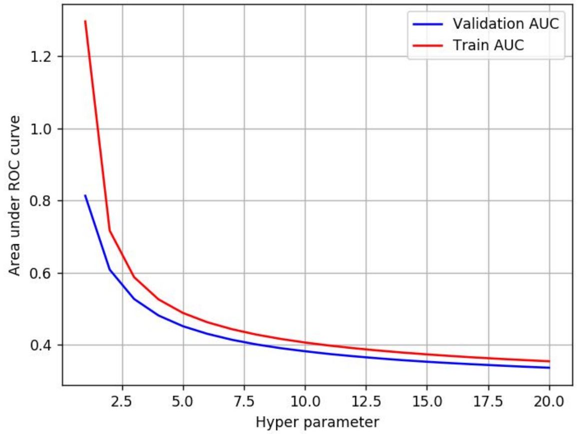
Assignment 6: Apply NB

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 and this

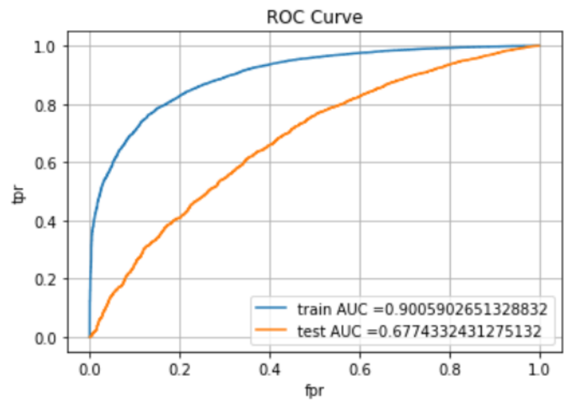
    - Set 1: categorical, numerical features + preprocessed_eassay (BOW)
    - Set 2: categorical, numerical features + preprocessed_eassay (TFIDF)
6. **The hyper paramter tuning(find best alpha:smoothing parameter)**
    - Consider alpha values in range: 10^-5 to 10^2 like [0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,100]

- Explore class_prior = [0.5, 0.5] parameter which can be present in MultinomialNB function(go through this) then check how results might change.
- Find the best hyper parameter which will give the maximum AUC value
- For hyper parameter tuning using k-fold cross validation(use GridsearchCV or RandomsearchCV)/simple cross validation data (write for loop to iterate over hyper parameter values)
- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



-while plotting take log(alpha) on your X-axis so that it will be more readable

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



curve on both train and test.

- Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points

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

-plot the confusion matrix in heatmaps, while plotting the confusion matrix go through the link

7. find the top 20 features from either from feature Set 1 or feature Set 2 using values of $feature_{\log}\_prob$ parameter of $Mt\_\in omialNB$ (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html) and print **BOTH** positive as well as negative corresponding feature names.

   - go through the link

8. You need to summarize the results at the end of the notebook, summarize it in the table format

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

1. Naive Bayes

1.1 Loading Data

In [247…

```
# NECESSARY LIBRARIES:
%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 [248…

```
# read the dataset and fetch 50k datapoints
data = pd.read_csv('preprocessed_data.csv')

# we use only 50k datapoints
mydata = data.iloc[0:50000,:]
print(mydata.shape)

y = mydata["project_is_approved"].values # returns a numpy nd array--> Target variables
X = mydata.drop("project_is_approved",axis = 1) # Creates a dataframe X-->Input variables
mydata.head(3)
```

```
(50000, 9)
```

Out[248…

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categories	clean_
0	ca	mrs	grades_prek_2	53	1	math_science	a he

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categories	clean_
1	ut	ms	grades_3_5	4	1	specialneeds	
2	ca	mrs	grades_prek_2	10	1	literacy_language	

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

In [249…

```
# Split data into train and test set:(stratified sampling)

from sklearn.model_selection import train_test_split # necessary library

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.30,stratify = y,random_state=17)
```

1.3 Make Data Model Ready: encoding essay(BOW)

In [250…

```
#TEXT FEATURE --> ESSAY ENCODING INTO NUMERIC VECTOR:

print("(i) Shape before vectorization of essay(feature) :")
print(X_train.shape,y_train.shape)
print(X_test.shape,y_test.shape)
print("\n")

# Use the BOW vectorizer to encode the text data
vectorizer_bow = CountVectorizer(min_df= 25,ngram_range=(1,2),max_features=20000)
vectorizer_bow.fit(X_train['essay'].values) # fit on train data

# we use the fitted Count Vectorizer to convert the text to vector
X_train_essay_bow = vectorizer_bow.transform(X_train['essay'].values)
```

```
X_test_essay_bow = vectorizer_bow.transform(X_test['essay'].values)

print("(ii) Shape After vectorization of essay(feature) :")
print(X_train_essay_bow.shape, y_train.shape)
print(X_test_essay_bow.shape, y_test.shape)
```

```
(i) Shape before vectorization of essay(feature) :
(35000, 8) (35000,)
(15000, 8) (15000,)


(ii) Shape After vectorization of essay(feature) :
(35000, 20000) (35000,)
(15000, 20000) (15000,)
```

1.4 Make Data Model Ready: encoding numerical, categorical features

ENCODING CATEGORICAL FEATURES:

In [251…

```
# ENCODING CATEGORICAL AND NUMERICAL FEATURES:

# CATEGORICAL FEATURE--> SCHOOL STATE:
vectorizer1 = CountVectorizer()
vectorizer1.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_school = vectorizer1.transform(X_train['school_state'].values)
X_test_state_school = vectorizer1.transform(X_test['school_state'].values)

print("(A) Shape After vectorization of school state :")
print(X_train_state_school.shape, y_train.shape)
print(X_test_state_school.shape, y_test.shape)
print("="*100)




# CATEGORICAL FEATURES TEACHER_PREFIX:
vectorizer2 = CountVectorizer()
vectorizer2.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_prefix_teacher = vectorizer2.transform(X_train['teacher_prefix'].values)
X_test_prefix_teacher = vectorizer2.transform(X_test['teacher_prefix'].values)

print("(B) Shape After vectorization of teacher_prefix :")
print(X_train_prefix_teacher.shape, y_train.shape)
print(X_test_prefix_teacher.shape, y_test.shape)
print("="*100)




# CATEGORICAL FEATURES project_grade_category:
vectorizer3 = CountVectorizer()
vectorizer3.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_cat = vectorizer3.transform(X_train['project_grade_category'].values)
X_test_grade_cat = vectorizer3.transform(X_test['project_grade_category'].values)

print("(C) Shape After vectorization of project_grade_Category :")
print(X_train_grade_cat.shape, y_train.shape)
print(X_test_grade_cat.shape, y_test.shape)
print("="*100)




# CATEGORICAL FEATURES CLEAN_CATEGORIES:

vectorizer4 = CountVectorizer(ngram_range=(1,3))
vectorizer4.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_categories = vectorizer4.transform(X_train['clean_categories'].values)
X_test_categories = vectorizer4.transform(X_test['clean_categories'].values)

print("(D) Shape After vectorization of project Categories :")
print(X_train_categories.shape, y_train.shape)
print(X_test_categories.shape, y_test.shape)
print("="*100)
```

```
# CATEGORICAL FEATURES CLEAN_SUBCATEGORIES:

vectorizer5 = CountVectorizer(ngram_range=(1,2))
vectorizer5.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_subcategories = vectorizer5.transform(X_train['clean_subcategories'].values)
X_test_subcategories = vectorizer5.transform(X_test['clean_subcategories'].values)

print("(E) Shape After vectorization of project Subcategories :")
print(X_train_subcategories.shape, y_train.shape)
print(X_test_subcategories.shape, y_test.shape)
```

```
(A) Shape After vectorization of school state :
(35000, 51) (35000,)
(15000, 51) (15000,)
====================================================================================================
(B) Shape After vectorization of teacher_prefix :
(35000, 5) (35000,)
(15000, 5) (15000,)
====================================================================================================
(C) Shape After vectorization of project_grade_Category :
(35000, 4) (35000,)
(15000, 4) (15000,)
====================================================================================================
(D) Shape After vectorization of project Categories :
(35000, 50) (35000,)
(15000, 50) (15000,)
====================================================================================================
(E) Shape After vectorization of project Subcategories :
(35000, 347) (35000,)
(15000, 347) (15000,)
```

ENCODING NUMERICAL FEATURES:

PRICE:

In [252…

```
# Normalizing: map all the values to range of (0,1)

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

```
# fit and transform the train and test data:
# reshape of the data to single row allows the normalizer() to fit & transform
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))

print("(F) Shape After Nomalization of feature Price :")
print(X_train_price_norm.transpose().shape, y_train.shape)
print(X_test_price_norm.transpose().shape, y_test.shape)

# assign to another variable for readability purpose :
X_train_price_norm = X_train_price_norm.transpose()
X_test_price_norm  = X_test_price_norm.transpose()
```

```
(F) Shape After Nomalization of feature Price :
(35000, 1) (35000,)
(15000, 1) (15000,)
```

TEACHER_NUMBER_OF_PREVIOUSLY_POSTED_PROJECTS:

In [253…

```
# Normalizing:map all the values to range of (0,1)

from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

# fit and transform the train and test data:
# reshape of the data to single row allows the normalizer() to fit & transform
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
X_train_previous_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1
X_test_previous_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-

print("(G) Shape After Normalization of Number of previously posted projects :")
print(X_train_previous_norm.transpose().shape, y_train.shape)
print(X_test_previous_norm.transpose().shape, y_test.shape)

# assign to another variable for readability purpose :
X_train_previous_norm = X_train_previous_norm.transpose()
X_test_previous_norm = X_test_previous_norm.transpose()
```

```
(G) Shape After Normalization of Number of previously posted projects :
(35000, 1) (35000,)
```

```
(15000, 1) (15000,)
```

CONCATENATING CATEGORICAL ,NUMERICAL & TEXT FEATURES:

In [254…

```
# concatenate all the features :

from scipy.sparse import hstack

# hstack() helps in concatenating "n" number of array like shapes into one dataframe.
# we store the concatenated outcome in a csr matrix format.

train_X = hstack((X_train_essay_bow,X_train_state_school,
                  X_train_prefix_teacher,X_train_grade_cat,
                 X_train_categories,X_train_subcategories,
                 X_train_price_norm,X_train_previous_norm)).tocsr()

test_X = hstack((X_test_essay_bow,X_test_state_school,
                  X_test_prefix_teacher,X_test_grade_cat,
                 X_test_categories,X_test_subcategories,
                 X_test_price_norm,X_test_previous_norm)).tocsr()


print("(H) Final Data matrix :")

print(train_X.shape, y_train.shape)#we totally have 35000 rows & 20449 columns in train data

print(test_X.shape, y_test.shape)#we totally have 15000 rows & 20449 columns in test data
```

```
(H) Final Data matrix :
(35000, 20459) (35000,)
(15000, 20459) (15000,)
```

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

For Set 1:

Train the model and find optimal hyperparameter(SET 1):

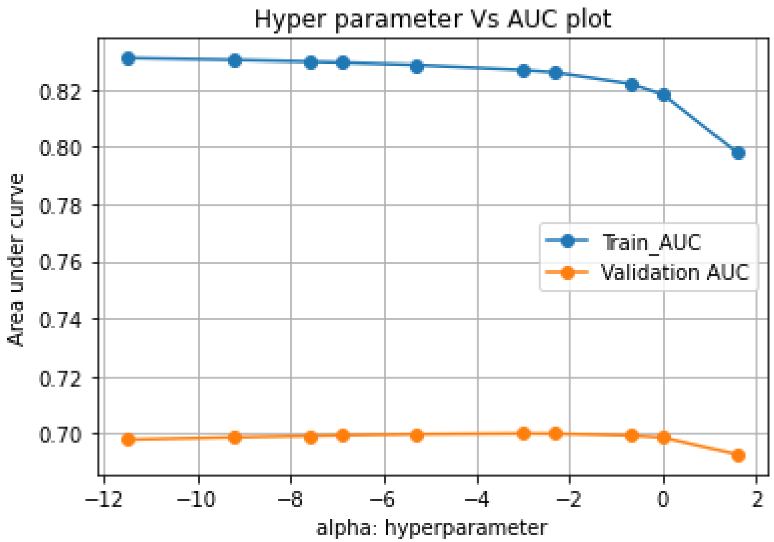
In [257…

```
#Reference : https://iq.opengenus.org/naive-bayes-on-tf-idf-vectorized-matrix/
#Reference : https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

# build naive bayes model

# import necessary libraries:
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import RandomizedSearchCV
import math
from time import time

start = time()

# fit the model to the training data using randomsearchCV
model = MultinomialNB()
param = {'alpha':[0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10]}

# use "ROC_AUC" as a scoring and CV = 10
clf = RandomizedSearchCV(model,param,cv=10,scoring='roc_auc',return_train_score = True)
clf.fit(train_X, y_train)

# make a dataframe out of cv_results:
cv_results = pd.DataFrame.from_dict(clf.cv_results_)
cv_results = cv_results.sort_values(['param_alpha'])

# obtain mean train,Cv scores and their corresponding hyperparameter alpha:
train_auc = cv_results['mean_train_score']
cv_auc = cv_results['mean_test_score']
alpha =  cv_results['param_alpha']

log_alpha = [math.log(val) for val in alpha] # take log(alpha) for stability purpose
print("(A) The alpha values are :",alpha.values)
print("\n")
print("(B) The log of alpha values are :",log_alpha)
print("\n")

# lets observe the point of minimal gap between CV and train curves:
difference = train_auc - cv_auc
print("(C) Difference between train_auc & cv_auc :\n",difference)
```

```
print("\n")
print("(D) The minimal difference is :",min(difference))
print("\n")
print("(E) The minimal difference is observed at :",alpha[difference.idxmin()])
print("\n")

# plot the Hyperparameters vs AUC plot for train and Cv data:
plt.plot(log_alpha,train_auc,label = "Train_AUC",marker = "o")
plt.plot(log_alpha,cv_auc,label = "Validation AUC",marker = "o")
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("Area under curve")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

end = time()
training_time = end - start
print("training & validation time: %0.2fs" % training_time)

# observe the validation_results
cv_results.head(10)
```

```
(A) The alpha values are : [1e-05 0.0001 0.0005 0.001 0.005 0.05 0.1 0.5 1 5]


(B) The log of alpha values are : [-11.512925464970229, -9.210340371976182, -7.600902459542082, -6.907755278982137, -
5.298317366548036, -2.995732273553991, -2.3025850929940455, -0.6931471805599453, 0.0, 1.6094379124341003]


(C) Difference between train_auc & cv_auc :
 8    0.133225
9    0.131759
2    0.130675
0    0.130174
5    0.128992
6    0.126994
7    0.126203
4    0.122885
3    0.119964
1    0.105346
dtype: float64
```

```
(D) The minimal difference is : 0.10534554160980036

(E) The minimal difference is observed at : 5
```




```
training & validation time: 15.63s
```

Out[257…

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_test_score	split3_
8	0.083575	0.012716	0.007580	0.000662	1e-05	{'alpha': 1e-05}	0.699092	0.713211	0.694922	
9	0.078687	0.008270	0.009468	0.005844	0.0001	{'alpha': 0.0001}	0.699400	0.714896	0.695864	
2	0.078285	0.009385	0.007882	0.000833	0.0005	{'alpha': 0.0005}	0.699649	0.715763	0.696525	
0	0.078689	0.010294	0.008281	0.001342	0.001	{'alpha': 0.001}	0.699691	0.716003	0.696732	
5	0.077387	0.008052	0.007682	0.000895	0.005	{'alpha': 0.005}	0.699820	0.716302	0.697152	
6	0.075703	0.007860	0.007882	0.000941	0.05	{'alpha': 0.05}	0.699916	0.716437	0.697507	

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_test_score	split3_
7	0.079881	0.008727	0.007978	0.000994	0.1	{'alpha': 0.1}	0.699805	0.716322	0.697476	
4	0.078684	0.010011	0.007488	0.000662	0.5	{'alpha': 0.5}	0.699267	0.715517	0.697031	
3	0.077698	0.004769	0.008772	0.001719	1	{'alpha': 1}	0.698717	0.714931	0.696297	
1	0.084175	0.013606	0.008482	0.002572	5	{'alpha': 5}	0.694339	0.708986	0.691250	

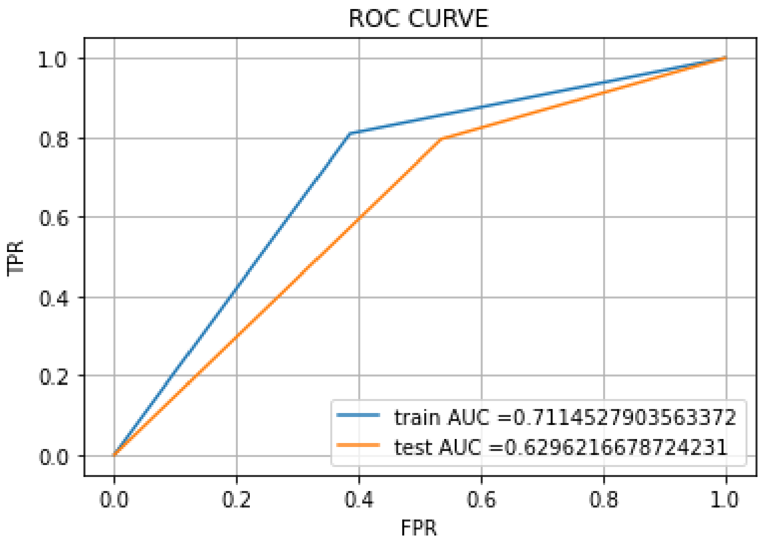
10 rows × 31 columns

Testing performance of model on test data(SET 1):

In [258…

```
#REFERENCE :https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

from sklearn.metrics import roc_curve, auc # -->necessary libraries

# Fit the classifier with the optimal alpha:
clf = MultinomialNB(alpha = 5,fit_prior = True)
clf.fit(train_X,y_train)

# predict for train and test data :
y_pred_train = clf.predict(train_X)
y_pred_test = clf.predict(test_X)

# compute TPR,FPR values to construt ROC curve:
train_fpr,train_tpr,tr_thresholds = roc_curve(y_train,y_pred_train)
test_fpr,test_tpr,te_thresholds = roc_curve(y_test,y_pred_test)

#plot the ROC with Train AUC and Test AUC:
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")
```

```
plt.grid()
plt.show()
```



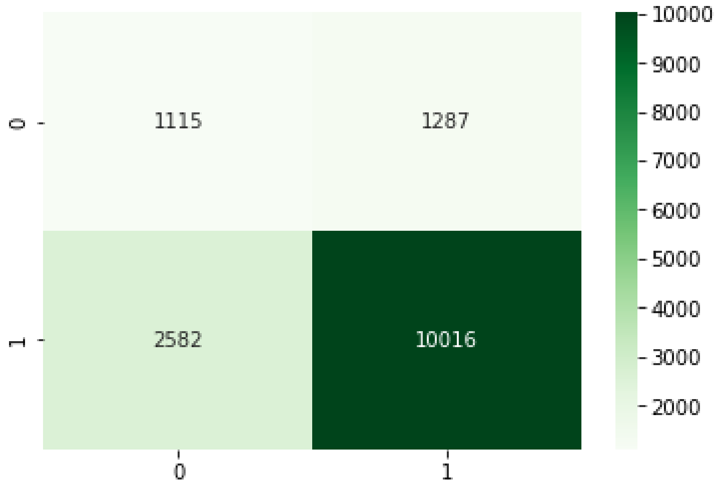
In [260…

```
# compute confusion matrix:
# Reference :https://stackoverflow.com/questions/61748441/how-to-fix-the-values-displayed-in-a-confusion-matrix-in-ex
from sklearn.metrics import confusion_matrix
confusion_mat = confusion_matrix(y_test,y_pred_test)
print(confusion_mat)

# Represent confusion matrix as a heatmap:
cm = np.array([[1115,1287],[2582,10016]])
sns.heatmap(cm, annot=True,fmt="d",cmap='Greens')
```

```
[[ 1115  1287]
 [ 2582 10016]]
```

Out[260…

```
<AxesSubplot:>
```



Observation: Here we can clearly observe that false positives are high than True negatives because of severe imbalance in the dataset where positive class dominates over negative class.

For Set 2:

Encoding essay(TFIDF):

In [261…

```
#TEXT FEATURE --> ESSAY ENCODING INTO NUMERIC VECTOR(tfidf):

from sklearn.feature_extraction.text import TfidfVectorizer

print("(i) Shape before vectorization of essay(feature) :")
print(X_train.shape,y_train.shape)
print(X_test.shape,y_test.shape)
print("\n")

# Use the tfidf vectorizer to encode the text data
vectorizer = TfidfVectorizer(min_df= 30,ngram_range=(1,2),max_features=20000)
vectorizer.fit(X_train['essay'].values) # fit on train data

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

```
print("(ii) Shape After vectorization of essay(feature) :")
print(X_train_essay_bow.shape, y_train.shape)
print(X_test_essay_bow.shape, y_test.shape)
```

```
(i) Shape before vectorization of essay(feature) :
(35000, 8) (35000,)
(15000, 8) (15000,)


(ii) Shape After vectorization of essay(feature) :
(35000, 20000) (35000,)
(15000, 20000) (15000,)
```

Concatenating features for SET 2:

In [262…

```
# concatenate all the features :

from scipy.sparse import hstack

# hstack() helps in concatenating "n" number of array like shapes into one dataframe.
# we store the concatenated outcome in a csr matrix format.

train_X = hstack((X_train_essay_bow,X_train_state_school,
                  X_train_prefix_teacher,X_train_grade_cat,
                 X_train_categories,X_train_subcategories,
                 X_train_price_norm,X_train_previous_norm)).tocsr()

test_X = hstack((X_test_essay_bow,X_test_state_school,
                  X_test_prefix_teacher,X_test_grade_cat,
                 X_test_categories,X_test_subcategories,
                 X_test_price_norm,X_test_previous_norm)).tocsr()

print("(H) Final Data matrix :")

print(train_X.shape, y_train.shape)#we totally have 35000 rows & 20449 columns in train data

print(test_X.shape, y_test.shape)#we totally have 15000 rows & 20449 columns in test data
```

```
(H) Final Data matrix :
(35000, 20459) (35000,)
(15000, 20459) (15000,)
```

Train the model and find optimal hyperparameter(SET 2):

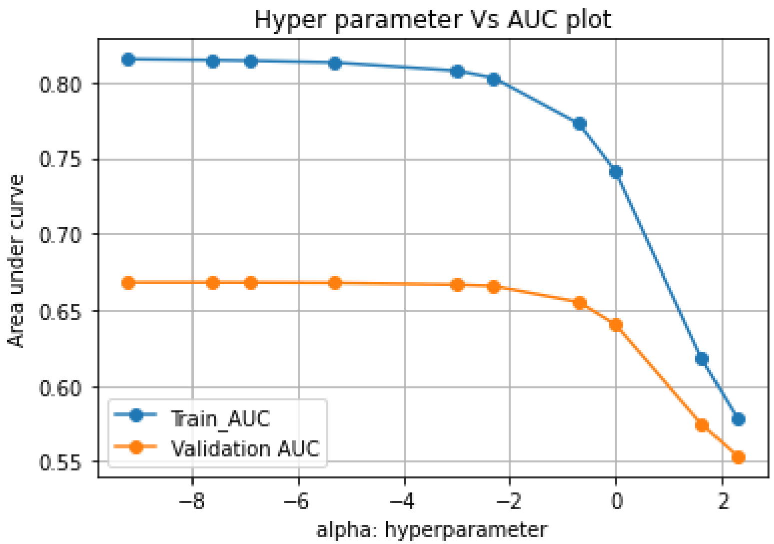
In [263…

```
# build naive bayes model:

# import necessary libraries:
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import RandomizedSearchCV
import math
from time import time

start = time()

# Build the model & cross validate using randomsearchCV:
model = MultinomialNB()
param = {'alpha':[0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50]}

clf2 = RandomizedSearchCV(model,param,cv=10,scoring='roc_auc',return_train_score = True)
clf2.fit(train_X, y_train)

# make a dataframe out of validation_results:
cv_results = pd.DataFrame.from_dict(clf2.cv_results_)
cv_results = cv_results.sort_values(['param_alpha'])

# store mean train, Cv scores and their corresponding alpha:
train_auc = cv_results['mean_train_score']
cv_auc = cv_results['mean_test_score']
alpha =  cv_results['param_alpha']

# Use log alpha to ensure numerical stability:
log_alpha = [math.log(val) for val in alpha]
print("(A) The alpha values are :",alpha.values)
print("\n")
print("(B) The log of alpha values are :",log_alpha)
print("\n")

#lets observe the point of minimal gap between CV and train curves:
difference = train_auc - cv_auc
print("(C) Difference between train_auc & cv_auc :\n",difference)
print("\n")
print("(D) The minimal difference is :",min(difference))
print("\n")
```

```
print("(E) The minimal difference is observed at :",alpha[difference.idxmin()])
print("\n")

# plot the Hyperparameters vs AUC plot for train and Cv data:
plt.plot(log_alpha,train_auc,label = "Train_AUC",marker = "o")
plt.plot(log_alpha,cv_auc,label = "Validation AUC",marker = "o")
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("Area under curve")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

end = time()
training_time = end - start
print("training & validation time: %0.2fs" % training_time)

# observe the Cv_results:
cv_results.head(10)
```

```
(A) The alpha values are : [0.0001 0.0005 0.001 0.005 0.05 0.1 0.5 1 5 10]


(B) The log of alpha values are : [-9.210340371976182, -7.600902459542082, -6.907755278982137, -5.298317366548036, -
2.995732273553991, -2.3025850929940455, -0.6931471805599453, 0.0, 1.6094379124341003, 2.302585092994046]


(C) Difference between train_auc & cv_auc :
 5    0.146989
8    0.146336
3    0.146016
9    0.145056
6    0.140737
1    0.136988
2    0.117389
4    0.100822
7    0.043538
0    0.024854
dtype: float64


(D) The minimal difference is : 0.024854338701383427
```

```
(E) The minimal difference is observed at : 10
```




```
training & validation time: 14.51s
```

Out[263…

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_test_score	split3_
5	0.071509	0.004343	0.007381	0.000489	0.0001	{'alpha': 0.0001}	0.679810	0.679883	0.658557	
8	0.070514	0.004368	0.007481	0.000499	0.0005	{'alpha': 0.0005}	0.679682	0.679815	0.658034	
3	0.077593	0.013954	0.007879	0.001132	0.001	{'alpha': 0.001}	0.679667	0.679774	0.657753	
9	0.071708	0.003689	0.007480	0.000499	0.005	{'alpha': 0.005}	0.679373	0.679619	0.656933	
6	0.071508	0.002853	0.007681	0.000897	0.05	{'alpha': 0.05}	0.677736	0.680163	0.654921	
1	0.070811	0.004987	0.007577	0.000486	0.1	{'alpha': 0.1}	0.676622	0.681605	0.653995	
2	0.073690	0.005049	0.007679	0.000458	0.5	{'alpha': 0.5}	0.668109	0.668864	0.646082	

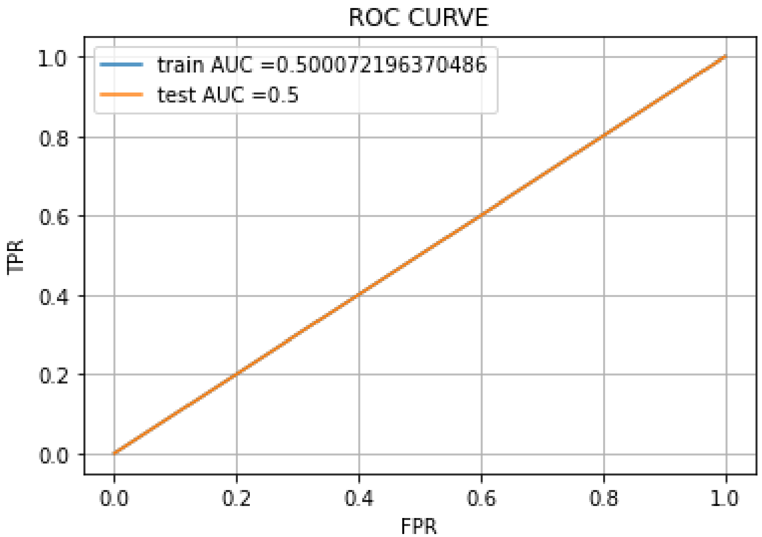
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_test_score	split3_
4	0.075398	0.010173	0.007279	0.000453	1	{'alpha': 1}	0.656970	0.649968	0.631780	
7	0.072007	0.004107	0.007477	0.000502	5	{'alpha': 5}	0.607112	0.573305	0.563718	
0	0.073996	0.005168	0.007777	0.000741	10	{'alpha': 10}	0.588692	0.548175	0.542662	

10 rows × 31 columns

Test the model performance on Test data(SET 2):

In [265…

```
from sklearn.metrics import roc_curve, auc

# Fit the classifier with the optimal alpha:
clf2 = MultinomialNB(alpha = 10,fit_prior = True)
clf2.fit(train_X,y_train)

# predict for train and test data :
y_pred_train = clf2.predict(train_X)
y_pred_test_tfidf = clf2.predict(test_X)

# compute TPR,FPR values to construt ROC curve:
train_fpr,train_tpr,tr_thresholds = roc_curve(y_train,y_pred_train)
test_fpr,test_tpr,te_thresholds = roc_curve(y_test,y_pred_test_tfidf)

#plot the ROC with Train AUC and Test AUC:
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")
plt.grid()
plt.show()
```



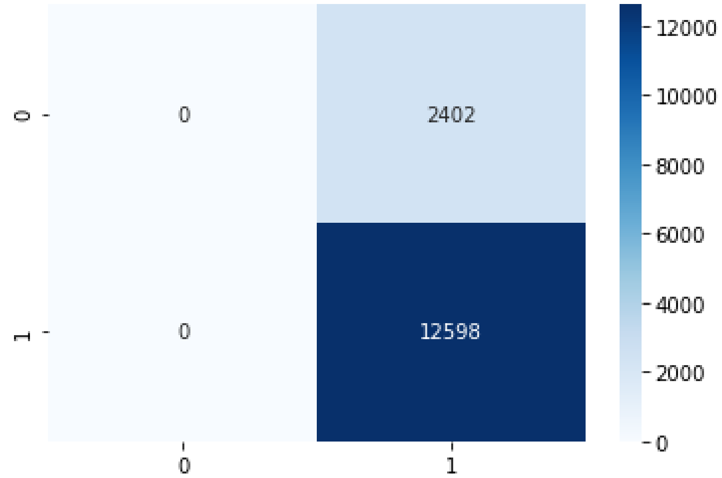
In [272…

```
# compute confusion matrix:
# Reference :https://stackoverflow.com/questions/61748441/how-to-fix-the-values-displayed-in-a-confusion-matrix-in-ex
from sklearn.metrics import confusion_matrix
confusion_mat = confusion_matrix(y_test,y_pred_test_tfidf)
print(confusion_mat)

# Represent confusion matrix using Heatmap:
cm = np.array([[0,2402],[0,12598]])
sns.heatmap(cm, annot=True,fmt="d",cmap='Blues')
```

```
[[    0  2402]
 [    0 12598]]
```

Out[272…

```
<AxesSubplot:>
```



Observation: Here all the data points are classified as positive because of high imbalance and it behaves like a random model because of wrong predictions.

1.6 Pick top 20 features that helps in determining class labels in SET 1:

In [268…

```
#reference:https://numpy.org/doc/stable/reference/generated/numpy.argsort.html
# pick top 20 useful features:

features_likelihoods_neg_class = clf.feature_log_prob_[0]
features_likelihoods_pos_class = clf.feature_log_prob_[1]

# sort the likelihoods and reverse it to get descending order:
# fetch top 20 indices corresponding to useful features:

sorted_pos = np.argsort(features_likelihoods_pos_class)
rev_sorted_pos = sorted_pos[::-1] # we will get indices

sorted_neg = np.argsort(features_likelihoods_neg_class)
rev_sorted_neg = sorted_neg[::-1] # we will get indices

# store all the feature names of train data in a list:
mylist = []
mylist.extend(vectorizer_bow.get_feature_names())
```

```python
mylist.extend(vectorizer1.get_feature_names())
mylist.extend(vectorizer2.get_feature_names())
mylist.extend(vectorizer3.get_feature_names())
mylist.extend(vectorizer4.get_feature_names())
mylist.extend(vectorizer5.get_feature_names())
mylist.append("Price")
mylist.append("Teacher_number_of_previously_posted_projects")


# code to find top20 useful features corresponding to positive class:
pos_indices = rev_sorted_pos[0:20]
top20_pos = []
for i in pos_indices:
    top20_pos.append(mylist[i])
print("(A) Top 20 features corresponding to positive class :\n",top20_pos)
print("\n")

# code to find top20 useful features corresponding to negative class:
neg_indices = rev_sorted_neg[0:20]
top20_neg = []
for i in neg_indices:
    top20_neg.append(mylist[i])
print("(B) Top 20 features corresponding to negative class :\n",top20_neg)
```

```
(A) Top 20 features corresponding to positive class :
 ['students', 'school', 'my', 'learning', 'classroom', 'the', 'not', 'they', 'learn', 'my students', 'help', 'many', 'nannan', 'work', 'we', 'reading', 'need', 'use', 'day', 'able']


(B) Top 20 features corresponding to negative class :
 ['students', 'school', 'learning', 'my', 'classroom', 'not', 'learn', 'they', 'help', 'the', 'my students', 'many', 'nannan', 'need', 'we', 'work', 'come', 'year', 'reading', 'able']
```

1.7 Summary

as mentioned in the step 5 of instructions

In [267…

```python
from prettytable import PrettyTable
```

```python
# final results of the task:
# create a table with desired attributes:
summary = [["BOW","MultinomialNB","5","0.62"],["TFIDF","MultinomialNB","10","0.50"]]
table = PrettyTable(["Vectorizer","Model","Hyperparameter","AUC"])

# add rows to the table:
for j in summary:
    table.add_row(j)

print(table)
```

```
+------------+---------------+----------------+------+
| Vectorizer |     Model     | Hyperparameter | AUC  |
+------------+---------------+----------------+------+
|    BOW     | MultinomialNB |       5        | 0.62 |
|   TFIDF    | MultinomialNB |       10       | 0.50 |
+------------+---------------+----------------+------+
```