Assignment 6: Apply NB

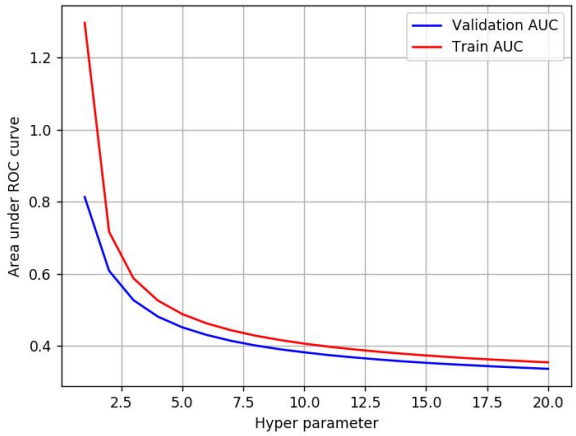
1. Minimum data points need to be considered for people having 4GB RAM is **50k** and for 8GB RAM is **100k**
2. When you are using ramdomsearchcv or gridsearchcv you need not split the data into X_train,X_cv,X_test. As the above methods use kfold. The model will learn better if train data is more so 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 (https://imgur.com/ldZA1zg) and this (https://ac-classroom-production.s3.amazonaws.com/public/COMMENT/Annotation_2020-05-21_225912_0lyZzN8.jpg)

    - 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)**

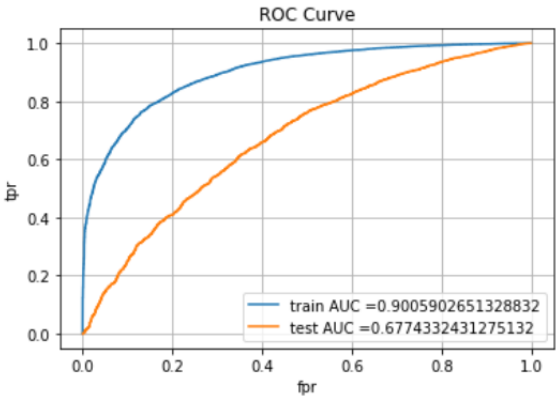
    - Consider alpha values in range: 10^-5 to 10^2 like [0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,100]
    - Explore class_prior = [0.5, 0.5] parameter which can be present in MultinomialNB function(go through this (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html)) then check how results might change.
    - 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
    - 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 (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 = ??

-plot the confusion matrix in heatmaps, while plotting the confusion matrix go through the link (https://stackoverflow.com/questions/61748441/how-to-fix-the-values-displayed-in-a-confusion-matrix-in-exponential-form-to-nor)

7. find the top 20 features from either from feature Set 1 or feature Set 2 using values of `feature_log_prob_ ` parameter of `MultinomialNB` (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 (https://imgur.com/mWvE7gj)
8. You need to summarize the results at the end of the notebook, summarize it in the table format

Vectorizer	Model	Hyper parameter	AUC
BOW	Brute	7	0.78
TFIDF	Brute	12	0.79
W2V	Brute	10	0.78
TFIDFW2V	Brute	6	0.78

In [ ]:

2. Naive Bayes

1.1 Loading Data

In [137]:
```
import pandas
data = pandas.read_csv('preprocessed_data.csv')
```

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

In [138]:
```
# please write all the code with proper documentation, and proper titles for each
# 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 debuggi
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reade
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

1.3 Make Data Model Ready: encoding eassay, and project_title

In [139]:
```
# please write all the code with proper documentation, and proper titles for each
# 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 debuggi
# 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 reade
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

1.4 Make Data Model Ready: encoding numerical, categorical features

In [140]:
```
# please write all the code with proper documentation, and proper titles for each
# 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 debuggi
# 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
```

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

In [141]:
```
# please write all the code with proper documentation, and proper titles for each
# 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 debuggi
# 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
```

3. Summary

as mentioned in the step 5 of instructions

In [142]:
```
#!pip install chart-studio
```

In [143]:
```
from chart_studio.plotly import plotly
#import plotly.offline as offline
```

In [144]:
```
# this all libraries code taken from reperence file.
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

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

import re


import pickle
from tqdm import tqdm
import os

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

Minimum data points need to be considered for people having 4GB RAM is 50k and for 8GB RAM is 100k

In [145]:
```
data  = pd.read_csv('preprocessed_data.csv', nrows=50000)
data.head()
```

Out[145]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projec
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	

In [146]:
```
# this code is taken from https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won\'t", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase


test = "Hey I'm Yann, how're you and how's it going ? That's interesting: I'd lov
print(decontracted(test))
```

```
Hey I am Yann, how are you and how is it going ? That is interesting: I would l
ove to hear more about it.
```

In [147]:
```
# https://gist.github.com/sebleier/554280
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itse
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'ha
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 't
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma',
            'won', "won't", 'wouldn', "wouldn't"]
```

In [148]:
```
#code is taken by reference file
# Combining all the above stundents
from tqdm import tqdm
def preprocess_text(text_data):
    preprocessed_text = []
    # tqdm is for printing the status bar
    for sentance in tqdm(text_data):
        sent = decontracted(sentance)
        sent = sent.replace('\\r', ' ')
        sent = sent.replace('\\n', ' ')
        sent = sent.replace('\\"', ' ')
        sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
        # https://gist.github.com/sebleier/554280
        sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
        preprocessed_text.append(sent.lower().strip())
    return preprocessed_text
```

In [149]:
```
preprocessed_essay = preprocess_text(data['essay'].values)
```

```
100%|██████████████████████████████████████████████████████████████████████████████
| 50000/50000 [00:36<00:00, 1367.00it/s]
```

In [150]:
```
data['preprocessed_essays'] = preprocessed_essay
data.drop(['essay'], axis=1, inplace=True)
```

In [151]:
```
data.head(2)
```

Out[151]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projec
0	ca	mrs	grades_prek_2	
1	ut	ms	grades_3_5	

In [152]:
```
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
X.head(1)
```

Out[152]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projec
0	ca	mrs	grades_prek_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.

In [153]:
```
# 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, strati
```

bow to essay

In [154]:
```
#code is taken from reference file
print(X_train.shape, y_train.shape)
#print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)


vectorizer = CountVectorizer(min_df=10)#,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['preprocessed_essays'].values) # fit has to happen only o
vectorizer_essay_bow=vectorizer
# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['preprocessed_essays'].values)
#X_cv_essay_bow = vectorizer.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer.transform(X_test['preprocessed_essays'].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)
```

```
(33500, 8) (33500,)
(16500, 8) (16500,)
====================================================================================
====================
After vectorizations
(33500, 10255) (33500,)
(16500, 10255) (16500,)
====================================================================================
====================
```

tfidf for essay

In [196]:
```
# tfidf on preprocessed_essay values
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_essay_tfidf = TfidfVectorizer(min_df=10)
vectorizer_essay_tfidf.fit(X_train['preprocessed_essays'])

X_train_essays_tfidf = vectorizer_essay_tfidf.transform(X_train['preprocessed_es
X_test_essays_tfidf  = vectorizer_essay_tfidf.transform(X_test['preprocessed_ess
print(X_train_essays_tfidf.shape,y_train.shape)
print(X_test_essays_tfidf.shape,y_test.shape)
```

```
(33500, 10255) (33500,)
(16500, 10255) (16500,)
```

categorical features

In [156]:
```
#code taken from reference file
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values) # fitting train dataset

vectorizer_state =vectorizer
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 51) (33500,)
(16500, 51) (16500,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia',
'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms',
'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or', 'pa',
'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'wy']
====================================================================================
====================
```

In [157]:
```
#code taken from reference file
vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values)
vectorizer_prefix=vectorizer
print("vectorizer_states \n",vectorizer_prefix.get_feature_names())
#teacher prefix text converting into vector.
X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
vectorizer_states
 ['dr', 'mr', 'mrs', 'ms', 'teacher']
After vectorizations
(33500, 5) (33500,)
(16500, 5) (16500,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
====================================================================================
====================
```

In [158]:
```
#code taken from reference file
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values)

vectorizer_grade=vectorizer
X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].value
#X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
#print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 4) (33500,)
(16500, 4) (16500,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
====================================================================================
======================
```

In [159]:
```
#code taken from reference file
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values)
vectorizer_categories=vectorizer

X_train_categories_ohe = vectorizer.transform(X_train['clean_categories'].values

X_test_categories_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_categories_ohe.shape, y_train.shape)

print(X_test_categories_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 9) (33500,)
(16500, 9) (16500,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy
_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
====================================================================================
======================
```

In [160]:
```
#code taken from reference file
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values)
vectorizer_subcategories=vectorizer
X_train_subcategories_ohe = vectorizer.transform(X_train['clean_subcategories'].
X_test_subcategories_ohe = vectorizer.transform(X_test['clean_subcategories'].va

print("After vectorizations")
print(X_train_subcategories_ohe.shape, y_train.shape)

print(X_test_subcategories_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 30) (33500,)
(16500, 30) (16500,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'env
ironmentalscience', 'esl', 'extracurricular', 'financialliteracy', 'foreignlang
uages', 'gym_fitness', 'health_lifescience', 'health_wellness', 'history_geogra
phy', 'literacy', 'literature_writing', 'mathematics', 'music', 'nutritioneduca
tion', 'other', 'parentinvolvement', 'performingarts', 'socialsciences', 'speci
alneeds', 'teamsports', 'visualarts', 'warmth']
====================================================================================
====================
```

numerical values

In [161]:
```
#code taken from reference file
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_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1)).r



print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

In [162]:
```
#code taken from reference file
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.re

X_train_teacher_number_of_previously_posted_projects_norm = normalizer.transform

X_test_teacher_number_of_previously_posted_projects_norm = normalizer.transform(

print("After vectorizations")
print(X_train_teacher_number_of_previously_posted_projects_norm.shape, y_train.sh

print(X_test_teacher_number_of_previously_posted_projects_norm.shape, y_test.sha
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

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

In [163]:
```
# preparing set1 dataset
#code taken from reference file
from scipy.sparse import hstack
X_tr_bow = hstack((X_train_essay_bow,X_train_state_ohe,X_train_teacher_ohe, X_tra

X_te_bow = hstack((X_test_essay_bow,X_test_state_ohe,X_test_teacher_ohe, X_test_
print("Final Data matrix")
print(X_tr_bow.shape, y_train.shape)

print(X_te_bow.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(33500, 10356) (33500,)
(16500, 10356) (16500,)
====================================================================================================
```

Set 2: categorical, numerical features + preprocessed_eassay (TFIDF)

In [197]:
```
#preparing set2 dataset
X_tr_tfidf = hstack((X_train_essays_tfidf,X_train_state_ohe,X_train_teacher_ohe,

X_te_tfidf = hstack((X_test_essays_tfidf,X_test_state_ohe,X_test_teacher_ohe, X_t
print("Final Data matrix")
print(X_tr_tfidf.shape, y_train.shape)
print(X_te_tfidf.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(33500, 10356) (33500,)
(16500, 10356) (16500,)
====================================================================================================
```

In [ ]:

In [ ]:

grid search

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

In [165]:
```
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
```

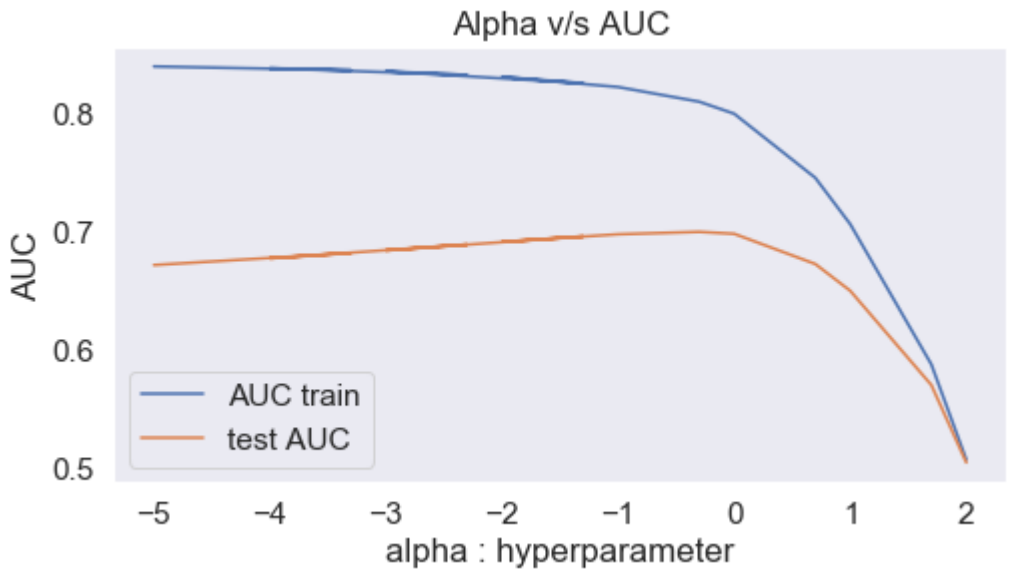
In [166]:
```
mn_bow=MultinomialNB(class_prior=[0.5,0.5])
parameters={'alpha':[0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10
clf=GridSearchCV(mn_bow, parameters,cv=10,scoring='roc_auc',verbose=1,return_tra
clf.fit(X_tr_bow,y_train)
tr_auc=clf.cv_results_['mean_train_score']
tr_auc_std=clf.cv_results_['std_test_score']
te_auc = clf.cv_results_['mean_test_score']
te_auc_std= clf.cv_results_['std_test_score']
best_alpha1=clf.best_params_['alpha']
best_score1=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

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done 140 out of 140 | elapsed:   15.0s finished

Best alpha:  0.5 Best score:  0.6995476072236618
```

You need to plot the performance of model both on train data and cross validation data for each hyper parameter

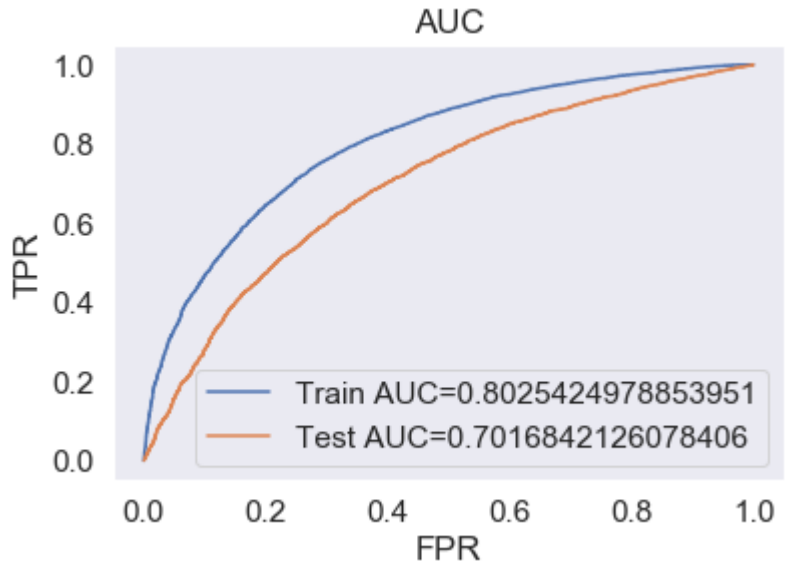
In [167]:
```
alpha=[0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,100]
log_alpha=[]
for a in tqdm(alpha):
    b=np.log10(a)
    log_alpha.append(b)
plt.figure(figsize=(8,4))
plt.plot(log_alpha,tr_auc,label="AUC train")
plt.plot(log_alpha,te_auc, label='test AUC')
plt.title("Alpha v/s AUC")
plt.xlabel("alpha : hyperparameter")
plt.ylabel("AUC")
plt.legend()
plt.grid()
plt.show()
```

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



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.

In [190]:
```
from sklearn.metrics import roc_curve, auc
mn_bow_testModel = MultinomialNB(alpha = best_alpha1,class_prior=[0.5, 0.5])
mn_bow_testModel.fit(X_tr_bow, y_train)
y_train_pred=mn_bow_testModel.predict_proba(X_tr_bow)[:,1]
y_test_pred=mn_bow_testModel.predict_proba(X_te_bow)[:,1]
train_fpr,train_tpr,train_theshold=roc_curve(y_train,y_train_pred)
test_fpr,test_tpr,test_theshold=roc_curve(y_test,y_test_pred)
ax=plt.subplot()
auc1_train=auc(train_fpr,train_tpr)
auc1_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("FPR")
plt.ylabel("TPR")
plt.title("AUC")
plt.grid()
plt.show()
```



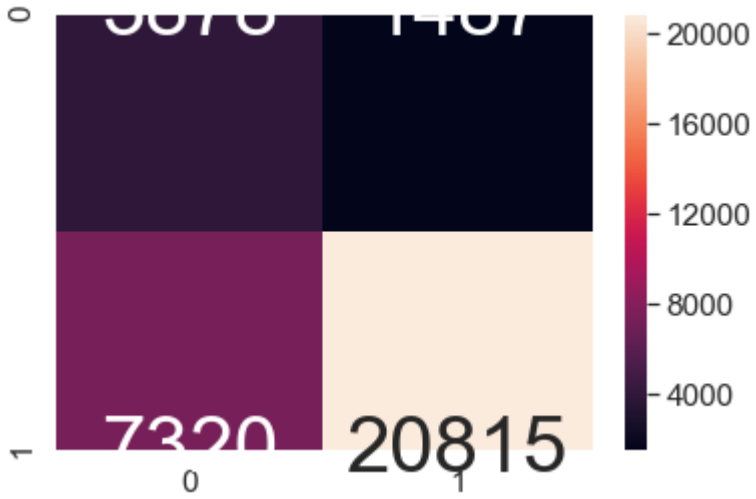
In [169]:
```
def pre(proba,threshould,fpr,tpr):
    a=threshould[np.argmax(tpr*(1-fpr))]
    print("the maximum value of tpr*(1-fpr)",max(tpr*(1-fpr)), "for threshold ",
    predictions=[]
    for i in proba:
        if i>+a:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

confusion matrix for train dataset.

In [170]:
```
from sklearn.metrics import confusion_matrix
print("Confusion Matrix for Train dataset")
print(confusion_matrix(y_train,pre(y_train_pred,train_theshold,train_fpr,train_t
matrix_train=pd.DataFrame(confusion_matrix(y_train,pre(y_train_pred,train_thesho
sns.set(font_scale=1.4)
sns.heatmap(matrix_train,annot=True,annot_kws={"size":40}, fmt='g')
```

```
Confusion Matrix for Train dataset
the maximum value of tpr*(1-fpr) 0.5347963544824738 for threshold  0.4
[[ 3878  1487]
 [ 7320 20815]]
the maximum value of tpr*(1-fpr) 0.5347963544824738 for threshold  0.4
```

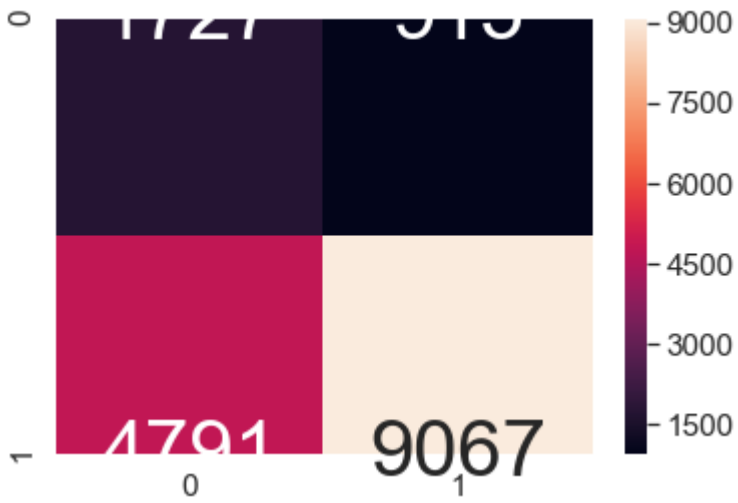
Out[170]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1e3f4dd9208>
```



In [171]:
```
from sklearn.metrics import confusion_matrix
print("Confusion Matrix for Test dataset")
print(confusion_matrix(y_test,pre(y_test_pred,test_theshold,test_fpr,test_tpr)))
matrix_train=pd.DataFrame(confusion_matrix(y_test,pre(y_test_pred,test_theshold,
sns.set(font_scale=1.4)
sns.heatmap(matrix_train,annot=True,annot_kws={"size":40}, fmt='g')
```

```
Confusion Matrix for Test dataset
the maximum value of tpr*(1-fpr) 0.4277307554104795 for threshold  0.65
[[1727  915]
 [4791 9067]]
the maximum value of tpr*(1-fpr) 0.4277307554104795 for threshold  0.65
```

Out[171]: <matplotlib.axes._subplots.AxesSubplot at 0x1e3f57a0a88>



In [172]:
```
# top 20 positive features in set1
```

In [173]:
```
all_feature_names_bow=[]
## FOR SET 1 and SET 2
for i in vectorizer_categories.get_feature_names():
    all_feature_names_bow.append(i)

for i in vectorizer_subcategories.get_feature_names():
    all_feature_names_bow.append(i)

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

for i in vectorizer_prefix.get_feature_names():
    all_feature_names_bow.append(i)

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



for i in vectorizer_essay_bow.get_feature_names():
    all_feature_names_bow.append(i)

all_feature_names_bow.append("price")

all_feature_names_bow.append("teacher_number_of_previously_posted_projects")
```

In [174]:
```
print(len(all_feature_names_bow))
```

```
10356
```

In [203]:
```
total_features_bow=len(all_feature_names_bow)
```

In [204]:
```
X_tr_bow.shape
```

Out[204]:
```
(33500, 10356)
```

In [205]:
```
nb_bow=MultinomialNB(alpha=0.5,class_prior=[0.5,0.5])
nb_bow.fit(X_tr_bow,y_train)
```

Out[205]:
```
MultinomialNB(alpha=0.5, class_prior=[0.5, 0.5], fit_prior=True)
```

In [237]:
```
bow_features_negative = {}
for a in range(total_features_bow) :
# for a in range(101) :
    bow_features_negative[a] = nb_bow.feature_log_prob_[0,a]

bow_features_negative = pd.DataFrame({'feature_probability_estimates' : list(bow_
'feature_name' : list(all_feature_names_bow)})

features_bow =bow_features_negative.sort_values(by = ['feature_probability_estima
```

In [238]:
```
print("TOP 20 Negative features - BOW")
features_bow.head(20)
```

```
TOP 20 Negative features - BOW
```

Out[238]:

	feature_probability_estimates	feature_name
8871	-2.977304	stix
8043	-4.076240	sacks
5301	-4.379624	labor
1705	-4.498591	chef
6187	-4.699686	needles
5297	-4.740442	label
4360	-4.778680	handout
5617	-4.976998	luckiest
6045	-4.988801	moon
6084	-5.129443	movement
10149	-5.169132	whichever
1825	-5.287171	climb
10216	-5.314386	wishing
176	-5.325031	34
7394	-5.357375	questioners
5501	-5.362626	linear
2385	-5.370848	cuba
9955	-5.378246	vibrant
1693	-5.394115	checkers
5667	-5.405344	maintains

In [239]:
```
bow_features_pv = {}
for a in range(total_features_bow) :
# for a in range(101) :
    bow_features_pv[a] = nb_bow.feature_log_prob_[1,a]

len(bow_features_probs_pos)

final_bow_positive_features = pd.DataFrame({'feature_probability_estimates' : li
'feature_name' : list(all_feature_names_bow)})

features_positive_bow = final_bow_positive_features.sort_values(by = ['feature_p

print("TOP 20 Positive features - BOW")
features_positive_bow.head(20)
```

```
TOP 20 Positive features - BOW
```

Out[239]:

	feature_probability_estimates	feature_name
8871	-2.973923	stix
8043	-4.122368	sacks
5301	-4.476625	labor
1705	-4.478098	chef
6187	-4.760730	needles
5297	-4.795264	label
4360	-4.843893	handout
5617	-5.005115	luckiest
6045	-5.060815	moon
7394	-5.117662	questioners
10149	-5.128573	whichever
6084	-5.150334	movement
9760	-5.213443	unfortunately
2385	-5.287747	cuba
176	-5.291066	34
5501	-5.307232	linear
10216	-5.359398	wishing
1693	-5.361231	checkers
1825	-5.364383	climb
10186	-5.407655	wimpy

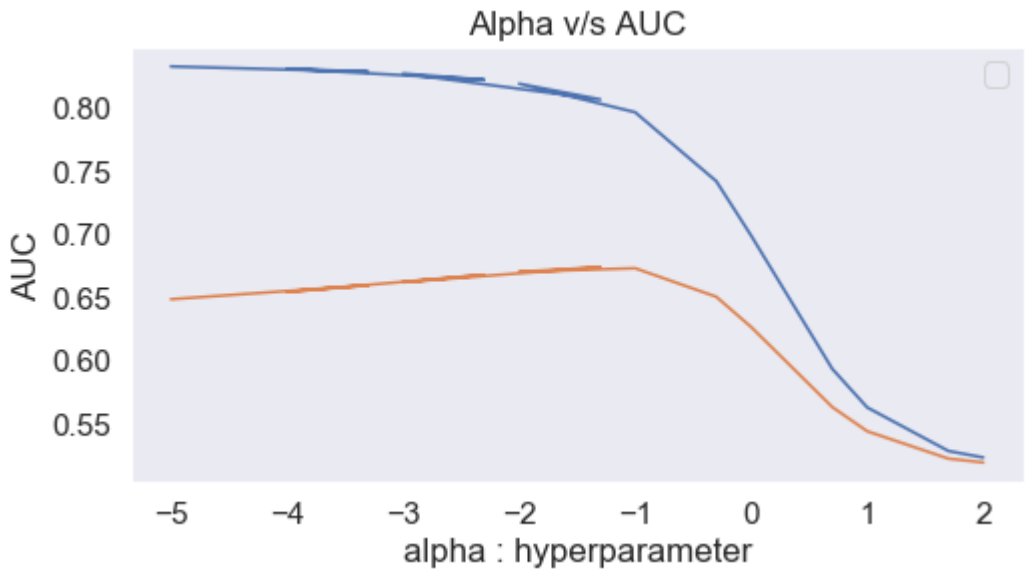
set 2

In [198]:
```
mn_tfidf=MultinomialNB(class_prior=[0.5,0.5])
parameters={'alpha':[0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10
clf=GridSearchCV(mn_tfidf, parameters,cv=10,scoring='roc_auc',verbose=1,return_t
clf.fit(X_tr_tfidf,y_train)
tr_auc=clf.cv_results_['mean_train_score']
tr_auc_std=clf.cv_results_['std_test_score']
te_auc = clf.cv_results_['mean_test_score']
te_auc_std= clf.cv_results_['std_test_score']
best_alpha2=clf.best_params_['alpha']
best_score2=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

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done 140 out of 140 | elapsed:   13.7s finished

Best alpha:  0.05 Best score:  0.6736836758548532
```

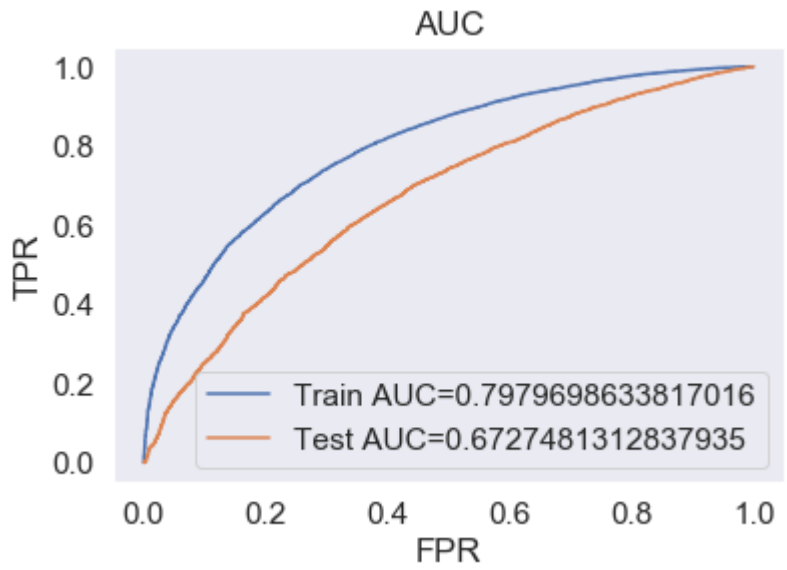
In [199]:
```
alpha=[0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,100]
log_alpha=[]
for a in tqdm(alpha):
    b=np.log10(a)
    log_alpha.append(b)
plt.figure(figsize=(8,4))
plt.legend()
plt.plot(log_alpha,tr_auc,label="AUC train")
plt.plot(log_alpha,te_auc, label='test AUC')
plt.title("Alpha v/s AUC")
plt.xlabel("alpha : hyperparameter")
plt.ylabel("AUC")
#plt.legend()
plt.grid()
plt.show()
```

```
100%|██████████| 14/14 [00:00<?, ?it/s]
No handles with labels found to put in legend.
```



In [200]:
```
from sklearn.metrics import roc_curve, auc
mn_tfidf_testModel = MultinomialNB(alpha = best_alpha2,class_prior=[0.5, 0.5])
mn_tfidf_testModel.fit(X_tr_tfidf, y_train)
y_train_pred=mn_tfidf_testModel.predict_proba(X_tr_tfidf)[:,1]
y_test_pred=mn_tfidf_testModel.predict_proba(X_te_tfidf)[:,1]
train_fpr,train_tpr,train_theshold=roc_curve(y_train,y_train_pred)
test_fpr,test_tpr,test_theshold=roc_curve(y_test,y_test_pred)
ax=plt.subplot()
auc2_train=auc(train_fpr,train_tpr)
auc2_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("FPR")
plt.ylabel("TPR")
plt.title("AUC")
plt.grid()
plt.show()
```

```
0.7979698633817016
0.6727481312837935
```



In [201]:
```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Hyperparameter: Alpha", "Train AUC", "T

x.add_row(["BOW", "Multinomial Naive Bayes", best_alpha1, round(auc1_train,2),ro
x.add_row(["TF-IDF", "Multinomial Naive Bayes", best_alpha2, round(auc2_train,2)

print(x)
```

```
+------------+------------------------+----------------------+-----------+---
-------+
| Vectorizer |          Model         | Hyperparameter: Alpha | Train AUC | Te
st AUC |
+------------+------------------------+----------------------+-----------+---
-------+
|    BOW     | Multinomial Naive Bayes |          0.5          |    0.8    |
0.7    |
|   TF-IDF   | Multinomial Naive Bayes |          0.05         |    0.8    |
0.67   |
+------------+------------------------+----------------------+-----------+---
-------+
```

In [240]:
```
#!pip install prettytable
```

```
Requirement already satisfied: prettytable in c:\users\win10\anaconda3\lib\site
-packages (2.0.0)
Requirement already satisfied: wcwidth in c:\users\win10\anaconda3\lib\site-pac
kages (from prettytable) (0.1.7)
Requirement already satisfied: setuptools in c:\users\win10\anaconda3\lib\site-
packages (from prettytable) (41.4.0)
```

In [240]:
```
best_alpha2
```

Out[240]: 0.05

In [241]:
```
auc1_train
```

Out[241]: 0.8025424978853951

In [242]:
```
auc2_train
```

Out[242]: 0.7979698633817016

In [ ]: