```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import sqlite3
import pandas
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import Normalizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
from tqdm import tqdm
import os
from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
from sklearn.naive_bayes import MultinomialNB
from prettytable import PrettyTable
```

```
import csv
import math as m
from sklearn.preprocessing import Normalizer
```

```
data = pandas.read_csv('/content/drive/MyDrive/preprocessed_data (1).csv')
```

```
!pip install plotly==3.10.0
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting plotly==3.10.0
  Downloading plotly-3.10.0-py2.py3-none-any.whl (41.5 MB)
     |████████████████████████████████| 41.5 MB 58.5 MB/s
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from plotly==3.10.0) (2.23.0)
Collecting retrying>=1.3.3
  Downloading retrying-1.3.3.tar.gz (10 kB)
Requirement already satisfied: decorator>=4.0.6 in /usr/local/lib/python3.7/dist-packages (from plotly==3.10.0) (4.4.2)
Requirement already satisfied: pytz in /usr/local/lib/python3.7/dist-packages (from plotly==3.10.0) (2022.4)
Requirement already satisfied: nbformat>=4.2 in /usr/local/lib/python3.7/dist-packages (from plotly==3.10.0) (5.7.0)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from plotly==3.10.0) (1.15.0)
Requirement already satisfied: traitlets>=5.1 in /usr/local/lib/python3.7/dist-packages (from nbformat>=4.2->plotly==3.10.0) (5
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.7/dist-packages (from nbformat>=4.2->plotly==3.10.0) (
Requirement already satisfied: fastjsonschema in /usr/local/lib/python3.7/dist-packages (from nbformat>=4.2->plotly==3.10.0) (2
Requirement already satisfied: importlib-metadata>=3.6 in /usr/local/lib/python3.7/dist-packages (from nbformat>=4.2->plotly==3
```

```
y = data['project_is_approved'].values
data.drop(['project_is_approved'], axis='columns', inplace=True)
X = data
```

```
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in /usr/local/lib/python3.7/dist-packages (from js
```

```
data.columns
```

```
Index(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects', 'clean_categories',
       'clean_subcategories', 'essay', 'price'],
      dtype='object')
```

```
Stored in directory: /root/.cache/pip/wheels/f9/8d/8d/f6af3f7f9ea3553bc2fe6d53e4b287dad18b06a861ac56ddf
```

```
# train test split
from sklearn.model_selection import train_test_split
# from sklearn importing train_test_split to split the data
# test size 0.3 mean 70% splitted as train and 30% as test

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y)
```

```
WARNING: The following packages were previously imported in this runtime:
```

```
vectorizer_bow = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=50000)
```

```
# # HERE WE ARE FITTING THE TEXT FEATURE ESSAY
vectorizer_bow.fit(X_train['essay'].values)  #fitting

# vectorizing the train essay features
X_train_essay_bow = vectorizer_bow.transform(X_train['essay'].values)

# vectorizing the test  essay features
X_test_essay_bow = vectorizer_bow.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
print(X_test_essay_bow.shape, y_test.shape)
print("==>"*100)
```

```
After vectorizations
(76473, 50000) (76473,)
(32775, 50000) (32775,)
==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>=
```

1.4 Make Data Model Ready: encoding numerical, categorical features

```
tfidfvectorizer = TfidfVectorizer(min_df=10,max_features=50000)
text_tfidf = tfidfvectorizer.fit(X_train['essay'].values) #fitting

# vectorizing the train essay features
X_train_essay_tfidf =tfidfvectorizer.transform(X_train['essay'].values)

# vectorizing the train essay features
X_test_essay_tfidf =tfidfvectorizer.transform(X_test['essay'].values)

print("After vectorizations")
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)
```

```
After vectorizations
(76473, 14464) (76473,)
(32775, 14464) (32775,)
==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>=
```

```
vectorizer_school_state = CountVectorizer()
vectorizer_school_state.fit(X_train['school_state'].values) # fitting

X_train_state_ohe = vectorizer_school_state.transform(X_train['school_state'].values)
#X_cv_state_ohe = vectorizer_state.transform(X_cv['school_state'].values)  #transform
X_test_state_ohe = vectorizer_school_state.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("==>"*100)
```

```
After vectorizations
(76473, 51) (76473,)
(32775, 51) (32775,)
==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>=
```

```
vectorizer_clean_categories = CountVectorizer()
vectorizer_clean_categories.fit(X_train['clean_categories'].values) # fitting

X_train_category_ohe = vectorizer_clean_categories.transform(X_train['clean_categories'].values)
#X_cv_category_ohe = vectorizer.transform(X_cv['clean_categories'].values)   #transform
X_test_category_ohe = vectorizer_clean_categories.transform(X_test['clean_categories'].values)

print("After vectorizations")
```

```
print(X_train_category_ohe.shape, y_train.shape)
#print(X_cv_category_ohe.shape, y_cv.shape)
print(X_test_category_ohe.shape, y_test.shape)
print("==>"*100)
```

```
After vectorizations
(76473, 9) (76473,)
(32775, 9) (32775,)
==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>=
```

```
vectorizer_clean_subcategories = CountVectorizer()
vectorizer_clean_subcategories.fit(X_train['clean_subcategories'].values) # fitting

X_train_subcategory_ohe = vectorizer_clean_subcategories.transform(X_train['clean_subcategories'].values)
#X_cv_subcategory_ohe = vectorizer_sub.transform(X_cv['clean_subcategories'].values)  #transform
X_test_subcategory_ohe = vectorizer_clean_subcategories.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_subcategory_ohe.shape, y_train.shape)
#print(X_cv_subcategory_ohe.shape, y_cv.shape)
print(X_test_subcategory_ohe.shape, y_test.shape)
print("==>"*100)
```

```
After vectorizations
(76473, 30) (76473,)
(32775, 30) (32775,)
==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>=
```

```
vectorizer_project_grade_category = CountVectorizer()
vectorizer_project_grade_category.fit(X_train['project_grade_category'].values) # fitting

X_train_grade_ohe = vectorizer_project_grade_category.transform(X_train['project_grade_category'].values)
#X_cv_grade_ohe = vectorizer_grade.transform(X_cv['project_grade_category'].values)  #transform
X_test_grade_ohe = vectorizer_project_grade_category.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("==>"*100)
```

```
After vectorizations
(76473, 4) (76473,)
(32775, 4) (32775,)
==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>=
```

```
vectorizer_teacher_prefix= CountVectorizer()
vectorizer_teacher_prefix.fit(X_train['teacher_prefix'].values) # fitting

X_train_teacher_ohe = vectorizer_teacher_prefix.transform(X_train['teacher_prefix'].values)
#X_cv_teacher_ohe = vectorizer_prefix.transform(X_cv['teacher_prefix'].values)  #transform
X_test_teacher_ohe = vectorizer_teacher_prefix.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("==>"*100)
```

```
After vectorizations
(76473, 5) (76473,)
(32775, 5) (32775,)
==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>=
```

Vectorizing Numerical Data

```
from sklearn.preprocessing import MinMaxScaler   # USING MINMAX SCALER TO AVOID DIMENSIONAL ERROR
price_scalar = MinMaxScaler()       # HERE WE ARE SCALLING THE FEATURE

X_train_price = price_scalar.fit_transform(X_train['price'].values.reshape(-1, 1))          # USING FIT_TRANFORM TO FITTING THE DATA
X_test_price = price_scalar.transform(X_test['price'].values.reshape(-1, 1))
print("After vectorizations")
print(X_train_price.shape, y_train.shape)                 # PRINT THE SHAPE
print(X_test_price.shape, y_test.shape)
print("==>"*100)
```

```
After vectorizations
(76473, 1) (76473,)
(32775, 1) (32775,)
==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>=
```

```
from sklearn.preprocessing import MinMaxScaler   # USING MINMAX SCALER TO AVOID DIMENSIONAL ERROR
teacher_number_of_previously_posted_projects_scalar = MinMaxScaler()       # HERE WE ARE SCALLING THE FEATURE

X_train_teacher_number_of_previously_posted_projects = teacher_number_of_previously_posted_projects_scalar.fit_transform(X_train['tea
X_test_teacher_number_of_previously_posted_projects = teacher_number_of_previously_posted_projects_scalar.transform(X_test['teacher_n
print("After vectorizations")
print(X_train_teacher_number_of_previously_posted_projects.shape, y_train.shape)                 # PRINT THE SHAPE
print(X_test_teacher_number_of_previously_posted_projects.shape, y_test.shape)
print("==>"*100)
```

```
After vectorizations
(76473, 1) (76473,)
(32775, 1) (32775,)
==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>=
```

Concatinating all the features

SET-1

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

```
from scipy.sparse import hstack
X_tr_set_one = hstack((X_train_essay_bow, X_train_state_ohe,
                       X_train_teacher_ohe, X_train_grade_ohe,
                       X_train_price,
                       X_train_category_ohe,
                       X_train_subcategory_ohe,
                       X_train_teacher_number_of_previously_posted_projects )).tocsr()

X_te_set_one = hstack((X_test_essay_bow,
                       X_test_state_ohe,
                       X_test_teacher_ohe,
                       X_test_grade_ohe,
                       X_test_price,
                       X_test_category_ohe,
                       X_test_subcategory_ohe,
                       X_test_teacher_number_of_previously_posted_projects)).tocsr()
```

```
print("SHAPE OF TRAIN AND TEST AFTER STACKING")
print(X_tr_set_one.shape)
print(X_te_set_one.shape)
```

```
SHAPE OF TRAIN AND TEST AFTER STACKING
(76473, 50101)
(32775, 50101)
```

Double-click (or enter) to edit

```
from scipy.sparse import hstack
```

```
X_tr_set_two = hstack((X_train_essay_bow,
                       X_train_state_ohe,
                       X_train_teacher_ohe,
                       X_train_grade_ohe,
                       X_train_price,
                       X_train_category_ohe,
                       X_train_subcategory_ohe,
                       X_train_teacher_number_of_previously_posted_projects )).tocsr()

X_te_set_two = hstack((X_test_essay_bow,
                       X_test_state_ohe,
                       X_test_teacher_ohe,
                       X_test_grade_ohe,
                       X_test_price,
                       X_test_category_ohe,
                       X_test_subcategory_ohe,
                       X_test_teacher_number_of_previously_posted_projects)).tocsr()



print("SHAPE OF TRAIN AND TEST AFTER STACKING")
print(X_tr_set_two.shape)
print(X_te_set_two.shape)
```

```
SHAPE OF TRAIN AND TEST AFTER STACKING
(76473, 50101)
(32775, 50101)
```

MULTINOMIAL NAIVE BAYES USING GRID SEARCH CROSS VALIDATION (SET - 1)

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

from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
```

```
model=MultinomialNB(class_prior=[0.5,0.5])
param={'alpha': [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]}
clf=GridSearchCV(model,param,scoring='roc_auc',
                 cv=12,return_train_score=True) # running 10 fold cross validation grid search

clf.fit(X_tr_set_one,y_train) #FITTING THE MODEL
```

```
GridSearchCV(cv=12, estimator=MultinomialNB(class_prior=[0.5, 0.5]),
             param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.1, 1, 10, 100,
                                   1000]},
             return_train_score=True, scoring='roc_auc')
```

```
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']
```

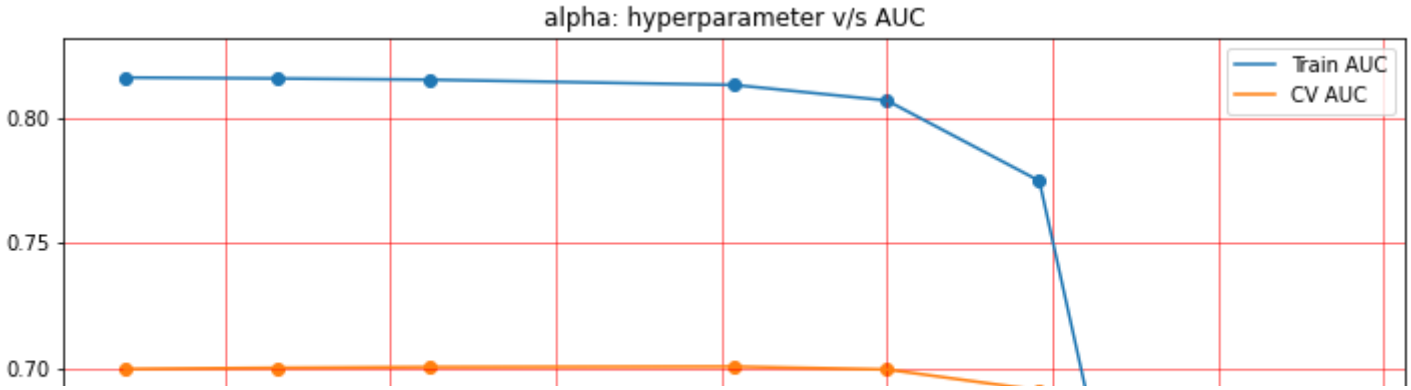
Double-click (or enter) to edit

```
#HERE I M TRYING TO GET THE BEST VALUE FOR ALGO
print('THE BEST CLF SCORE IS : ',clf.best_score_)

# THE I M TRYING TO FINDOUT THAT WHAT SHOULD BE BEST VALUE FOR GBDT PARAMETERS VALUE
print('BEST VALUE FOR HYPERPARAMETER : ',clf.best_params_)
print('==>'*100)
```

```
THE BEST CLF SCORE IS :  0.7006215706152047
BEST VALUE FOR HYPERPARAMETER :  {'alpha': 0.1}
==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>=
```

plotting hyperparameter v/s auc

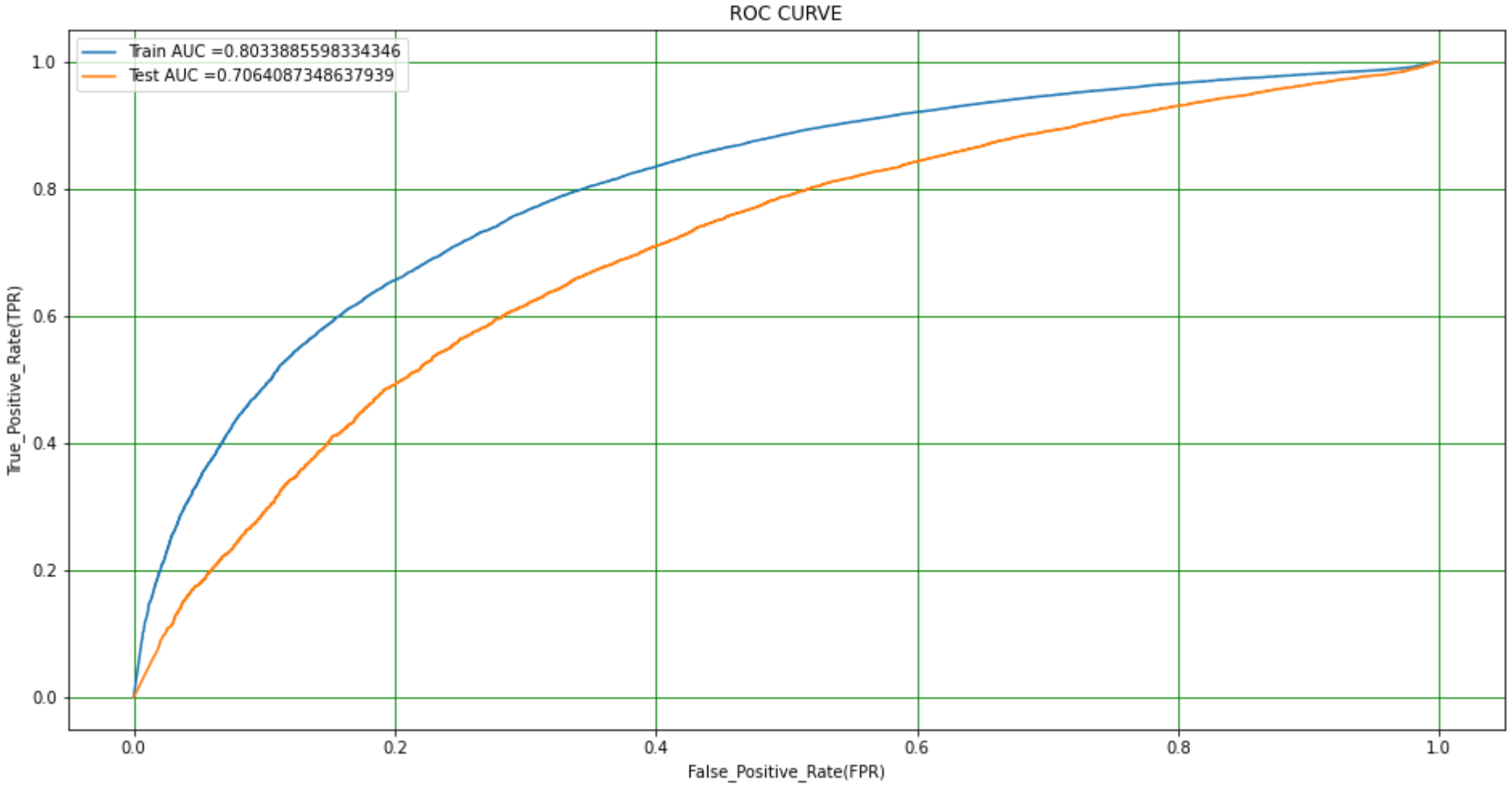
```
log_param=[]
for i in param['alpha']:  # converting alpha into log- alpha
    log_param.append(m.log(i))

plt.figure(figsize=(12,8))
plt.grid()
plt.plot(log_param, train_auc, label='Train AUC')
plt.plot(log_param, cv_auc, label='CV AUC')
plt.scatter(log_param,train_auc)
plt.scatter(log_param,cv_auc)
plt.legend()
plt.xlabel("log alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("alpha: hyperparameter v/s AUC")
plt.grid(color='red', linestyle='-', linewidth=0.5)
```



roc plot of train and test data

```
model_set1=MultinomialNB(alpha=1e-05,class_prior=[0.5,0.5])
model_set1.fit(X_tr_set_one,y_train)

# converting train and test output into probability
y_train_probs = clf.predict_proba(X_tr_set_one)[:,1] # converting train and test output into probability
y_test_probs= clf.predict_proba(X_te_set_one )[:,1]

 # storing values of fpr and tpr
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_probs)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_probs)


# PLOTING THE ROC CURVE
plt.figure(figsize=(16,8))
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()

# NOW WE LABEL THE PLOT X AND Y
plt.xlabel("False_Positive_Rate(FPR)")
plt.ylabel("True_Positive_Rate(TPR)")
# TITLE OF THE CURVE IS ROC CURVE
```

```
plt.title("ROC CURVE")

# DEFINING THE GRID PARAMETERS
plt.grid(color='green',lw=0.8)
```



confusion matrix

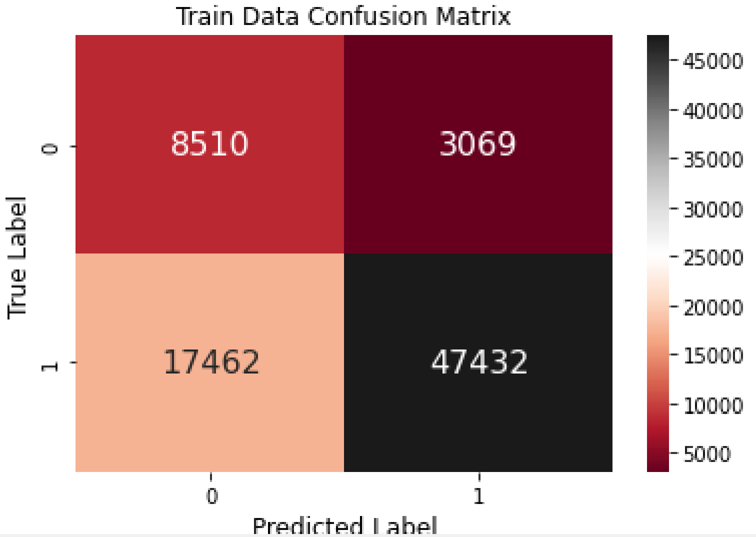
```
# REFER : https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

def find_best_threshold(threshould, fpr, tpr):
    Thres = 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(Thres,3))
    return Thres

# DEFINING THE THRESHOLD VALUE IF VALUE GREATHER THAN THRESHOLD THEN 1 AND IF ITS LESS THAN THEN ITS 0
def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

Train data

```
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
CONFUSION_MATRIX=metrics.confusion_matrix(y_train,predict_with_best_t(y_train_probs, best_t))


# REFER :  https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
# REFER :  https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

print("CONFUSION MATRIX OF TRAIN DATA")
print("\n")
print('==>'*50)
print(CONFUSION_MATRIX)
sns.heatmap(CONFUSION_MATRIX, annot=True, fmt='d',cmap='RdGy',annot_kws = {"size":16})
```

```
plt.ylabel('True Label',size=12)
plt.xlabel('Predicted Label',size=12)
plt.title('Train Data Confusion Matrix',size=12)
```

```
the maximum value of tpr*(1-fpr) 0.5371868823700227 for threshold 0.577
CONFUSION MATRIX OF TRAIN DATA


==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>=
[[ 8510  3069]
 [17462 47432]]
Text(0.5, 1.0, 'Train Data Confusion Matrix')
```

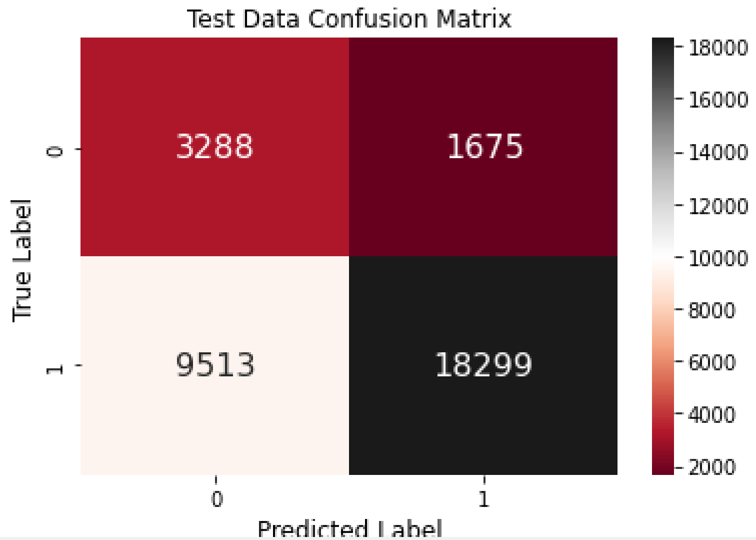


Test data

```
# REFER : https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
best_t = find_best_threshold(te_thresholds, test_fpr, test_tpr)
confusion_matrix =metrics.confusion_matrix(y_test,predict_with_best_t(y_test_probs, best_t))
```

```
# REFER :  https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
# REFER :  https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

print("CONFUSION MATRIX OF TEST DATA")
print('\n')
print('==>'*50)
print(confusion_matrix)
sns.heatmap(confusion_matrix, annot=True, fmt='d',cmap='RdGy',annot_kws = {"size":16})
plt.ylabel('True Label',size=12)
plt.xlabel('Predicted Label',size=12)
plt.title('Test Data Confusion Matrix',size=12)
```

```
the maximum value of tpr*(1-fpr) 0.43589578558015635 for threshold 0.88
CONFUSION MATRIX OF TEST DATA


==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>=
[[ 3288  1675]
 [ 9513 18299]]
Text(0.5, 1.0, 'Test Data Confusion Matrix')
```



MULTINOMIAL NAIVE BAYES USING GRID SEARCH CROSS VALIDATION (SET - 2)

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV

model=MultinomialNB(class_prior=[0.5,0.5])
param={'alpha': [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]}
clf=GridSearchCV(model,param,scoring='roc_auc',cv=10,return_train_score=True) # running 10 fold cross validation grid search
clf.fit(X_tr_set_two,y_train) #fitting
```

```
GridSearchCV(cv=10, estimator=MultinomialNB(class_prior=[0.5, 0.5]),
             param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.1, 1, 10, 100,
                                   1000]},
             return_train_score=True, scoring='roc_auc')
```

```
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']
```

```
print('Best score: ',clf.best_score_)
print('alpha value with best score: ',clf.best_params_)
print('='*40)
```

```
Best score:  0.7006215706152047
alpha value with best score:  {'alpha': 0.1}
========================================
```

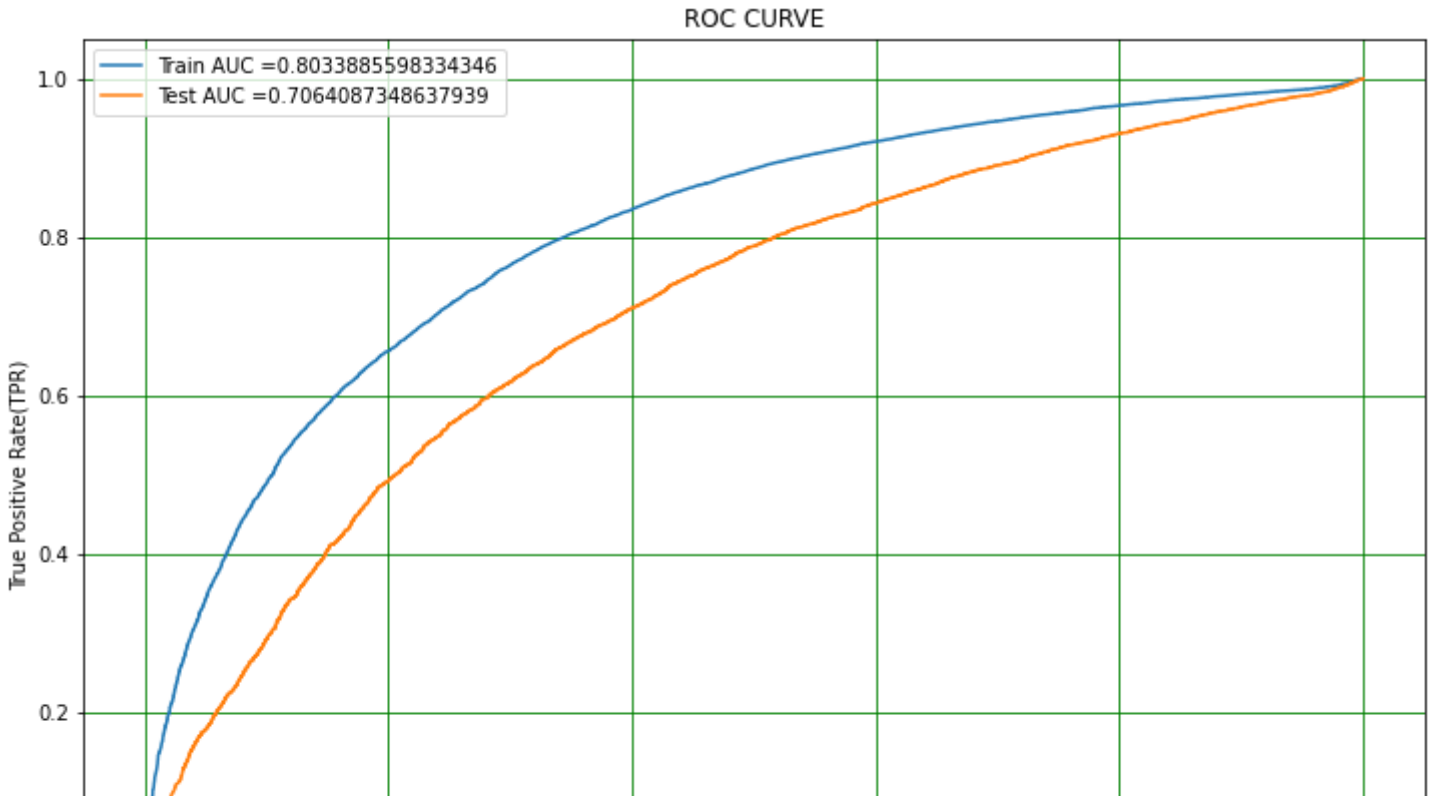
plotting hyperparameter v/s auc

```
log_param=[]
for i in param['alpha']:  # converting alpha into log- alpha
    log_param.append(m.log(i))
```

```
plt.figure(figsize=(12,8))
plt.grid()
plt.plot(log_param, train_auc, label='Train AUC')
plt.plot(log_param, cv_auc, label='CV AUC')
plt.scatter(log_param,train_auc)
plt.scatter(log_param,cv_auc)
plt.legend()
plt.xlabel("log alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("alpha: hyperparameter v/s AUC")
plt.grid(color='red', linestyle='-', linewidth=0.5)
```

roc plot of train and test data

```
model_set2=MultinomialNB(alpha=1e-05,class_prior=[0.5,0.5])
model_set2.fit(X_tr_set_two,y_train)

# converting train and test output into probability
y_train_probs = clf.predict_proba(X_tr_set_two)[:,1] # converting train and test output into probability
y_test_probs= clf.predict_proba(X_te_set_two )[:,1]

 # storing values of fpr and tpr
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_probs) # storing values of fpr and tpr
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_probs)

# PLOTING THE ROC CURVE
plt.figure(figsize=(12,8))
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()

# NOW WE LABEL THE PLOT X AND Y
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")

# TITLE OF THE CURVE IS ROC CURVE
plt.title("ROC CURVE")
plt.grid(color='green',lw=0.8)
```



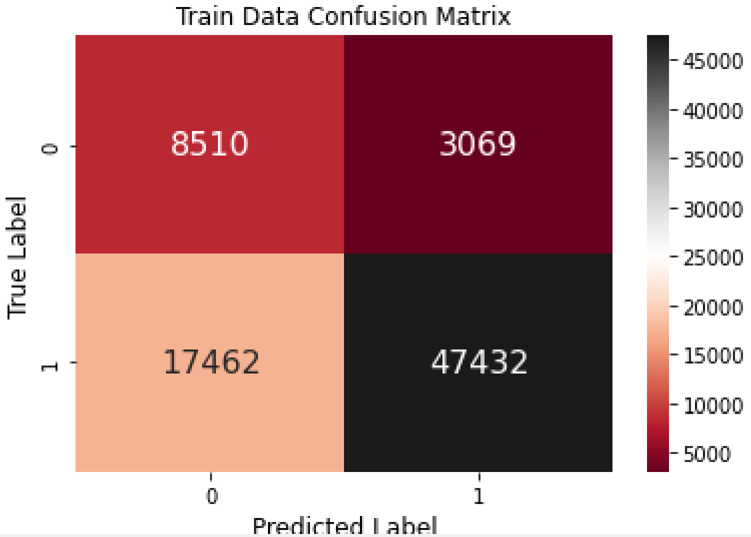
Confusion matrix

Train data

```
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
CONFUSION_MATRIX=metrics.confusion_matrix(y_train,predict_with_best_t(y_train_probs, best_t))


# REFER :  https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
# REFER :  https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

print("CONFUSION MATRIX OF TRAIN DATA")
print("\n")
```

```
print('==>'*50)
print(CONFUSION_MATRIX)
sns.heatmap(CONFUSION_MATRIX, annot=True, fmt='d',cmap='RdGy',annot_kws = {"size":16})
plt.ylabel('True Label',size=12)
plt.xlabel('Predicted Label',size=12)
plt.title('Train Data Confusion Matrix',size=12)
```

```
the maximum value of tpr*(1-fpr) 0.5371868823700227 for threshold 0.577
CONFUSION MATRIX OF TRAIN DATA


==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>=
[[ 8510  3069]
 [17462 47432]]
Text(0.5, 1.0, 'Train Data Confusion Matrix')
```

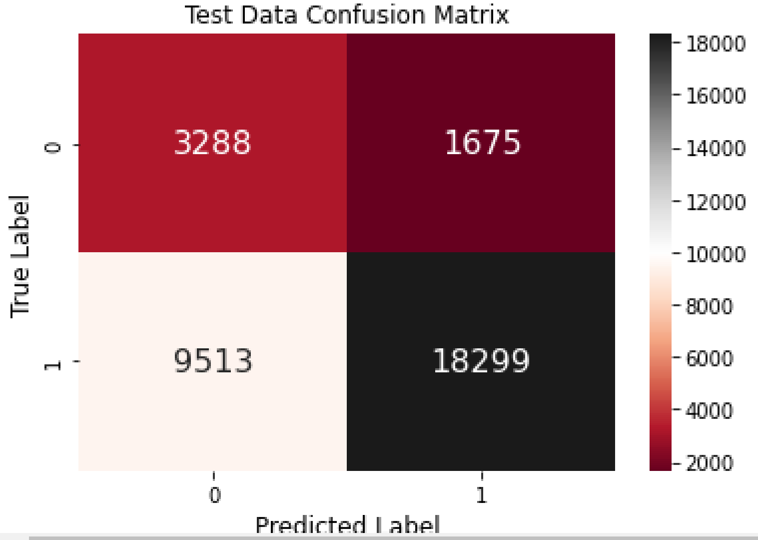


Test data

```
# REFER : https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
best_t = find_best_threshold(te_thresholds, test_fpr, test_tpr)
confusion_matrix =metrics.confusion_matrix(y_test,predict_with_best_t(y_test_probs, best_t))
```

```
# REFER :  https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
# REFER :  https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

print("CONFUSION MATRIX OF TEST DATA")
print('\n')
print('==>'*50)
print(confusion_matrix)
sns.heatmap(confusion_matrix, annot=True, fmt='d',cmap='RdGy',annot_kws = {"size":16})
plt.ylabel('True Label',size=12)
plt.xlabel('Predicted Label',size=12)
plt.title('Test Data Confusion Matrix',size=12)
```

```
the maximum value of tpr*(1-fpr) 0.43589578558015635 for threshold 0.88
CONFUSION MATRIX OF TEST DATA


==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>=
[[ 3288  1675]
 [ 9513 18299]]
Text(0.5, 1.0, 'Test Data Confusion Matrix')
```



Top 20 features from set-1

```
from scipy.sparse import hstack


X_tr_set_one = hstack((X_train_essay_bow,
                       X_train_state_ohe,
                       X_train_teacher_ohe,
                       X_train_grade_ohe,
                       X_train_price,
                       X_train_category_ohe,
                       X_train_subcategory_ohe,
                       X_train_teacher_number_of_previously_posted_projects )).tocsr()

X_te_set_one = hstack((X_test_essay_bow,
                       X_test_state_ohe,
                       X_test_teacher_ohe,
                       X_test_grade_ohe,
                       X_test_price,
                       X_test_category_ohe,
                       X_test_subcategory_ohe,
                       X_test_teacher_number_of_previously_posted_projects)).tocsr()



print("SHAPE OF TRAIN AND TEST AFTER STACKING")
print(X_tr_set_one.shape)
print(X_te_set_one.shape)
```

```
SHAPE OF TRAIN AND TEST AFTER STACKING
(76473, 50101)
(32775, 50101)
```

```
features=[]
for fe in vectorizer_bow.get_feature_names() :
    features.append(fe)

for fe in vectorizer_school_state.get_feature_names() :
    features.append(fe)

for fe in vectorizer_teacher_prefix.get_feature_names() :  # adding all features into list as the order of data frame
    features.append(fe)

for fe in vectorizer_project_grade_category.get_feature_names() :
    features.append(fe)

features.append("price")

for fe in vectorizer_clean_categories.get_feature_names() :
    features.append(fe)

for fe in vectorizer_clean_subcategories.get_feature_names() :
    features.append(fe)

features.append("teacher_number_of_previously_posted_projects")


class_0=model_set1.feature_log_prob_[0, :].argsort()   # finding probability and making argsort for each class
class_1=model_set1.feature_log_prob_[1, :].argsort()
#https://stackoverflow.com/questions/50526898/how-to-get-feature-importance-in-naive-bayes


print("top 20 features of class_0")
print("==>"*100)
print(np.take(features, class_0[-20:])) # since argsort is asscending order
```

```
top 20 features of class_0
==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>=
['able' 'reading' 'love' 'come' 'work' 'need' 'we' 'many' 'nannan'
 'my students' 'the' 'help' 'they' 'learn' 'not' 'classroom' 'my'
 'learning' 'school' 'students']
```

```
print("top 20 features of class_1")
print("==>"*100)
print(np.take(features, class_1[-20:]))
```

```
top 20 features of class_1
==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>=
['day' 'love' 'use' 'reading' 'work' 'need' 'we' 'nannan' 'many' 'help'
 'learn' 'my students' 'they' 'not' 'the' 'classroom' 'learning' 'my'
 'school' 'students']
```

Summary

```
# refer: https://ptable.readthedocs.io/en/latest/tutorial.html
# http://zetcode.com/python/prettytable/

# here we are using the pretty table to show our model performence

from prettytable import PrettyTable
table=PrettyTable()
table.field_names = ["Vectorizer", "Model", "Hyper Parameter", "Test-AUC"]
table.add_row(["BOW", "Naive Bayes", 1e-05, 0.70648])
table.add_row(["TFIDF", "Naive Bayes", 1e-05, 0.70648])
print(table)
```

```
+------------+-------------+-----------------+----------+
| Vectorizer |    Model    | Hyper Parameter | Test-AUC |
```

```
+------------+-------------+-----------------+----------+
|    BOW     | Naive Bayes |      1e-05      | 0.70648  |
|   TFIDF    | Naive Bayes |      1e-05      | 0.70648  |
+------------+-------------+-----------------+----------+
```

optional work

Top 20 features from set-2

```
from scipy.sparse import hstack

X_tr_set_two = hstack((X_train_essay_bow,
                       X_train_state_ohe,
                       X_train_teacher_ohe,
                       X_train_grade_ohe,
                       X_train_price,
                       X_train_category_ohe,
                       X_train_subcategory_ohe,
                       X_train_teacher_number_of_previously_posted_projects )).tocsr()

X_te_set_two = hstack((X_test_essay_bow,
                       X_test_state_ohe,
                       X_test_teacher_ohe,
                       X_test_grade_ohe,
                       X_test_price,
                       X_test_category_ohe,
                       X_test_subcategory_ohe,
                       X_test_teacher_number_of_previously_posted_projects)).tocsr()



print("SHAPE OF TRAIN AND TEST AFTER STACKING")
print(X_tr_set_two.shape)
print(X_te_set_two.shape)
```

```
SHAPE OF TRAIN AND TEST AFTER STACKING
(76473, 50101)
(32775, 50101)
```

```
features=[]
for fe in vectorizer_bow.get_feature_names() :
    features.append(fe)

for fe in vectorizer_school_state.get_feature_names() :
    features.append(fe)

for fe in vectorizer_teacher_prefix.get_feature_names() :  # adding all features into list as the order of data frame
    features.append(fe)

for fe in vectorizer_project_grade_category.get_feature_names() :
    features.append(fe)

features.append("price")

for fe in vectorizer_clean_categories.get_feature_names() :
    features.append(fe)

for fe in vectorizer_clean_subcategories.get_feature_names() :
    features.append(fe)

features.append("teacher_number_of_previously_posted_projects")


class_0=model_set2.feature_log_prob_[0, :].argsort()   # finding probability and making argsort for each class
class_1=model_set2.feature_log_prob_[1, :].argsort()
#https://stackoverflow.com/questions/50526898/how-to-get-feature-importance-in-naive-bayes


print("top 20 features of class_0")
print("="*80)
print(np.take(features, class_0[-20:])) # since argsort is asscending order
```

```
top 20 features of class_0
================================================================================
['able' 'reading' 'love' 'come' 'work' 'need' 'we' 'many' 'nannan'
 'my students' 'the' 'help' 'they' 'learn' 'not' 'classroom' 'my'
 'learning' 'school' 'students']
```

```
print("top 20 features of class_1")
print("="*80)
print(np.take(features, class_1[-20:]))
```

```
top 20 features of class_1
================================================================================
['day' 'love' 'use' 'reading' 'work' 'need' 'we' 'nannan' 'many' 'help'
 'learn' 'my students' 'they' 'not' 'the' 'classroom' 'learning' 'my'
 'school' 'students']
```

Top 100 features optional work

```
print("top 100 features of class_0")
print("==>"*100)
print(np.take(features, class_0[-100:])) # since argsort is asscending order
```

```
top 100 features of class_0
==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>=
['excited' 'experience' 'lunch' 'special' 'using' 'access' 'mathematics'
 'see' 'literacy' 'working' 'resources' 'low' 'in' 'hard' 'many students'
 'students come' 'first' 'environment' 'give' 'fun' 'opportunity' 'free'
 'group' 'teacher' 'well' 'home' 'language' 'community' 'hands' 'learners'
 'read' 'grades_3_5' 'activities' 'way' 'kids' 'it' 'life' 'needs' 'like'
 'create' 'high' 'science' 'allow' 'ms' 'education' 'supplies' 'books'
 'teach' 'best' 'different' 'world' 'math_science' 'every' 'project'
 'grades_prek_2' 'provide' 'literacy_language' 'math' 'get' 'grade'
 'children' 'time' 'one' 'mrs' 'technology' 'student' 'these' 'also'
 'would' 'make' 'new' 'this' 'want' 'year' 'class' 'our' 'use' 'skills'
 'day' 'materials' 'able' 'reading' 'love' 'come' 'work' 'need' 'we'
```

```
 'many' 'nannan' 'my students' 'the' 'help' 'they' 'learn' 'not'
 'classroom' 'my' 'learning' 'school' 'students']
```

top 100 features optional

```
print("top 100 features of class_1")
print("==>"*100)
print(np.take(features, class_1[-100:]))
```

```
top 100 features of class_1
==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>=
['hands' 'excited' 'english' 'great' 'environment' 'mathematics' 'level'
 'take' 'see' 'small' 'lunch' 'fun' 'students come' 'working' 'in'
 'teacher' 'hard' 'opportunity' 'activities' 'community' 'many students'
 'give' 'first' 'life' 'language' 'using' 'well' 'literacy' 'science'
 'access' 'kids' 'free' 'home' 'education' 'needs' 'best' 'create' 'way'
 'grades_3_5' 'group' 'learners' 'it' 'ms' 'like' 'project' 'different'
 'high' 'math_science' 'teach' 'world' 'grades_prek_2' 'materials'
 'provide' 'children' 'every' 'math' 'read' 'allow' 'get'
 'literacy_language' 'these' 'grade' 'one' 'student' 'mrs' 'want' 'time'
 'make' 'year' 'new' 'this' 'skills' 'books' 'also' 'technology' 'our'
 'would' 'class' 'come' 'able' 'day' 'love' 'use' 'reading' 'work' 'need'
 'we' 'nannan' 'many' 'help' 'learn' 'my students' 'they' 'not' 'the'
 'classroom' 'learning' 'my' 'school' 'students']
```

Colab paid products - Cancel contracts here

✓ 0s completed at 9:40 AM