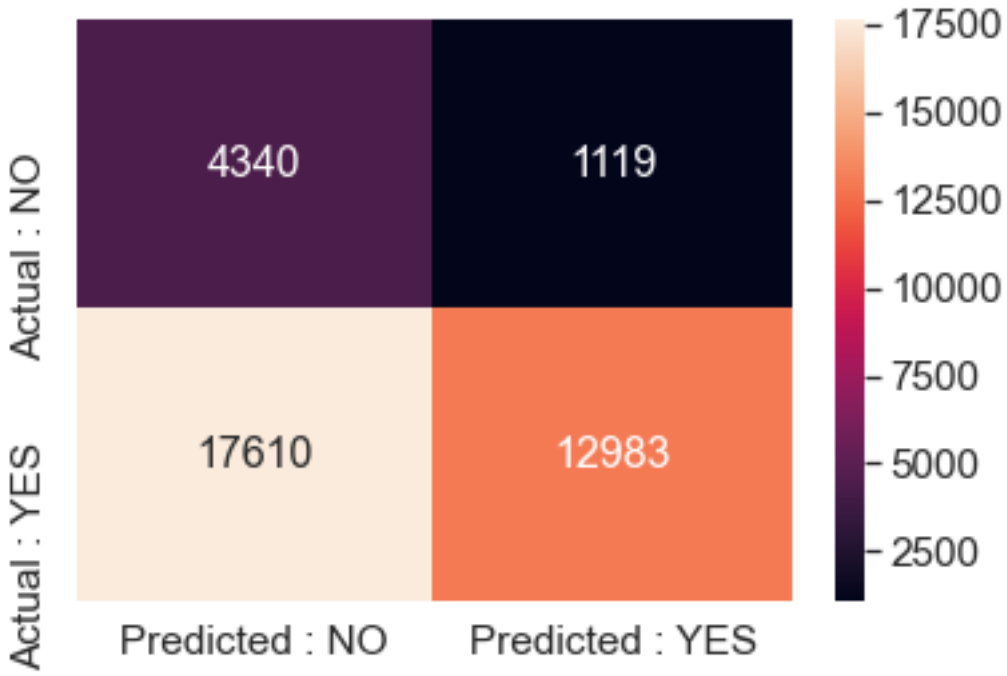
0.4.2 Confusion Matrix

[21]:
```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr,␣
 ↪test_tpr)))

conf_matr_df_test = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred,␣
 ↪tr_thresholds, test_fpr, test_tpr)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
xticklabels = ["Predicted : NO","Predicted : YES"]
yticklabels = ["Actual : NO","Actual : YES"]
sns.heatmap(conf_matr_df_test,xticklabels=xticklabels,yticklabels=yticklabels␣
 ↪,annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.3910625475058149 for threshold 0.597
[[ 4340  1119]
```

```
 [17610 12983]]
the maximum value of tpr*(1-fpr) 0.3910625475058149 for threshold 0.597
```

[21]: <AxesSubplot:>



0.5 Top 20 features from Set 1 :

[22]:
```
l = []

for i in vectorizer_essay.get_feature_names():      # appending all the␣
 ↪features into a single list
    l.append(i)

for i in vectorizer_teacher.get_feature_names():
    l.append(i)

for i in vectorizer_grade.get_feature_names():
    l.append(i)

for i in vectorizer_state.get_feature_names():
    l.append(i)

for i in vectorizer_categ.get_feature_names():
    l.append(i)
```

```
for i in vectorizer_sub_categ.get_feature_names():
    l.append(i)

l.append("price")
l.append("teacher_number_of_previously_posted_projects")

Total_features = len(l)
print(Total_features)
```

```
5101
```

0.6 Top 20 features (Negative)

[23]:
```
nb_bow = MultinomialNB(alpha=0.5,class_prior=[0.5,0.5])
nb_bow.fit(x_tr_bow, y_train)

feature_neg = {}
for i in range(Total_features):
    feature_neg[i] = nb_bow.feature_log_prob_[0,i]

print(len(feature_neg))
final_features_neg = pd.DataFrame({'feature_prob_estimates' : list(feature_neg.
 ↪values()),
'feature_names' : list(l)})

a = final_features_neg.sort_values(by = ['feature_prob_estimates'], ascending =␣
 ↪False)
a.head(20)
```

```
5101
```

[23]:

	feature_prob_estimates	feature_names
4000	-3.233630	students
3585	-4.321065	school
2303	-4.648746	learning
2772	-4.694874	my
682	-4.806502	classroom
2935	-4.994736	not
2244	-5.006784	learn
4466	-5.024025	they
1864	-5.044711	help
2793	-5.057298	my students
4405	-5.062789	the
5099	-5.156264	price
2841	-5.201670	nannan
2577	-5.243509	many
4779	-5.295334	we

2855	-5.352184	need
4880	-5.390898	work
772	-5.531273	come
5100	-5.545686	teacher_number_of_previously_posted_projects
2470	-5.597200	love

0.7 Top 20 features (Positive)

[24]:
```
feature_pos = {}
for i in range(Total_features):
    feature_pos[i] = nb_bow.feature_log_prob_[1,i]

print(len(feature_pos))
final_features_pos = pd.DataFrame({'feature_prob_estimates' : list(feature_pos.
 ↪values()),
'feature_names' : list(1)})

b = final_features_pos.sort_values(by = ['feature_prob_estimates'], ascending =␣
 ↪False)
b.head(20)
```

```
5101
```

[24]:

	feature_prob_estimates	feature_names
4000	-3.218647	students
3585	-4.361710	school
2772	-4.676848	my
2303	-4.726478	learning
682	-4.756782	classroom
4405	-4.984109	the
4466	-5.022468	they
2935	-5.023002	not
2793	-5.053421	my students
2244	-5.071260	learn
1864	-5.095638	help
5099	-5.209705	price
2577	-5.235851	many
2841	-5.254652	nannan
4779	-5.287762	we
2855	-5.364507	need
4880	-5.369783	work
3405	-5.376133	reading
4663	-5.432331	use
5100	-5.518684	teacher_number_of_previously_posted_projects

0.8 Summary

[26]:
```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Hyperparameter", "AUC"]

x.add_row(["BOW", "Multinomial Naive Bayes", bestAlpha_1, round(bestScore_1,␣
↪2)])
x.add_row(["TF-IDF", "Multinomial Naive Bayes", bestAlpha_2, round(bestScore_2,␣
↪2)])

print(x)
```

```
+------------+--------------------------+-----------------+------+
| Vectorizer |          Model           | Hyperparameter | AUC  |
+------------+--------------------------+-----------------+------+
|    BOW     | Multinomial Naive Bayes |     1e-05       | 0.69 |
|   TF-IDF   | Multinomial Naive Bayes |     1e-05       | 0.68 |
+------------+--------------------------+-----------------+------+
```