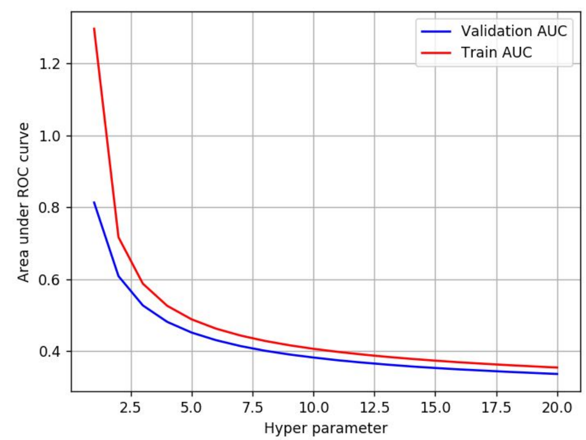
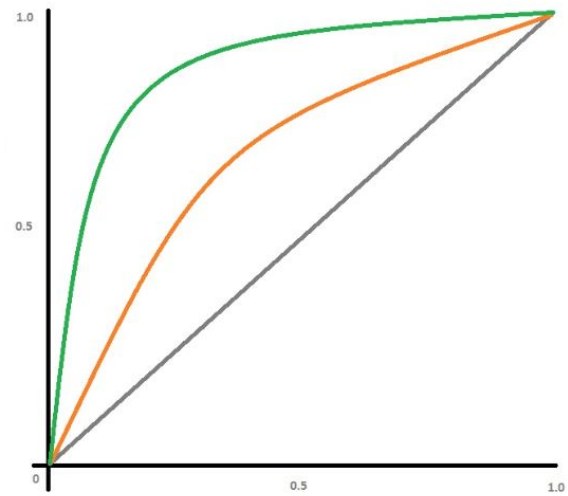
Assignment 6: Apply NB

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



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



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

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

4. fine the top 20 features from either from feature Set 1 or feature Set 2 using absolute values of `feature_log_prob_ ` parameter of `MultinomialNB` (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html) and print their corresponding feature names
5. You need to summarize the results at the end of the notebook, summarize it in the table format

2. Naive Bayes

1.1 Loading Data

In [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
```

In [2]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
import math
```

In [3]:
```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
import plotly
```

In [4]:
```
from chart_studio import plotly
```

In [5]:
```
#reading the data
```

In [6]:
```
data = pd.read_csv('preprocessed_data.csv')
data.shape
```

Out[6]:
```
(109248, 9)
```

In [7]:
```
data.head()
```

Out[7]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categori
0	ca	mrs	grades_prek_2	53	1	math_scien
1	ut	ms	grades_3_5	4	1	specialnee
2	ca	mrs	grades_prek_2	10	1	literacy_langua
3	ga	mrs	grades_prek_2	2	1	appliedlearni
4	wa	mrs	grades_3_5	2	1	literacy_langua

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

In [8]:
```
y = data['project_is_approved'].values #making the required label and gievn the calss abel as y
X = data.drop(['project_is_approved'], axis=1)
X.head(1)#printing the data
```

Out[8]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategori
0	ca	mrs	grades_prek_2	53	math_science	appliedscienc health_lifescien

In [9]:
```
# train test split
#this code is taken from the refrence notebook
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

In [10]:
```
#printing the shapes of the x,y ies train,test and cv data
```

In [11]:
```
X_train.shape
```

Out[11]: (49041, 8)

In [12]:
```
X_cv.shape
```

Out[12]: (24155, 8)

In [13]:
```
X_test.shape
```

Out[13]: (36052, 8)

In [14]:
```
y_train.shape
```

Out[14]:
```
(49041,)
```

In [15]:
```
y_test.shape
```

Out[15]:
```
(36052,)
```

In [16]:
```

y_cv.shape
```

Out[16]:
```
(24155,)
```

1.3 Make Data Model Ready: encoding eassay

BOW

In [17]:
```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)
```

```
(49041, 8) (49041,)
(24155, 8) (24155,)
(36052, 8) (36052,)
====================================================================================================
```

In [19]:
```
vectorizer_bow = CountVectorizer(min_df=10)
vectorizer_bow.fit(X_train['essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer_bow.transform(X_train['essay'].values)
X_cv_essay_bow = vectorizer_bow.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer_bow.transform(X_test['essay'].values)
```

In [24]:
```
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)
#print(vectorizer_bow.get_feature_names())
#printing the featured names in the essay
```

```
After vectorizations
(49041, 12164) (49041,)
(24155, 12164) (24155,)
(36052, 12164) (36052,)
====================================================================================================
```

TFIDF

In [25]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf = TfidfVectorizer(min_df=15)
vectorizer_tfidf.fit(X_train['essay'])

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf=vectorizer_tfidf.transform(X_train['essay'].values)
X_test_essay_tfidf=vectorizer_tfidf.transform(X_test['essay'].values)
X_cv_essay_tfidf=vectorizer_tfidf.transform(X_cv['essay'].values)
```

In [26]:
```
print("after vectorization")
print(X_train_essay_tfidf.shape,y_train.shape)
print(X_test_essay_tfidf.shape,y_test.shape)
print(X_cv_essay_tfidf.shape,y_cv.shape)
```

```
after vectorization
(49041, 10255) (49041,)
(36052, 10255) (36052,)
(24155, 10255) (24155,)
```

In [ ]:
```

```

1.4 Make Data Model Ready: encoding numerical, categorical features

In [27]:
```
# 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

one hot encoding the catogorical features: school_state

In [29]:
```
vectorizer_ss = CountVectorizer()
vectorizer_ss.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer_ss.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer_ss.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer_ss.transform(X_test['school_state'].values)

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

```
After vectorizations
(49041, 51) (49041,)
(24155, 51) (24155,)
(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', 'o
k', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'wy']
====================================================================================================
```

one hot encoding the catogorical features :teacher_prefix

In [30]:
```
vectorizer_tp = CountVectorizer()
vectorizer_tp.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data

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

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

```
After vectorizations
(49041, 5) (49041,)
(24155, 5) (24155,)
(36052, 5) (36052,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
====================================================================================================
```

one hot encoding the catogorical features: project_grade_category

In [31]:
```
vectorizer_pgc = CountVectorizer()
vectorizer_pgc.fit(X_train['project_grade_category'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer_pgc.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vectorizer_pgc.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer_pgc.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_pgc.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 4) (49041,)
(24155, 4) (24155,)
(36052, 4) (36052,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
====================================================================================================
```

one hot encoding the catogorical features: clean_categories

In [32]:
```python
vectorizer_cc = CountVectorizer()
vectorizer_cc.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_ccat_ohe = vectorizer_cc.transform(X_train['clean_categories'].values)
X_cv_ccat_ohe = vectorizer_cc.transform(X_cv['clean_categories'].values)
X_test_ccat_ohe = vectorizer_cc.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_ccat_ohe.shape, y_train.shape)
print(X_cv_ccat_ohe.shape, y_cv.shape)
print(X_test_ccat_ohe.shape, y_test.shape)
print(vectorizer_cc.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 9) (49041,)
(24155, 9) (24155,)
(36052, 9) (36052,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'mu
sic_arts', 'specialneeds', 'warmth']
====================================================================================================
```

one hot encoding the catogorical features: clean_subcategories

In [33]:
```
vectorizer_csc= CountVectorizer()
vectorizer_csc.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_csubcat_ohe = vectorizer_csc.transform(X_train['clean_subcategories'].values)
X_cv_csubcat_ohe = vectorizer_csc.transform(X_cv['clean_subcategories'].values)
X_test_csubcat_ohe = vectorizer_csc.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_csubcat_ohe.shape, y_train.shape)
print(X_cv_csubcat_ohe.shape, y_cv.shape)
print(X_test_csubcat_ohe.shape, y_test.shape)
print(vectorizer_csc.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 30) (49041,)
(24155, 30) (24155,)
(36052, 30) (36052,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep', 'community
service', 'earlydevelopment', 'economics', 'environmentalscience', 'esl', 'extracurricular', 'financialliterac
y', 'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_wellness', 'history_geography', 'literac
y', 'literature_writing', 'mathematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'perform
ingarts', 'socialsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
====================================================================================================
```

Numerical features

Normalising numerical features: price

In [46]:
```
import warnings

warnings.filterwarnings("ignore")
from sklearn.preprocessing import StandardScaler
#https://machinelearningmastery.com/rescaling-data-for-machine-learning-in-python-with-scikit-learn/
standard_vec_p = StandardScaler(with_mean = False)
# 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.

standard_vec_p.fit(X_train['price'].values.reshape(-1,1))

X_train_price_std = standard_vec_p.transform(X_train['price'].values.reshape(-1,1))
X_cv_price_std = standard_vec_p.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_std = standard_vec_p.transform(X_test['price'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_price_std.shape, y_train.shape)
print(X_cv_price_std.shape, y_cv.shape)
print(X_test_price_std.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================================================
```

Normalising numerical features: teacher_number_of_previously_posted_projects

In [47]:
```
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import StandardScaler
#https://machinelearningmastery.com/rescaling-data-for-machine-learning-in-python-with-scikit-learn/
standard_vec_tpps = StandardScaler(with_mean = False)
# 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.
standard_vec_tpps.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

X_train_pre_project_std = standard_vec_tpps.transform(X_train['teacher_number_of_previously_posted_projects'].va
X_cv_pre_project_std = standard_vec_tpps.transform(X_cv['teacher_number_of_previously_posted_projects'].values.r
X_test_pre_project_std = standard_vec_tpps.transform(X_test['teacher_number_of_previously_posted_projects'].valu

print("After vectorizations")
print(X_train_pre_project_std.shape, y_train.shape)
print(X_cv_pre_project_std.shape, y_cv.shape)
print(X_test_pre_project_std.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================================================
```

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

Concatinating all the features

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

In [49]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
#here in set 1 i am consideribng the essay_bow ,all categoricala nd all numerical values
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_bow, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe,
               X_train_ccat_ohe,X_train_csubcat_ohe, X_train_price_std,X_train_pre_project_std)).tocsr()

X_cr = hstack((X_cv_essay_bow, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe,
               X_cv_ccat_ohe,X_cv_csubcat_ohe, X_cv_price_std,X_cv_pre_project_std)).tocsr()

X_te = hstack((X_test_essay_bow, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe,
               X_test_ccat_ohe,X_test_csubcat_ohe,X_test_price_std,X_test_pre_project_std)).tocsr()

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

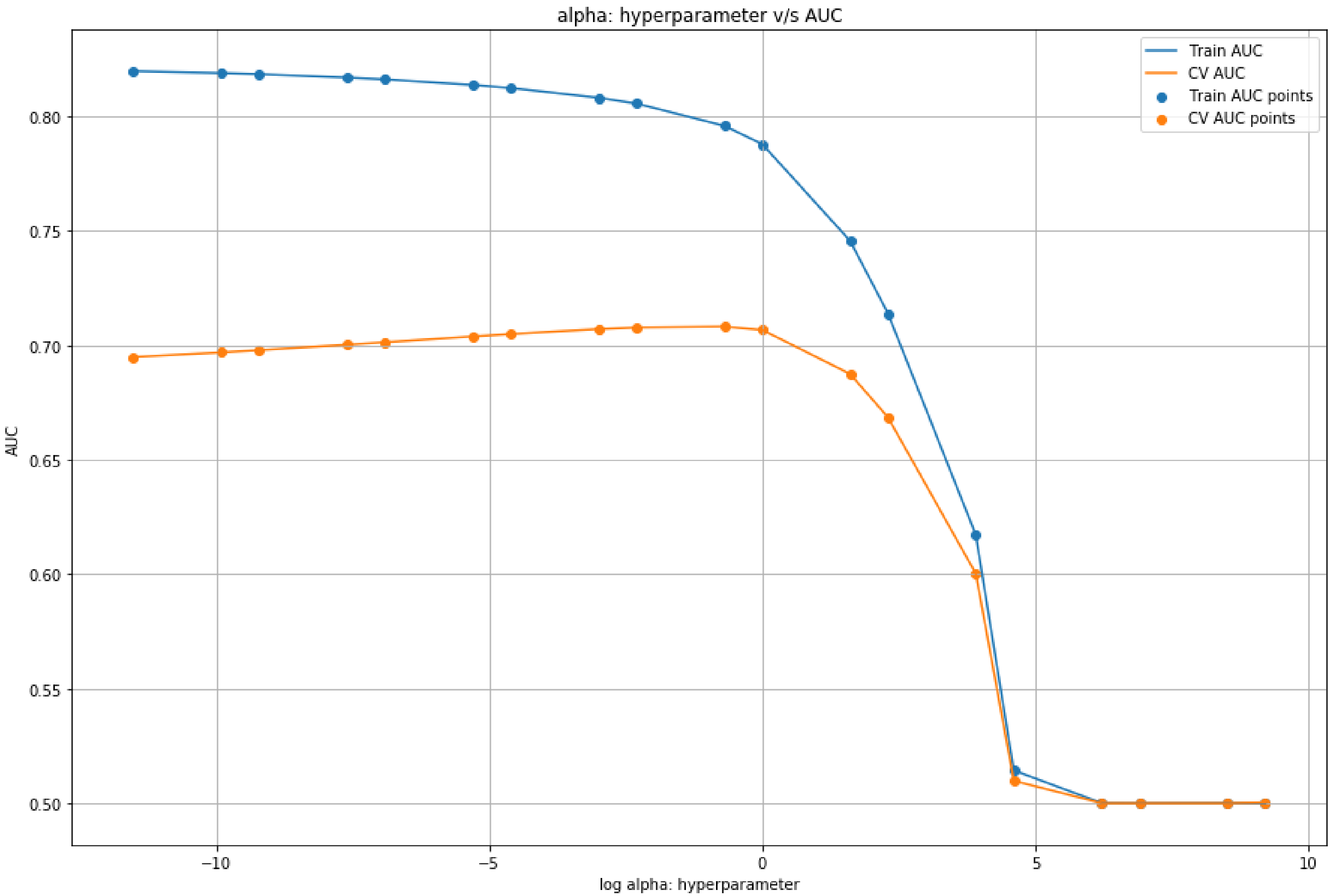
```
Final Data matrix
(49041, 12265) (49041,)
(24155, 12265) (24155,)
(36052, 12265) (36052,)
====================================================================================================
```

In [50]:
```
def batch_predict(clf, data): #using the batch predition metod
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [51]:
```
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
import math

train_auc = []
cv_auc = []
log_alphas = []

alphas = [10000,5000,1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001,0.00005,0.00001]

for i in tqdm(alphas):
    nb = MultinomialNB(alpha = i,class_prior=[0.5,0.5]) #defing the classification model
    nb.fit(X_tr, y_train) #fiting the model with train data set

    y_train_pred = batch_predict(nb, X_tr)
    y_cv_pred = batch_predict(nb, X_cr)


    # 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))

for a in tqdm(alphas): #using log function
    b = math.log(a)
    log_alphas.append(b)
```

```
100%|██████████████████████████████████████████████████| 19/19 [00:04<00:00,
4.36it/s]
100%|██████████████████████████████████████████████████████| 19/19 [00:00
<?, ?it/s]
```

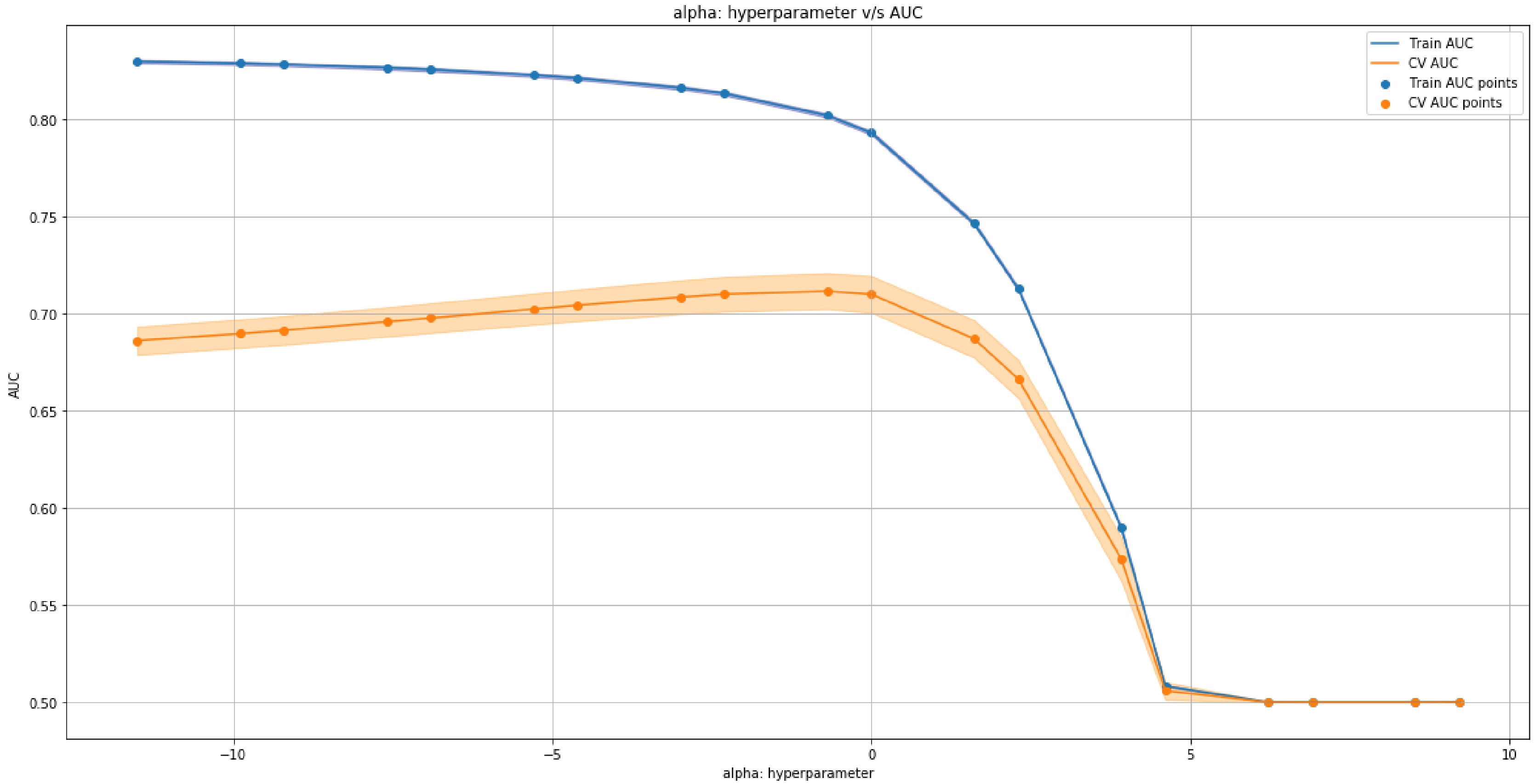
In [52]:
```
# ploting the graphn b/w train and cv
plt.figure(figsize=(15,10))
plt.plot(log_alphas, train_auc, label='Train AUC')
plt.plot(log_alphas, cv_auc, label='CV AUC')

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

plt.legend()
plt.xlabel("log alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("alpha: hyperparameter v/s AUC")
plt.grid()
plt.show()
```


alpha: hyperparameter v/s AUC
Train AUC
CV AUC
Train AUC points
CV AUC points
AUC
0.80
0.75
0.70
0.65
0.60
0.55
0.50
−10
−5
0
5
10
log alpha: hyperparameter

In [55]:
```
from sklearn.model_selection import GridSearchCV

#using the grid search cv for the creoss validation of the model
nb = MultinomialNB()

parameters = {'alpha':[10000,5000,1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001,0.00005,0.0
#taking the valies of alpha fro 10^4 to 10^-4 for the better hyperparaeter  tunning
clf = GridSearchCV(nb, parameters, cv= 10, scoring='roc_auc',return_train_score=True)
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [56]:
```python
alphas = [10000,5000,1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001,0.00005,0.00001]
log_alphas =[]

for a in tqdm(alphas):
    b = math.log(a)
    log_alphas.append(b)

plt.figure(figsize=(20,10))

plt.plot(log_alphas, train_auc, label='Train AUC')
# this code is taken 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, cv_auc, label='CV AUC')
# this code is taken from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alphas,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color='darkorange')

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


plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("alpha: hyperparameter v/s AUC")
plt.grid()
plt.show()
```

```
100%|██████████| 19/19 [00:00
<?, ?it/s]
```



In [57]:
```
print('Best score: ',clf.best_score_)
print('k value: ',clf.best_params_)
print('='*10)
```

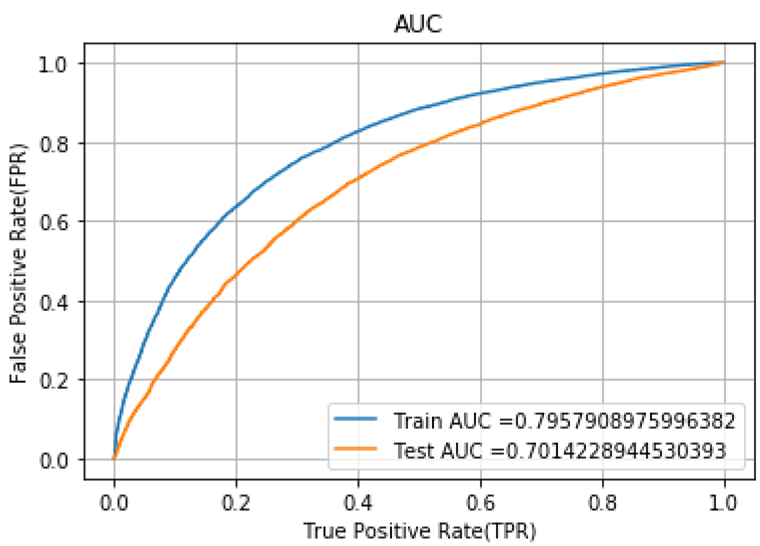
```
Best score:  0.7116797222248861
k value:  {'alpha': 0.5}
==========
```

In [58]:
```
best_k_1 = 0.5
```

In [59]:
```
print('Train AUC scores')
print(clf.cv_results_['mean_train_score'])
print('CV AUC scores')
print(clf.cv_results_['mean_test_score'])
```

```
Train AUC scores
[0.50006258 0.5        0.5        0.50005532 0.50828169 0.58991726
 0.71281137 0.74651529 0.7931575  0.80218895 0.81348439 0.81645642
 0.82142243 0.82302015 0.82587931 0.82682403 0.82852308 0.82908202
 0.83007256]
CV AUC scores
[0.50006259 0.5        0.5        0.50005529 0.50586215 0.57385746
 0.66639573 0.68711952 0.71005032 0.71167972 0.71012484 0.70859465
 0.70442195 0.70246053 0.69785198 0.69587373 0.69153133 0.68980147
 0.68618202]
```

In [60]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

nb_bow = MultinomialNB(alpha = best_k_1)

nb_bow.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 = batch_predict(nb_bow, X_tr)
y_test_pred = batch_predict(nb_bow, X_te)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

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



In [61]:
```
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(fpr*(1-tpr))]

    # (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
```

Confusion matrix for train data set

In [62]:
```
#confusion matrix for train data set
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

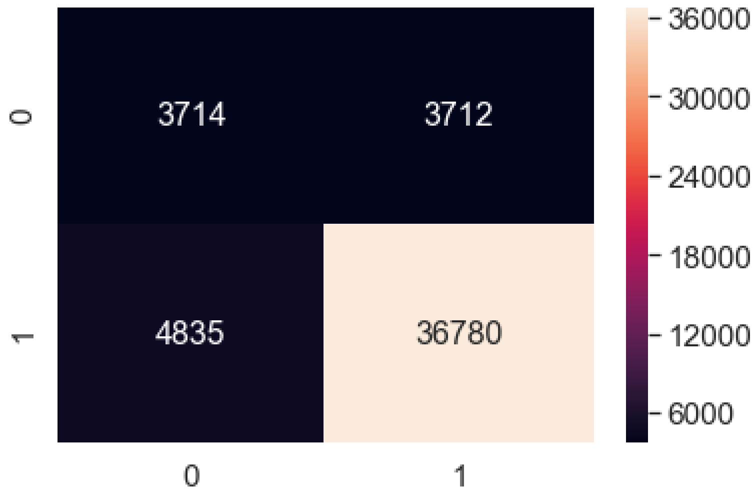
```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499999818661462 for threshold 0.133
[[ 3714  3712]
 [ 4835 36780]]
```

In [63]:
```
conf_matr_df_train_1 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, tr
                                    range(2),range(2))
```

```
the maximum value of tpr*(1-fpr) 0.2499999818661462 for threshold 0.133
```

In [64]:
```
#for the better vizulation of the confusion matrix
#this plot is build by using the seaborn
#https://seaborn.pydata.org/generated/seaborn.heatmap.html

sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_1, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[64]: <matplotlib.axes._subplots.AxesSubplot at 0x1cf014af4a8>



Confusion matrix for TEST data set

In [65]:
```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

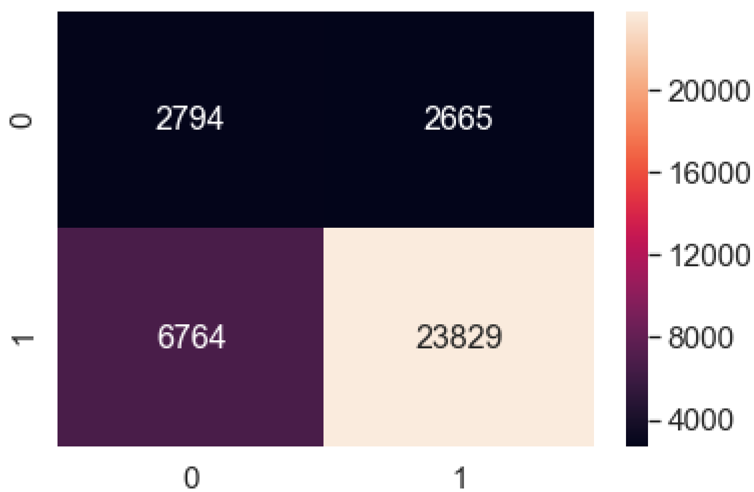
```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092995 for threshold 0.567
[[ 2794  2665]
 [ 6764 23829]]
```

In [66]:
```
conf_matr_df_test_1 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_f
```

```
the maximum value of tpr*(1-fpr) 0.24999999161092995 for threshold 0.567
```

In [67]:
```
#for the better vizulation of the confusion matrix
#this plot is build by using the seaborn
#https://seaborn.pydata.org/generated/seaborn.heatmap.html
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_1, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[67]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf0154c1d0>
```



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

In [68]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_tfidf, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe,
               X_train_ccat_ohe,X_train_csubcat_ohe, X_train_price_std,X_train_pre_project_std)).tocsr()

X_cr = hstack((X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe,
               X_cv_ccat_ohe,X_cv_csubcat_ohe, X_cv_price_std,X_cv_pre_project_std)).tocsr()

X_te = hstack((X_test_essay_tfidf, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe,
               X_test_ccat_ohe,X_test_csubcat_ohe,X_test_price_std,X_test_pre_project_std)).tocsr()

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(49041, 10356) (49041,)
(24155, 10356) (24155,)
(36052, 10356) (36052,)
====================================================================================================
```

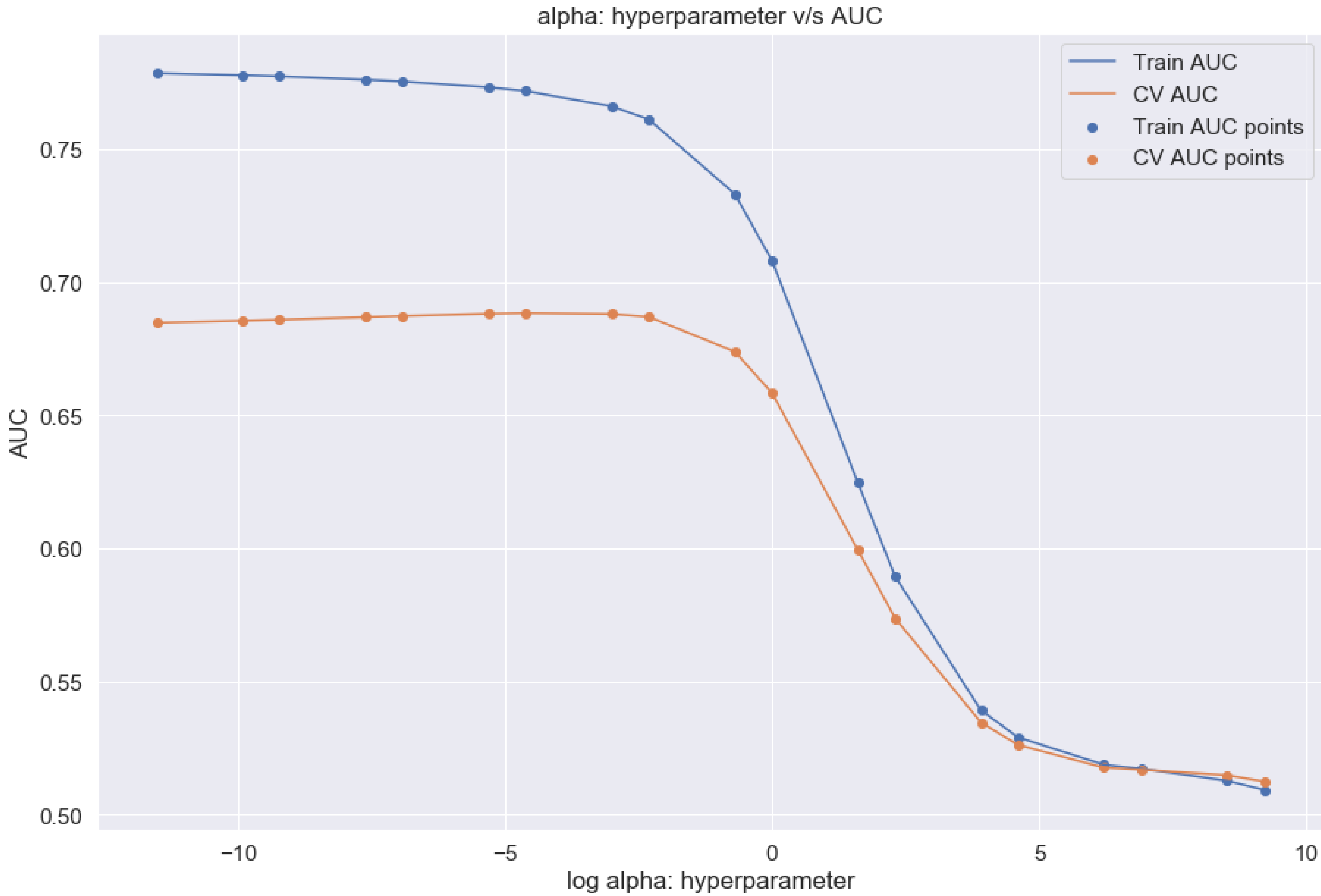
In [69]:
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [70]:

```
train_auc = []
cv_auc = []
log_alphas = []

alphas = [10000,5000,1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001,0.00005,0.00001]

for i in tqdm(alphas):
    nb = MultinomialNB(alpha = i,class_prior=[0.5,0.5])
    nb.fit(X_tr, y_train)

    y_train_pred = batch_predict(nb, X_tr)
    y_cv_pred = batch_predict(nb, X_cr)


    # 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))

for a in tqdm(alphas):
    b = math.log(a)
    log_alphas.append(b)
```

```
100%|████████████████████████████████████████| 19/19 [00:04<00:00,
4.92it/s]
100%|████████████████████████████████████████████| 19/19 [00:00
<?, ?it/s]
```

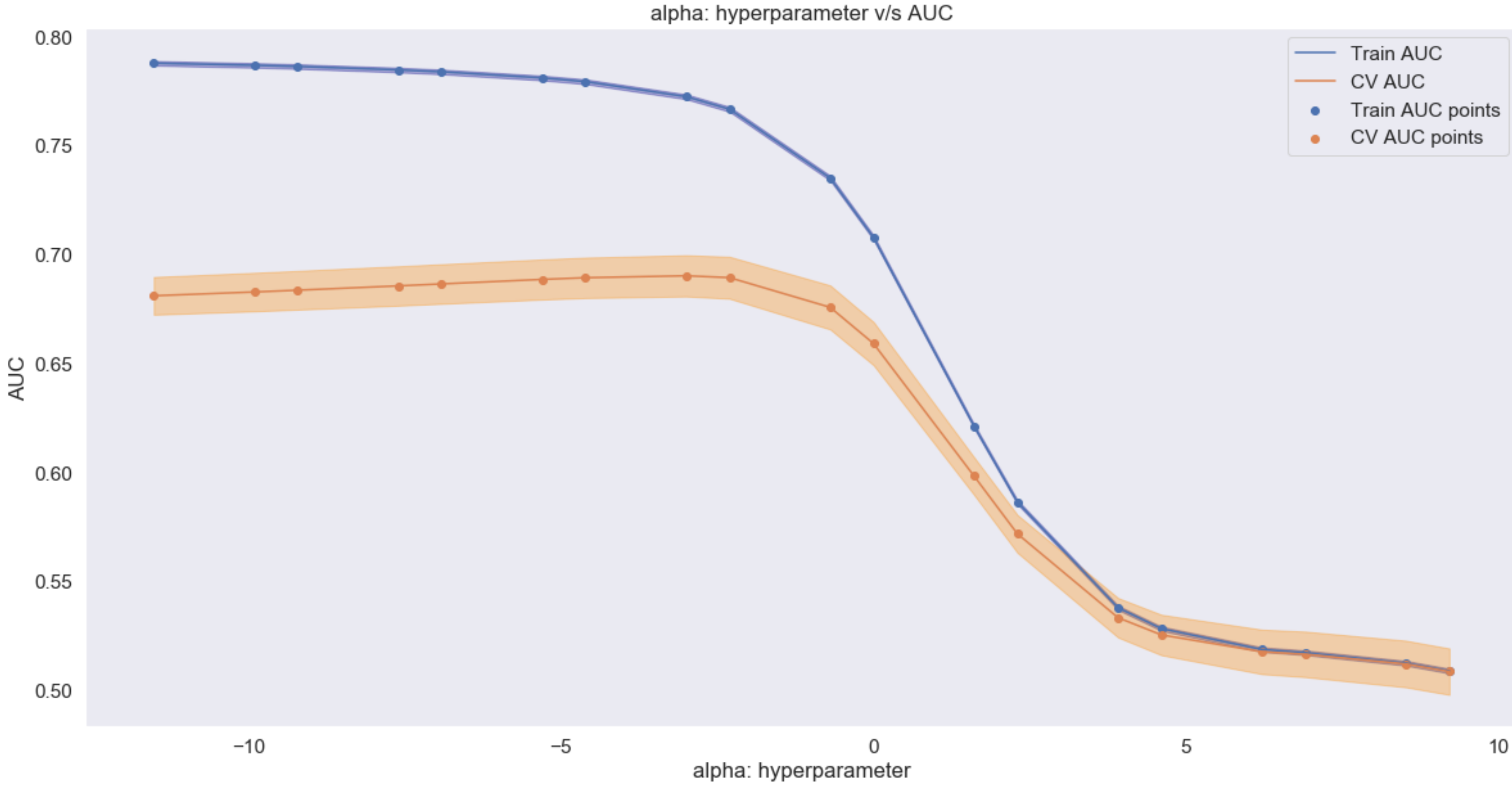
In [71]:
```
plt.figure(figsize=(15,10))
plt.plot(log_alphas, train_auc, label='Train AUC')
plt.plot(log_alphas, cv_auc, label='CV AUC')

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

plt.legend()
plt.xlabel("log alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("alpha: hyperparameter v/s AUC")
#plt.grid()
plt.show()
```


alpha: hyperparameter v/s AUC
Train AUC
CV AUC
Train AUC points
CV AUC points
AUC
0.75
0.70
0.65
0.60
0.55
0.50
−10
−5
0
5
10
log alpha: hyperparameter

In [72]:
```
from sklearn.model_selection import GridSearchCV

nb = MultinomialNB()

parameters = {'alpha':[10000,5000,1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001,0.00005,0.0

clf = GridSearchCV(nb, parameters, cv= 10, scoring='roc_auc',return_train_score=True)

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [73]:
```
alphas = [10000,5000,1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001,0.00005,0.00001]
log_alphas =[]

for a in tqdm(alphas):
    b = math.log(a)
    log_alphas.append(b)

plt.figure(figsize=(20,10))

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, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alphas,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color='darkorange')

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


plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("alpha: hyperparameter v/s AUC")
plt.grid()
plt.show()
```

```
100%|████████████████████████████████████████| 19/19 [00:00
<?, ?it/s]
```



In [74]:
```
print('Best score: ',clf.best_score_)
print('k value: ',clf.best_params_)
print('='*10)
```

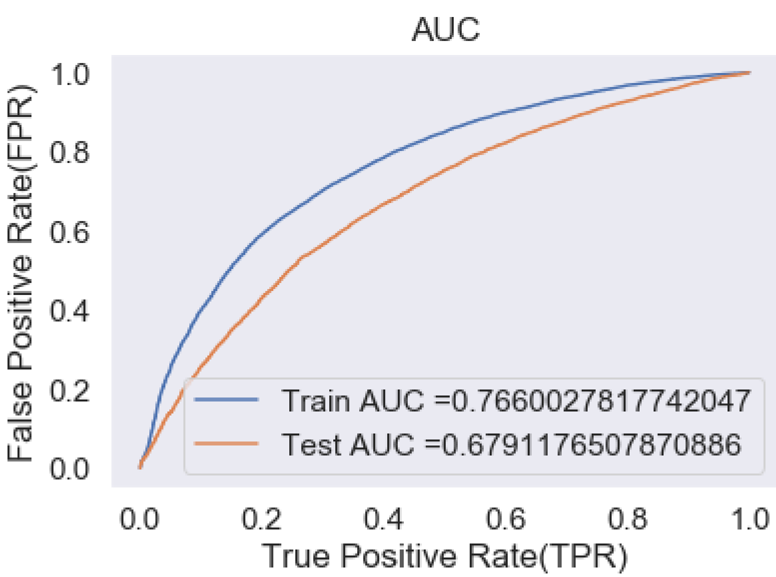
```
Best score:  0.690169164405337
k value:  {'alpha': 0.05}
==========
```

In [75]:
```
best_k_1 = 0.05
```

In [76]:
```
print('Train AUC scores')
print(clf.cv_results_['mean_train_score'])
print('CV AUC scores')
print(clf.cv_results_['mean_test_score'])
```

```
Train AUC scores
[0.50877954 0.51234951 0.51717129 0.51861261 0.52819717 0.5377839
 0.5862645  0.62111304 0.70776382 0.73504566 0.76663951 0.77237942
 0.77928396 0.78100398 0.78379268 0.78467766 0.78626595 0.78679571
 0.78776437]
CV AUC scores
[0.50850318 0.51199743 0.51643092 0.51755553 0.52531768 0.53321373
 0.57169871 0.5982098  0.65907504 0.67569235 0.68932114 0.69016916
 0.68927958 0.68850328 0.68643281 0.68552942 0.683546   0.68273514
 0.68099308]
```

In [77]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

nb_bow = MultinomialNB(alpha = best_k_1)

nb_bow.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 = batch_predict(nb_bow, X_tr)
y_test_pred = batch_predict(nb_bow, X_te)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

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



In [78]:
```
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(fpr*(1-tpr))]

    # (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
```

Confusion matrix for train data set

In [79]:
```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

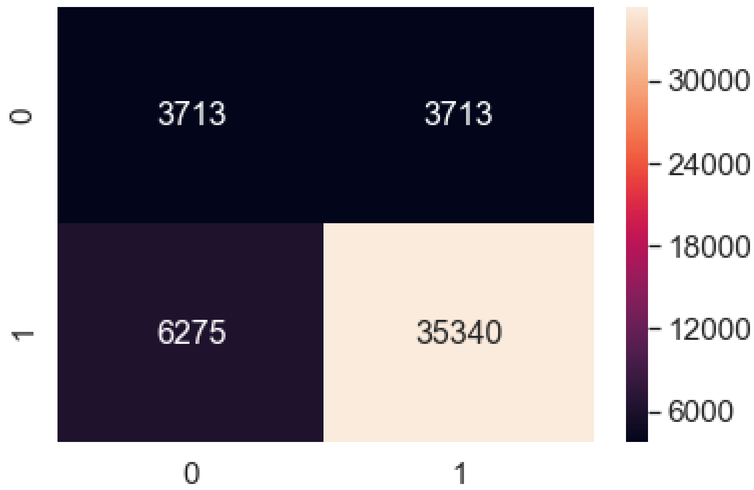
```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.779
[[ 3713  3713]
 [ 6275 35340]]
```

In [80]:
```
conf_matr_df_train_1 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, tr
                                    range(2),range(2))
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.779
```

In [81]:
```
#for the better vizulation of the confusion matrix
#this plot is build by using the seaborn
#https://seaborn.pydata.org/generated/seaborn.heatmap.html
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_1, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[81]: <matplotlib.axes._subplots.AxesSubplot at 0x1cf5dea1630>



Confusion matrix for TEST data set

In [82]:
```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

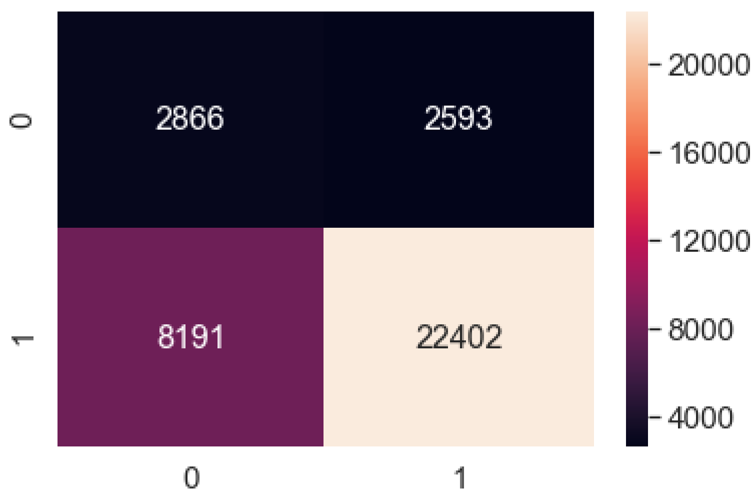
```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.827
[[ 2866  2593]
 [ 8191 22402]]
```

In [83]:
```
conf_matr_df_test_1 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_f
                                   range(2),range(2))
```

```
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.827
```

In [84]:
```
#for the better vizulation of the confusion matrix
#this plot is build by using the seaborn
#https://seaborn.pydata.org/generated/seaborn.heatmap.html
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_1, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[84]: <matplotlib.axes._subplots.AxesSubplot at 0x1cf5de0e438>



Top 20 features from BOW i.e SET 1 using absolute values of `feature_log_prob_` parameter of `MultinomialNB`

In [37]:
```
from scipy.sparse import hstack
X_tr = hstack(( X_train_essay_bow,X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe,
               X_train_ccat_ohe,X_train_csubcat_ohe)).tocsr()

X_cr = hstack(( X_cv_essay_bow,X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe,
               X_cv_ccat_ohe,X_cv_csubcat_ohe)).tocsr()

X_te = hstack(( X_test_essay_bow,X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe,
               X_test_ccat_ohe,X_test_csubcat_ohe)).tocsr()
```

In [38]:
```
print("Final Data-matrix:")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
```

```
Final Data-matrix:
(49041, 12263) (49041,)
(24155, 12263) (24155,)
(36052, 12263) (36052,)
```

In [39]:
```
NBModel = MultinomialNB(alpha=0.05, class_prior=[0.5,0.5])
NBModel.fit(X_tr, y_train)
```

Out[39]:
```
MultinomialNB(alpha=0.05, class_prior=[0.5, 0.5], fit_prior=True)
```

In [40]:
```
# For positive class
sorted_prob_class_1_ind = NBModel.feature_log_prob_[1, :].argsort()
# For negative class
sorted_prob_class_0_ind = NBModel.feature_log_prob_[0, :].argsort()
```

In [41]:
```
features_lst = list(vectorizer_ss.get_feature_names() +  vectorizer_bow.get_feature_names()+\
                    vectorizer_tp.get_feature_names() + vectorizer_pgc.get_feature_names()+\
                    vectorizer_cc.get_feature_names()+vectorizer_csc.get_feature_names() )
```

In [42]:
```
Most_imp_words_1 = []
Most_imp_words_0 = []

for index in sorted_prob_class_1_ind[-20:-1]:
    Most_imp_words_1.append(features_lst[index])

for index in sorted_prob_class_0_ind[-20:-1]:
    Most_imp_words_0.append(features_lst[index])
```

In [43]:
```

print("20 most imp features for positive class:\n")
print(Most_imp_words_1,)

print("\n" + "-"*100)

print("\n20 most imp features for negative class:\n")
print(Most_imp_words_0)
```

```
20 most imp features for positive class:

['70', 'cuties', 'log', 'untapped', 'random', 'wires', 'naked', 'wands', 'multiple', 'makeup', 'hdmi', 'latel
y', 'texts', 'nice', 'tender', 'church', 'latino', 'movin', 'saves']

----------------------------------------------------------------------------------------------------

20 most imp features for negative class:

['simplicity', 'random', '70', 'log', 'colds', 'wires', 'naked', 'wands', 'makeup', 'multiple', 'tender', 'hdm
i', 'texts', 'lately', 'nice', 'church', 'movin', 'latino', 'saves']
```

In [44]:
```
Most_imp_words=Most_imp_words_0+Most_imp_words_1
#comibiningb the +ve and -ve class
```

In [45]:
```
np.sort(Most_imp_words)# srtoring the new lsit

print(Most_imp_words)#printing the new list
```

```
['simplicity', 'random', '70', 'log', 'colds', 'wires', 'naked', 'wands', 'makeup', 'multiple', 'tender', 'hdm
i', 'texts', 'lately', 'nice', 'church', 'movin', 'latino', 'saves', '70', 'cuties', 'log', 'untapped', 'rando
m', 'wires', 'naked', 'wands', 'multiple', 'makeup', 'hdmi', 'lately', 'texts', 'nice', 'tender', 'church', 'l
atino', 'movin', 'saves']
```

In [221]:
```
#refrence of the code
```

In [ ]:
```
#https://stackoverflow.com/questions/30522724/take-multiple-lists-into-dataframe
#https://stats.stackexchange.com/questions/266031/what-is-log-probability-of-feature-in-sklearn-multinomialnb
#https://stackoverflow.com/questions/7271385/how-do-i-combine-two-lists-into-a-dictionary-in-python
#https://stackoverflow.com/questions/50526898/how-to-get-feature-importance-in-naive-bayes
#https://stackoverflow.com/questions/16486252/is-it-possible-to-use-argsort-in-descending-order
#https://datascience.stackexchange.com/questions/65219/find-the-top-n-features-from-feature-set-using-absolute-v
```

3. Summary

as mentioned in the step 5 of instructions

In [85]:
```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable

#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Alpha:Hyper Parameter", "AUC"]

x.add_row(["BOW", "Naive Bayes", 0.5, 0.7116])
x.add_row(['TFIDF','Navie Bayes',0.05, 0.6901])
print(x)
```

```
+------------+-------------+-----------------------+--------+
| Vectorizer |    Model    | Alpha:Hyper Parameter |  AUC   |
+------------+-------------+-----------------------+--------+
|    BOW     | Naive Bayes |          0.5          | 0.7116 |
|   TFIDF    | Navie Bayes |          0.05         | 0.6901 |
+------------+-------------+-----------------------+--------+
```

References

In [ ]:
```
#https://stackoverflow.com/questions/56416576/getting-keyerror-from-sklearn-model-selection-gridsearchcv
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html#sklearn.metrics.roc_auc_sc
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.average_precision_score.html#sklearn.metrics.
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
#https://stackoverflow.com/questions/19710602/concatenate-sparse-matrices-in-python-using-scipy-numpy/19710648#1
#https://www.appliedaicourse.com/lecture/11/Applied-Machine-Learning-course/2971/handling-categorical-and-numeri
#https://stackoverflow.com/questions/50526898/how-to-get-feature-importance-in-naive-bayes
#https://stackoverflow.com/a/19710648/4084039
#https://stackoverflow.com/a/48803361/4084039
#https://seaborn.pydata.org/generated/seaborn.heatmap.html
#http://zetcode.com/python/prettytable/
#https://github.com/vishalgupta1996/Naive-bayes-donor-choose/blob/master/NaiveBaiyes.ipynb
#https://classroom.appliedcourse.com/classrooms/JQOZ47Em/assignments/JgO5JAP8/
```