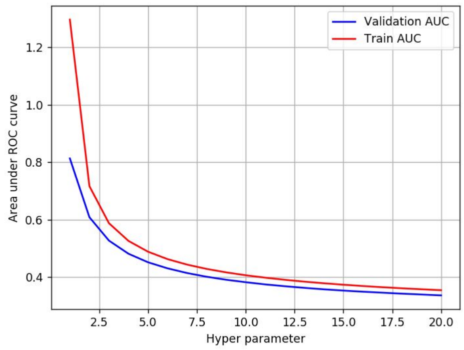
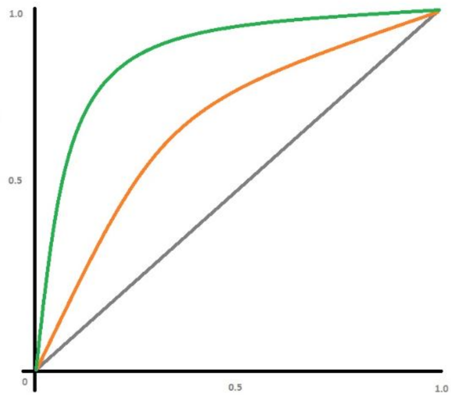
Assignment 6: Apply NB

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 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 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

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

```
| TFIDFW2V    | Brute      |        6         |  0.78    |
+-------------+------------+------------------+----------+
```

2. Naive Bayes

1.1 Loading Data

In [18]:

```
import pandas
data = pandas.read_csv('preprocessed_data.csv', nrows = 50000)
data.shape
data.head()
```

Out[18]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_cate
0	ca	mrs	grades_prek_2	53	1	math_s
1	ut	ms	grades_3_5	4	1	special
2	ca	mrs	grades_prek_2	10	1	literacy_lan
3	ga	mrs	grades_prek_2	2	1	appliedle
4	wa	mrs	grades_3_5	2	1	literacy_lan

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

In [12]:

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

In [13]:

```
from sklearn.model_selection import train_test_split

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
```

```
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)

print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
```

```
(22445, 8) (22445,)
(11055, 8) (11055,)
(16500, 8) (16500,)
```

1.3 Make Data Model Ready: encoding eassay, and project_title

In [0]:

```
# 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
```

In [15]:

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(X_train['essay'].values)
```

Out[15]:

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

In [16]:

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

1.4 Make Data Model Ready: encoding numerical, categorical features

In [0]:

```
# 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
```

In [54]:

```
# Encoding School State - OHE
# School State
vectorizer = CountVectorizer()
vectorizer.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.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)
school_state_feature_names = vectorizer.get_feature_names()
```

In [55]:

```
# Encoding Teacher Prefix OHE
# teacher_prefix
vectorizer = CountVectorizer()
vectorizer.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.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)
teacher_prefix_feature_names = vectorizer.get_feature_names()
```

In [56]:

```
# Encoding project_grade_category
vectorizer = CountVectorizer()
vectorizer.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.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)
grade_feature_names = vectorizer.get_feature_names()
```

In [57]:

```
# clean_categories
vectorizer = CountVectorizer()
vectorizer.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_category_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_cv_category_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_category_ohe = vectorizer.transform(X_test['clean_categories'].values)
category_feature_names = vectorizer.get_feature_names()
```

In [58]:

```
# Encoding sub categories
vectorizer = CountVectorizer()
vectorizer.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_subcategory_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_subcategory_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_subcategory_ohe = vectorizer.transform(X_test['clean_subcategories'].values)
subcategory_feature_names = vectorizer.get_feature_names()
```

In [25]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

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

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(1,-1))
X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1))
```

In [26]:

```
X_train_price_norm = X_train_price_norm.reshape(-1,1)
X_cv_price_norm = X_cv_price_norm.reshape(-1,1)
X_test_price_norm = X_test_price_norm.reshape(-1,1)
```

In [27]:

```
# teacher previously posted projects
normalizer = Normalizer()
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))

X_train_teach_prev_norm =
normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
X_cv_teach_prev_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].v
alues.reshape(1,-1))
X_test_teach_prev_norm =
normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
```

In [28]:

```
# reshaping the ndarrays post normalization
X_train_teach_prev_norm = X_train_teach_prev_norm.reshape(-1,1)
X_cv_teach_prev_norm = X_cv_teach_prev_norm.reshape(-1,1)
X_test_teach_prev_norm = X_test_teach_prev_norm.reshape(-1,1)
```

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 [0]:

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

In [33]:

```
# concatinating all the features
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_bow, X_train_state_ohe, X_train_teacher_ohe,
               X_train_grade_ohe, X_train_price_norm, X_train_category_ohe,
               X_train_subcategory_ohe, X_train_teach_prev_norm)).tocsr()

X_cr = hstack((X_cv_essay_bow, X_cv_state_ohe, X_cv_teacher_ohe,
               X_cv_grade_ohe, X_cv_category_ohe, X_cv_subcategory_ohe,
               X_cv_price_norm, X_cv_teach_prev_norm)).tocsr()

X_te = hstack((X_test_essay_bow, X_test_state_ohe, X_test_teacher_ohe,
               X_test_grade_ohe, X_test_category_ohe, X_test_subcategory_ohe,
               X_test_price_norm, X_test_teach_prev_norm)).tocsr()
```

In [34]:

```
# function to perform batch predict
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive 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 tr_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
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

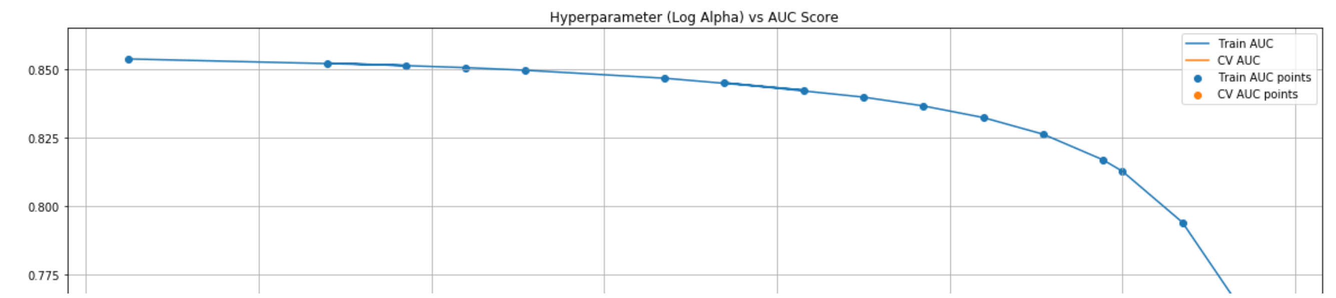
In [40]:

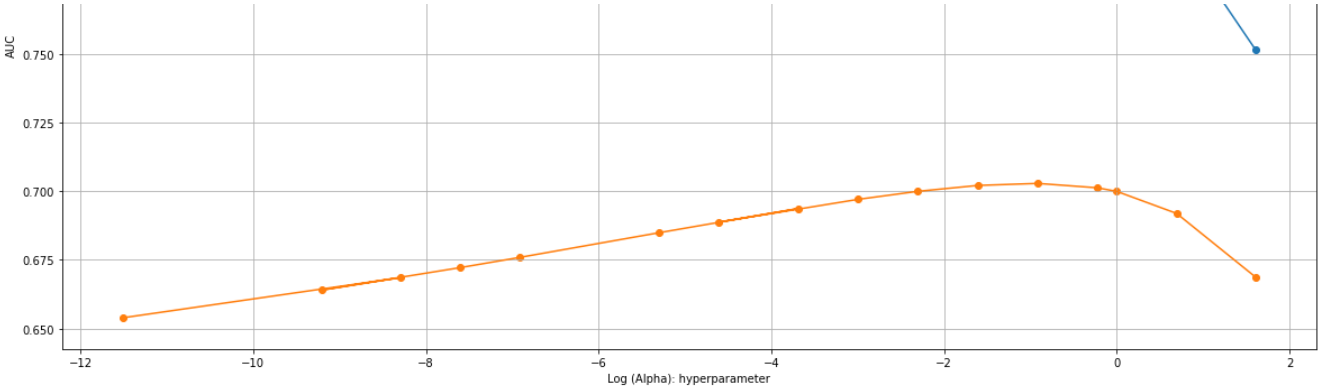
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
from tqdm import tqdm

# Performing Naive Bayes on a wide range of alphas
train_auc = []
cv_auc = []
alpha = [0.00001,0.00025, 0.0001, 0.0005, 0.001, 0.005, 0.025, 0.01, 0.05, 0.1, 0.2, 0.4, 0.8, 1,2,5
]
for i in tqdm(alpha):
    nb_output = MultinomialNB(alpha=i,class_prior=[0.5,0.5]) # class_prior is used since there is a
n imbalance in the dataset
    nb_output.fit(X_tr, y_train)

    y_train_pred = nb_output.predict_proba(X_tr)[:,1] # Returning the probablity score of greater c
lass label
    y_cv_pred = nb_output.predict_proba(X_cr)[:,1]

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive 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))
```

```
100%|██████████████████████████████████████████| 16/16
[00:01<00:00, 14.36it/s]
```

In [43]:

```
from math import log
log_alphas = [log(alph) for alph in alpha]

plt.figure(figsize=(20,10))
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("Hyperparameter (Log Alpha) vs AUC Score")
plt.grid()
plt.show()
```





In [45]:

```
# Lets use GridSearchCV to find the best hyperparameter
from sklearn.model_selection import GridSearchCV
import numpy as np
mnb_output =MultinomialNB(class_prior=[0.5,0.5])
parameters = {"alpha":np.arange(0.00001,50,0.5)}
clf = GridSearchCV(mnb_output, parameters, cv= 5, scoring='roc_auc',return_train_score=True)
clf.fit(X_tr, y_train)
```

Out[45]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=MultinomialNB(alpha=1.0, class_prior=[0.5, 0.5],
                                     fit_prior=True),
             iid='warn', n_jobs=None,
             param_grid={'alpha': array([1.000000e-05, 5.000100e-01, 1.000010e+00, 1.500010e+00,
       2.000010e+00, 2.500010e+00, 3.000010e+00, 3.500010e+00,
       4.000010e+00, 4.500010e+00, 5.000010e+00, 5.500010e+00,
       6.000010e+00, 6.500010e...
       4.000001e+01, 4.050001e+01, 4.100001e+01, 4.150001e+01,
       4.200001e+01, 4.250001e+01, 4.300001e+01, 4.350001e+01,
       4.400001e+01, 4.450001e+01, 4.500001e+01, 4.550001e+01,
       4.600001e+01, 4.650001e+01, 4.700001e+01, 4.750001e+01,
       4.800001e+01, 4.850001e+01, 4.900001e+01, 4.950001e+01])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [46]:

```
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']

#Output of GridSearchCV
print('Best score: ',clf.best_score_)
print('k value with best score: ',clf.best_params_)
print('='*75)
print('Train AUC scores')
print(clf.cv_results_['mean_train_score'])
print('CV AUC scores')
print(clf.cv_results_['mean_test_score'])
```

```
Best score:  0.6959442059820922
k value with best score:  {'alpha': 0.50001}
===========================================================================
Train AUC scores
[0.88186664 0.84226023 0.82830629 0.81622912 0.80520565 0.79506267
 0.78570799 0.77707971 0.76914189 0.76181788 0.7550484  0.74878709
 0.74297075 0.73755967 0.73250562 0.72779004 0.72336173 0.7192215
 0.7153138  0.71163958 0.70814902 0.70484096 0.70171135 0.69873254
 0.69587125 0.69315264 0.69056902 0.68808999 0.68570939 0.6834156
 0.68121017 0.67910564 0.67710176 0.67511588 0.67322194 0.67136743
 0.66959874 0.66785475 0.66609536 0.66438007 0.66258067 0.66086817
 0.65917875 0.65741685 0.65567372 0.65378603 0.65198799 0.65010705
 0.64831015 0.64653404 0.64467837 0.64241919 0.6409548  0.63909745
 0.63709036 0.6350978  0.63303198 0.63095473 0.62836458 0.62629549
 0.62364597 0.62104072 0.61853443 0.61596537 0.61335643 0.61139381
```

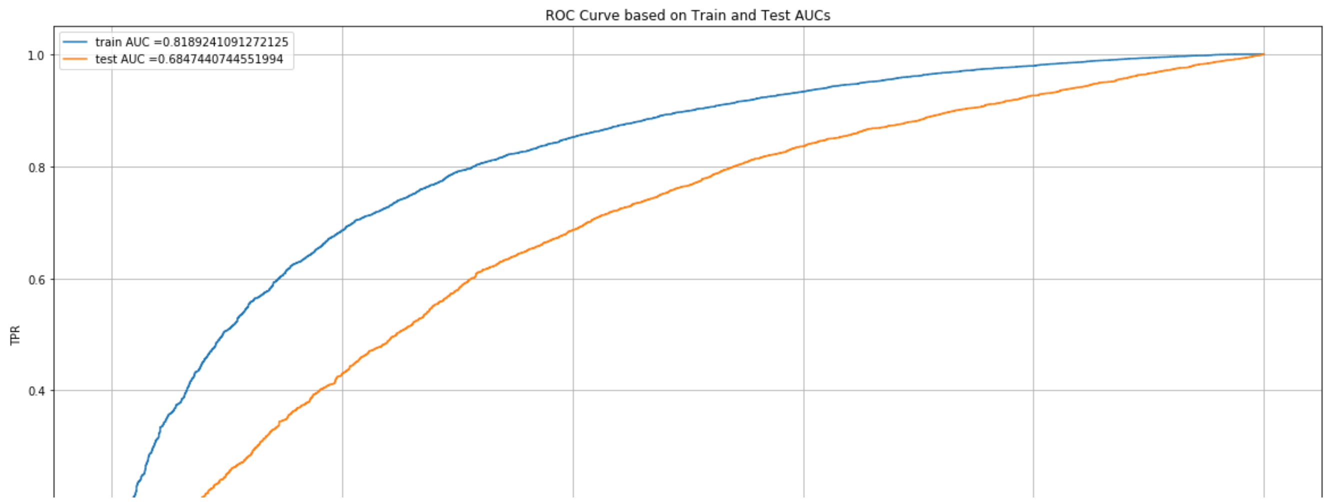
```
 0.609021   0.60653515 0.60460777 0.60255504 0.59959688 0.59722416
 0.59463828 0.59180512 0.58868122 0.585953   0.58273088 0.5810485
 0.57867332 0.57608584 0.57359221 0.5709614  0.56911484 0.56666309
 0.56374677 0.5611619  0.55868226 0.55720163 0.55474217 0.55267987
 0.55180052 0.5496434  0.54831653 0.54698286 0.54577981 0.54425121
 0.54330509 0.54199342 0.54090718 0.53984235]
CV AUC scores
[0.65496906 0.69594421 0.69335207 0.68918029 0.68468683 0.68005988
 0.67565271 0.67144884 0.66750222 0.66388621 0.66052353 0.65742983
 0.65458508 0.65189826 0.64944663 0.64713159 0.64497261 0.64297424
 0.64111425 0.63930163 0.63757607 0.63597796 0.63445719 0.6329902
 0.63162085 0.63033431 0.6290665  0.62783214 0.62669714 0.62559757
 0.62454381 0.62350076 0.62251872 0.62153038 0.62063944 0.61974006
 0.61876317 0.61780372 0.61684113 0.61609045 0.61493924 0.61419862
 0.61299322 0.61198581 0.61134178 0.61024457 0.60903699 0.60802293
 0.60681018 0.6056944  0.60423934 0.60335169 0.60207354 0.60068111
 0.59966708 0.59889657 0.59674618 0.59499043 0.59293106 0.59161934
 0.58904611 0.58733931 0.58588502 0.58418412 0.58287003 0.5799299
 0.57790286 0.57632478 0.57551648 0.57356591 0.57133068 0.56892452
 0.56640484 0.56422806 0.56148173 0.55921054 0.55753326 0.55611634
 0.5549544  0.5535733  0.55073598 0.54869484 0.5455632  0.54436651
 0.54206499 0.5409026  0.53920372 0.53651689 0.53513464 0.53447351
 0.53332642 0.53229114 0.53176299 0.53058512 0.52945397 0.5293568
 0.52791976 0.52654906 0.5259539  0.52608227]
```

In [47]:

```python
best_alpha = 0.70
from sklearn.metrics import roc_curve, auc


nb_output = MultinomialNB(alpha = best_alpha)
nb_output.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 = nb_output.predict_proba(X_tr)[:,1]  # returning probability estimates of positive class
y_test_pred = nb_output.predict_proba(X_te)[:,1]

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

In [48]:

```python
plt.figure(figsize=(20,10))
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("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve based on Train and Test AUCs")
plt.grid()
plt.show()
```



In [49]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [50]:

```
# Drawing the confusion matrix as a Seaborn Heatmap
import seaborn as sns
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
Train_CM = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
Test_CM = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
print("Train confusion matrix")
print(Train_CM)
print("Test confusion matrix")
print(Test_CM)
```

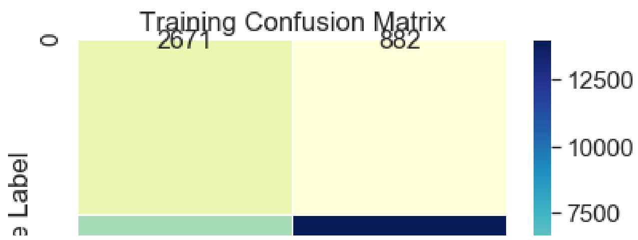
```
====================================================================================================

the maximum value of tpr*(1-fpr) 0.555542476800447 for threshold 0.822
Train confusion matrix
[[ 2671   882]
 [ 4931 13961]]
Test confusion matrix
[[1614 1028]
 [4506 9352]]
```

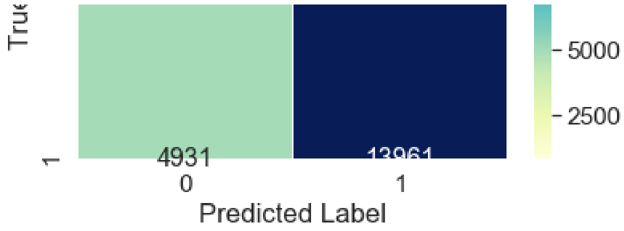
In [51]:

```
sns.set(font_scale=1.4)
sns.heatmap(Train_CM,annot=True,cbar=True,fmt="g", annot_kws = {"size":16},linewidths=.5,cmap="YlGn
Bu")
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Training Confusion Matrix')
```

Out[51]:

```
Text(0.5, 1, 'Training Confusion Matrix')
```



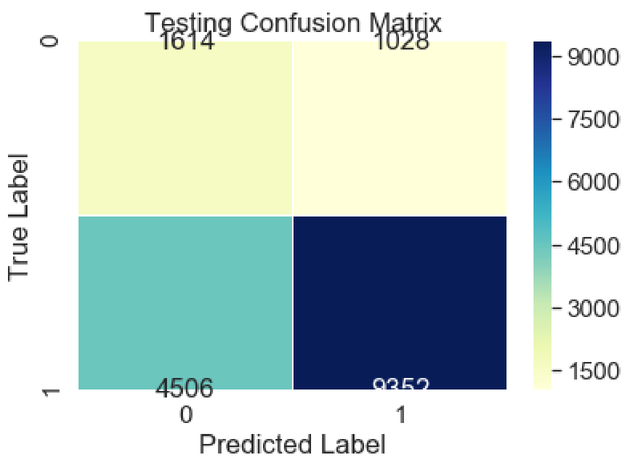


In [52]:

```
sns.heatmap(Test_CM,annot=True,cbar=True,fmt="d", linewidths=.5,cmap="YlGnBu")
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Testing Confusion Matrix')
```

Out[52]:

```
Text(0.5, 1, 'Testing Confusion Matrix')
```



In [60]:

```
# concatenating all feature names which is used later to find the best 10 features
bow_feature_names_list = []
bow_feature_names_list.extend(bow_essay_feature_names)
bow_feature_names_list.extend(school_state_feature_names)
bow_feature_names_list.extend(teacher_prefix_feature_names)
bow_feature_names_list.extend(grade_feature_names)
bow_feature_names_list.extend("Price")
bow_feature_names_list.extend(category_feature_names)
bow_feature_names_list.extend(subcategory_feature_names)
bow_feature_names_list.extend("Teacher Previously submitted projects")
print (len(bow_feature_names_list))
```

```
8947
```

In [61]:

```
# the attribute feature_log_prob_ contains the log probabilities of each feature.
# From X_test.shape, you can see that there were 10101 features
nb_output.feature_log_prob_
```

Out[61]:

```
array([[-10.33953053,  -9.57978856,  -8.46634745, ...,  -7.83829093,
        -12.624311  , -11.31010957],
       [-10.97909802,  -9.71064957,  -8.45586746, ...,  -8.05888316,
        -15.2417779 , -10.9626785 ]])
```

In [62]:

```
# Length of (nb_output.feature_log_prob_[0] == nb_output.feature_log_prob_[1] ==  No.of features =
= 10101)
```

```
print (len(nb_output.feature_log_prob_[0]))
len(nb_output.feature_log_prob_[1])
```

```
8907
```

Out[62]:

```
8907
```

In [63]:

```
nb_output.classes_
```

Out[63]:

```
array([0, 1], dtype=int64)
```

In [64]:

```
# Using Numpy we can sort these arrays and retrieve the indices which result in the highest log pr
obability
# code snippet from https://stackoverflow.com/questions/6910641/how-do-i-get-indices-of-n-maximum-
values-in-a-numpy-array
# using numpy argpartitition to retrieve top 10 features
class_zero_top_10_features = np.argpartition(nb_output.feature_log_prob_[0], -10)[-10:]
print (class_zero_top_10_features) # top 10 args
print (nb_output.feature_log_prob_[0][class_zero_top_10_features]) # respective values
```

```
[7910 3781 7936 4587 5356 1479 4591 5222 6927 7620]
[-4.88551535 -4.87329119 -4.84706067 -4.82583622 -4.79003514 -4.58331469
 -4.44667952 -4.52037199 -4.14211123 -3.04978922]
```

In [65]:

```
class_one_top_10_features = np.argpartition(nb_output.feature_log_prob_[1], -10)[-10:]
print (class_one_top_10_features) # top 10 args
print (nb_output.feature_log_prob_[1][class_one_top_10_features]) # respective values
```

```
[3781 4587 7936 5356 7910 5222 6927 1479 7620 4591]
[-4.90332654 -4.86122687 -4.85231231 -4.81895673 -4.78077063 -4.50843869
 -4.18063003 -4.54227396 -3.04366758 -4.54291863]
```

In [66]:

```
print('Top 10 features from negative class:')
print(np.take(bow_feature_names_list, class_zero_top_10_features))
print('-*'*50)
print('Top 10 features from positive class:')
print(np.take(bow_feature_names_list, class_one_top_10_features))
```

```
Top 10 features from negative class:
['the' 'help' 'they' 'learn' 'not' 'classroom' 'learning' 'my' 'school'
 'students']
-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
*-*
Top 10 features from positive class:
['help' 'learn' 'they' 'not' 'the' 'my' 'school' 'classroom' 'students'
 'learning']
```

Applying Naive Bayes on TFIDF

In [67]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train["essay"].values)
X_train_essay_tfidf = vectorizer.transform(X_train['essay'].values)
```

```
X_cv_essay_tfidf = vectorizer.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)

print("Shape of Datamatrix after TFIDF Vectorization")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
tfidf_essay_feature_names = vectorizer.get_feature_names()
```

```
Shape of Datamatrix after TFIDF Vectorization
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
====================================================================================================
```

In [70]:

```
X_tr = hstack((X_train_essay_tfidf, X_train_state_ohe, X_train_teacher_ohe,
               X_train_grade_ohe, X_train_price_norm, X_train_category_ohe,
               X_train_subcategory_ohe, X_train_teach_prev_norm)).tocsr()

X_cr = hstack((X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_ohe,
               X_cv_grade_ohe, X_cv_category_ohe, X_cv_subcategory_ohe,
               X_cv_price_norm, X_cv_teach_prev_norm)).tocsr()

X_te = hstack((X_test_essay_tfidf, X_test_state_ohe, X_test_teacher_ohe,
               X_test_grade_ohe, X_test_category_ohe, X_test_subcategory_ohe,
               X_test_price_norm, X_test_teach_prev_norm)).tocsr()

tfidf_feature_names_list = []
tfidf_feature_names_list.extend(tfidf_essay_feature_names)
tfidf_feature_names_list.extend(school_state_feature_names)
tfidf_feature_names_list.extend(teacher_prefix_feature_names)
tfidf_feature_names_list.extend(grade_feature_names)
tfidf_feature_names_list.extend("Price")
tfidf_feature_names_list.extend(category_feature_names)
tfidf_feature_names_list.extend(subcategory_feature_names)
tfidf_feature_names_list.extend("Teacher Previously submitted projects")
print (len(tfidf_feature_names_list))
```

```
5141
```

In [71]:

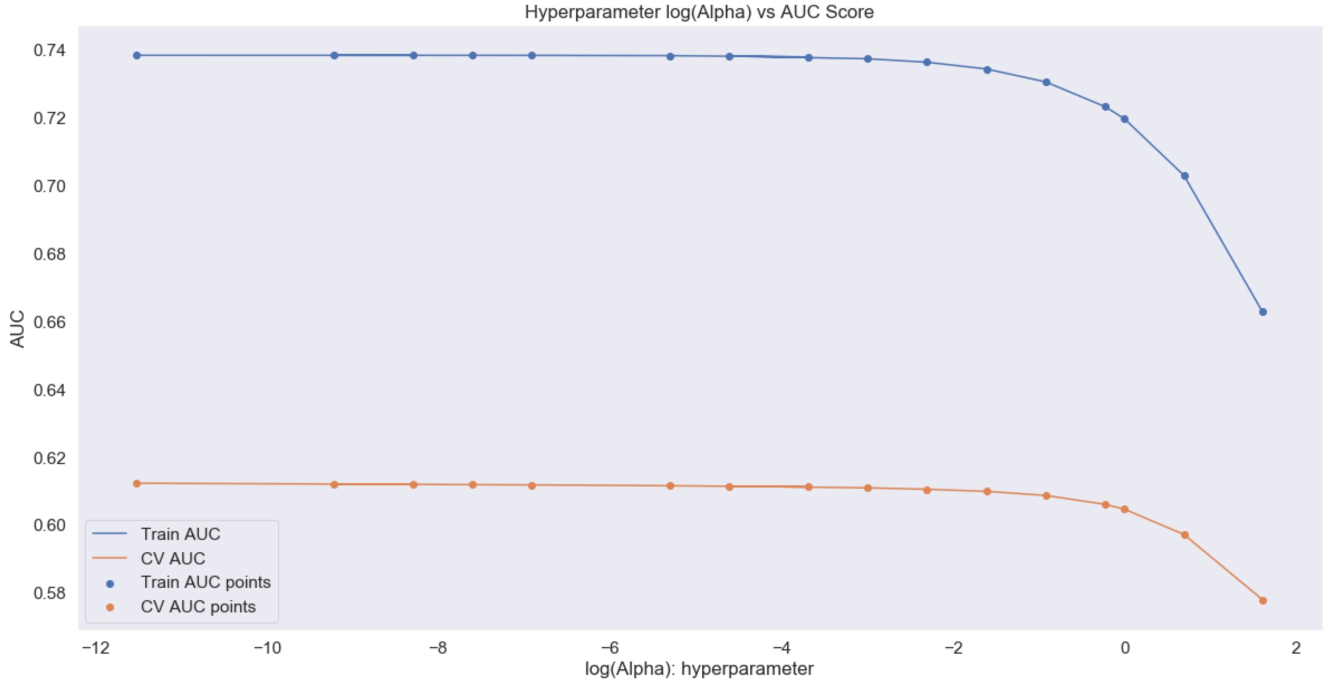
```
train_auc = []
cv_auc = []
alpha = [0.00001,0.00025, 0.0001, 0.0005, 0.001, 0.005, 0.025, 0.01, 0.05, 0.1, 0.2, 0.4, 0.8, 1, 2,
5]
for i in tqdm(alpha):
    nb_output = MultinomialNB(alpha=i,class_prior=[0.5,0.5]) # class_prior is used since there is a
n imbalance in the dataset
    nb_output.fit(X_tr, y_train)

    y_train_pred = nb_output.predict_proba(X_tr)[:,1] # Returning the probablity score of greater c
lass label
    y_cv_pred = nb_output.predict_proba(X_cr)[:,1]

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive 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))
```

```
100%|████████████████████████████████████████████████████████████████████████████████| 16/16
[00:00<00:00, 16.39it/s]
```

In [72]:

```
log_alphas = [log(alph) for alph in alpha]
plt.figure(figsize=(20,10))
```

```
plt.figure(figsize=(20,10))
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("Hyperparameter log(Alpha) vs AUC Score")
plt.grid()
plt.show()
```

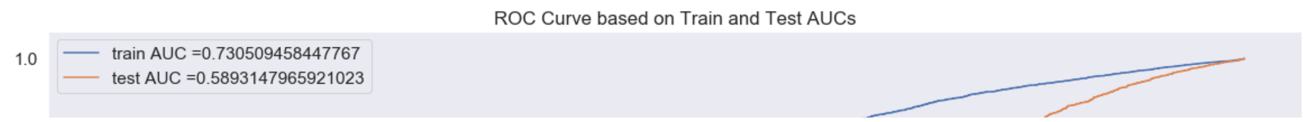


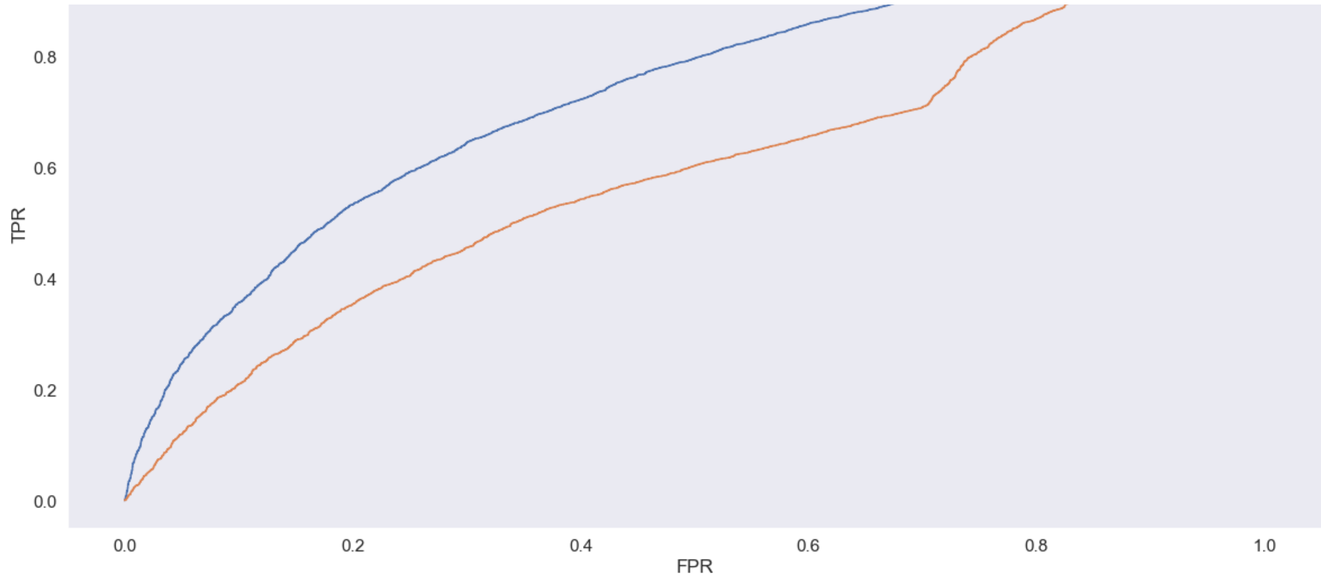
In [73]:

```
best_alpha = 0.4 # from graph it looks like 0.4 is best alpha
from sklearn.metrics import roc_curve, auc


nb_output = MultinomialNB(alpha = best_alpha)
nb_output.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_output, X_tr)
y_test_pred = batch_predict(nb_output, 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)
```

In [74]:

```
plt.figure(figsize=(20,10))
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("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve based on Train and Test AUCs")
plt.grid()
plt.show()
```





In [75]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
Train_CM = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
Test_CM = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
print("Train confusion matrix")
print(Train_CM)
print("Test confusion matrix")
print(Test_CM)
```

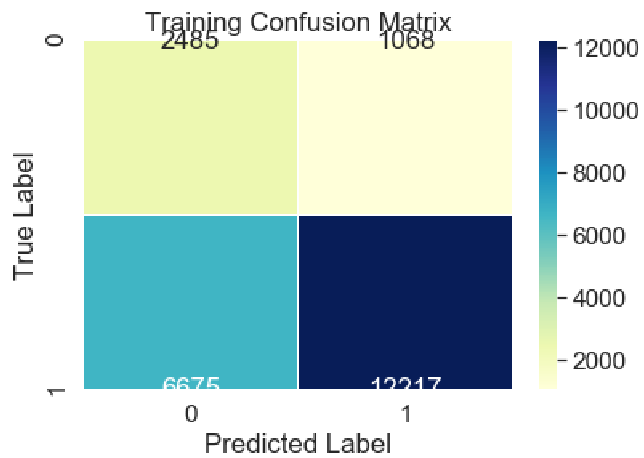
```
====================================================================================================

the maximum value of tpr*(1-fpr) 0.4522908715003719 for threshold 0.848
Train confusion matrix
[[ 2485  1068]
 [ 6675 12217]]
Test confusion matrix
[[1651  991]
 [6538 7320]]
```

In [76]:

```
sns.heatmap(Train_CM,annot=True,cbar=True,fmt="d", linewidths=.5,cmap="YlGnBu")
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Training Confusion Matrix')
```

Out[76]:

```
Text(0.5, 1, 'Training Confusion Matrix')
```

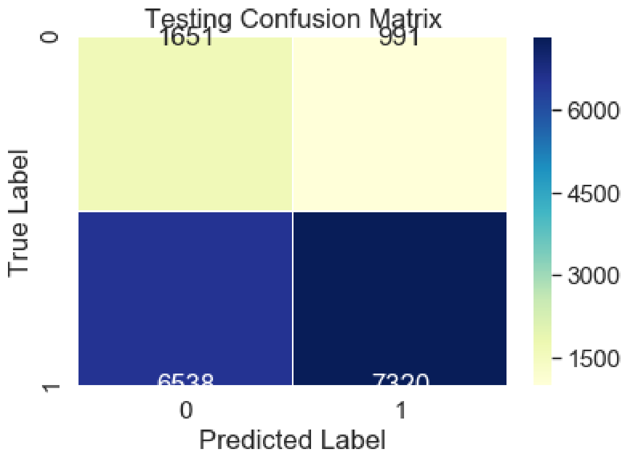


In [77]:

```
sns.heatmap(Test_CM,annot=True,cbar=True,fmt="d", linewidths=.5,cmap="YlGnBu")
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Testing Confusion Matrix')
```

Out[77]:

```
Text(0.5, 1, 'Testing Confusion Matrix')
```



In [78]:

```
# the attribute feature_log_prob_ contains the log probabilities of each feature.
# From X_test.shape, you can see that there were 10101 features
nb_output.feature_log_prob_
```

Out[78]:

```
array([[ -9.65427863,  -8.83776272,  -8.27202172, ...,  -5.67030249,
        -10.64900474,  -9.18922394],
       [ -9.80963433,  -8.808619  ,  -8.35064992, ...,  -5.81566757,
        -13.55785255,  -8.72509298]])
```

In [79]:

```
# Please write all the code with proper documentation
class_one_top_10_features = np.argpartition(nb_output.feature_log_prob_[1], -10)[-10:]
print (class_one_top_10_features) # top 10 args
print (nb_output.feature_log_prob_[1][class_one_top_10_features]) # respective values
```

```
[5063 5054 5053 5056 5089 5088 5059 5065 5066 5087]
[-4.49184455 -3.84652479 -3.41196799 -3.85565533 -4.20679417 -4.35923057
 -3.6891635  -3.52197801 -3.82524229 -3.95777003]
```

In [80]:

```
# Please write all the code with proper documentation
class_zero_top_10_features = np.argpartition(nb_output.feature_log_prob_[0], -10)[-10:]
print (class_zero_top_10_features) # top 10 args
print (nb_output.feature_log_prob_[0][class_zero_top_10_features]) # respective values
```

```
[5063 5089 5088 5087 5066 5053 5065 5054 5056 5059]
[-4.59624353 -4.17159291 -4.41263515 -4.14878814 -3.75679829 -3.47866544
 -3.66270237 -3.81121219 -3.94196777 -3.68879256]
```

In [81]:

```
print('Top 10 features from negative class:')
print(np.take(bow_feature_names_list, class_zero_top_10_features))
print('-*'*50)
print('Top 10 features from positive class:')
```

```
print("Top 10 features from positive class: ")
print(np.take(bow_feature_names_list, class_one_top_10_features))
```

```
Top 10 features from negative class:
['mini' 'mississippi' 'missions' 'mission' 'minimize' 'mindful' 'minimal'
 'mindfulness' 'mindset' 'mine']
-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
*-*
Top 10 features from positive class:
['mini' 'mindfulness' 'mindful' 'mindset' 'mississippi' 'missions' 'mine'
 'minimal' 'minimize' 'mission']
```

3. Summary

as mentioned in the step 5 of instructions

In [82]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyperparameter-Alpha", "Train AUC", "Test AUC"]

x.add_row(["BOW","Brute",1,0.79,0.70])
x.add_row(["TFIDF","Brute",0.4,0.74,0.65])
print(x)
```

```
+------------+-------+----------------------+-----------+----------+
| Vectorizer | Model | Hyperparameter-Alpha | Train AUC | Test AUC |
+------------+-------+----------------------+-----------+----------+
|    BOW     | Brute |          1           |    0.79   |   0.7    |
|   TFIDF    | Brute |         0.4          |    0.74   |   0.65   |
+------------+-------+----------------------+-----------+----------+
```

In [ ]: