# 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

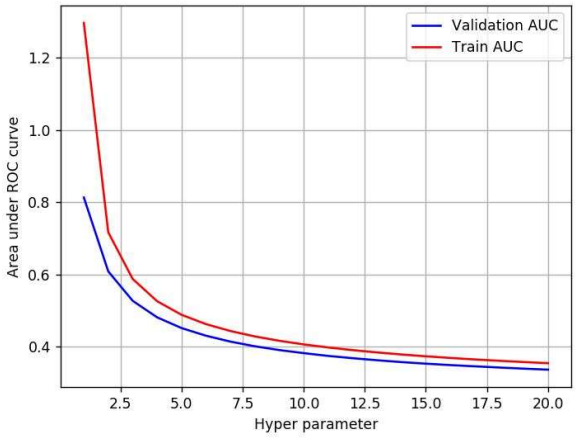
      - teacher_prefix
      - project_grade_category
      - school_state
      - clean_categories
      - clean_subcategories

      numerical features

      - price
      - teacher_number_of_previously_posted_projects

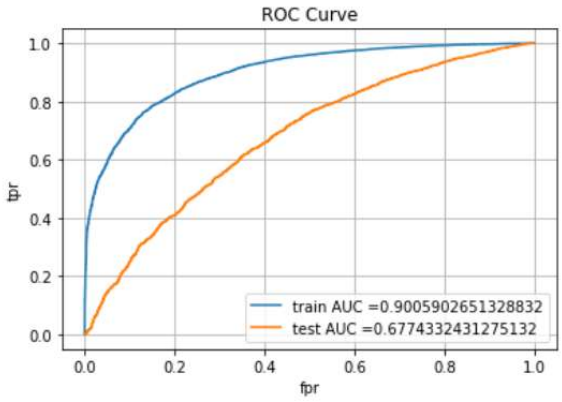
      while encoding the numerical features check this and this
    - Set 1: categorical, numerical features + preprocessed_eassay (BOW)
    - Set 2: categorical, numerical features + preprocessed_eassay (TFIDF)
6. **The hyper paramter tuning(find best alpha:smoothing parameter)**
    - Consider alpha values 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

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

and original labels of test data points

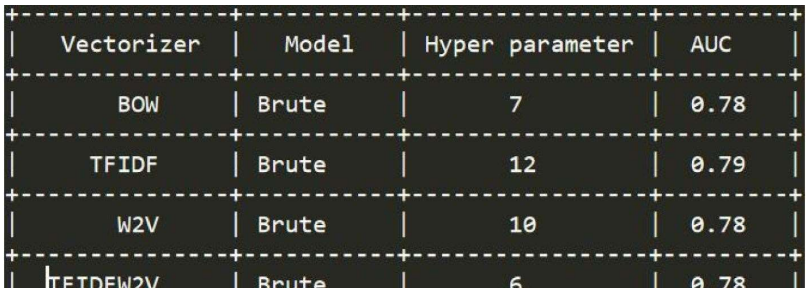
-plot the confusion matrix in heatmaps, while plotting the confusion matrix go through the link

7. find the top 20 features from either from feature Set 1 or feature Set 2 using values of `feature_log_prob_ ` parameter of `MultinomialNB` (https://scikit-

learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html) and print **BOTH** positive as well as negative corresponding feature names.

- go through the link

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



| 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 |

```
pip install chart_studio
```

```
Collecting chart_studio
  Downloading https://files.pythonhosted.org/packages/ca/ce/330794a6b6ca4b9182c38fc69dd
     |████████████████████████████████| 71kB 2.9MB/s
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from char
Requirement already satisfied: plotly in /usr/local/lib/python3.7/dist-packages (from c
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/li
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-pack
Installing collected packages: chart-studio
Successfully installed chart-studio-1.1.0
```

```
%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
```

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

# 2. Naive Bayes

## 1.1 Loading Data

```
import pandas
data = pandas.read_csv('preprocessed_data.csv', nrows=50000)
data.head(5)
```

| | school_state | teacher_prefix | project_grade_category | teacher_number_of_previously_ |
|---|---|---|---|---|

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

| | school_state | teacher_prefix | project_grade_category | teacher_number_of_previously_ |
|---|---|---|---|---|
| **0** | ca | mrs | grades_prek_2 | |

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

```
# 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

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_
```

## 1.3 Make Data Model Ready: encoding eassay, and project_title

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

print("="*100)
```

```
vectorizer_bow = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
```

```
vectorizer_bow = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer_bow.fit(X_train['essay'].values) # fit has to happen only on train data

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

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

```
(22445, 8) (22445,)
(11055, 8) (11055,)
(16500, 8) (16500,)
====================================================================================================
After vectorizations
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
====================================================================================================
```

# 1.4 Make Data Model Ready: encoding numerical, categorical features

## 1.4.1 Encoding categorical features: Teacher prefix

```
vectorizer_teacher = CountVectorizer()
vectorizer_teacher.fit(X_train['teacher_prefix'].values) # fit has to happen only on train da

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

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

```
After vectorizations
(22445, 5) (22445,)
(11055, 5) (11055,)
```

```
(16500, 5) (16500,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
==================================================================================================
```

### 1.4.2 Encoding categorical features: Project grade category

```
vectorizer_grade = CountVectorizer()
vectorizer_grade.fit(X_train['project_grade_category'].values) # fit has to happen only on tr

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer_grade.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vectorizer_grade.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer_grade.transform(X_test['project_grade_category'].values)

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

```
After vectorizations
(22445, 4) (22445,)
(11055, 4) (11055,)
(16500, 4) (16500,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
==================================================================================================
```

### 1.4.3 Encoding categorical features: School State

```
vectorizer_state = CountVectorizer()
vectorizer_state.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_state.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer_state.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer_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(vectorizer_state.get_feature_names())
print("="*100)
```

```
After vectorizations
```

```
(22445, 51) (22445,)
(11055, 51) (11055,)
(16500, 51) (16500,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'i
==================================================================================
```

### 1.4.4 Encoding categorical features: Clean Category

```
vectorizer_clean = CountVectorizer()
vectorizer_clean.fit(X_train['clean_categories'].values) # fit has to happen only on train da

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_ohe = vectorizer_clean.transform(X_train['clean_categories'].values)
X_cv_clean_ohe = vectorizer_clean.transform(X_cv['clean_categories'].values)
X_test_clean_ohe = vectorizer_clean.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_clean_ohe.shape, y_train.shape)
print(X_cv_clean_ohe.shape, y_cv.shape)
print(X_test_clean_ohe.shape, y_test.shape)
print(vectorizer_clean.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 9) (22445,)
(11055, 9) (11055,)
(16500, 9) (16500,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_languag
==================================================================================
```

### 1.4.5 Encoding categorical features: Clean SubCategory

```
vectorizer_clean_sub = CountVectorizer()
vectorizer_clean_sub.fit(X_train['clean_subcategories'].values) # fit has to happen only on t

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_sub_ohe = vectorizer_clean_sub.transform(X_train['clean_subcategories'].values)
X_cv_clean_sub_ohe = vectorizer_clean_sub.transform(X_cv['clean_subcategories'].values)
X_test_clean_sub_ohe = vectorizer_clean_sub.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_clean_sub_ohe.shape, y_train.shape)
print(X_cv_clean_sub_ohe.shape, y_cv.shape)
print(X_test_clean_sub_ohe.shape, y_test.shape)
print(vectorizer_clean_sub.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 30) (22445,)
(11055, 30) (11055,)
(16500, 30) (16500,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_
==================================================================================================
```

## 1.4.6 encoding numerical features: Price

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['price'].values.reshape(1,-1))

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(-1,1))
X_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))
```

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

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
==================================================================================================
```

## 1.4.7 encoding numerical features: teacher_number_of_previously_posted_projects

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
```

```
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))

X_train_teacher_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_proj
X_cv_teacher_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects']
X_test_teacher_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projec



print("After vectorizations")
print(X_train_teacher_norm.shape, y_train.shape)
print(X_cv_teacher_norm.shape, y_cv.shape)
print(X_test_teacher_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

## 1.4.8 Concatinating all the features

```
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_cr = hstack((X_cv_essay_bow, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_clean_c
X_te = hstack((X_test_essay_bow, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_te

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

```
Final Data matrix
(22445, 5101) (22445,)
(11055, 5101) (11055,)
(16500, 5101) (16500,)
====================================================================================================
```

# 1.5 Appling NB on different kind of featurization as mentioned in the instructions

# 1.5.1 Appling NB: BOW featurization

## 1.5.1.1 Hyper parameter Tuning

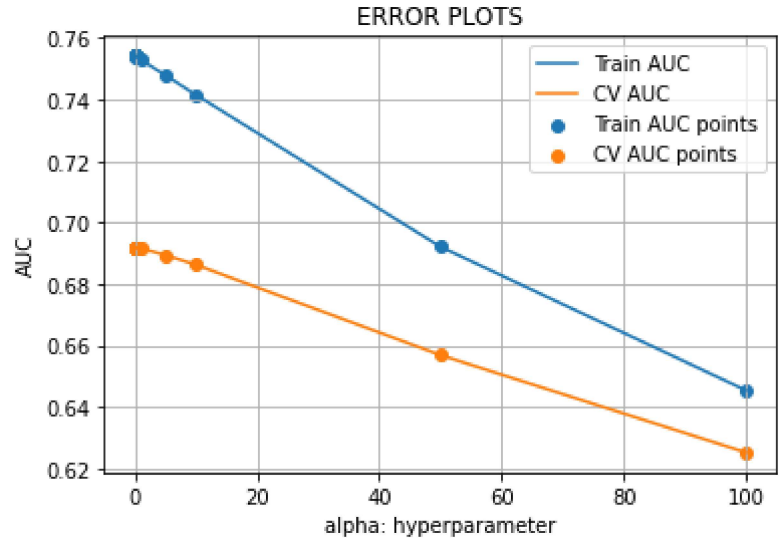
### 1.5.1.1.1 Method 1: Simple for loop

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the
    # 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
```

```
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values,
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
alpha = [0.00001, 0.0005, 0.0001, 0.005, 0.001, 0.05, 0.01, 0.1, 0.5, 1, 5, 10, 50, 100]
```

```
alpha = [0.00001, 0.0005, 0.0001, 0.005, 0.001, 0.05, 0.01, 0.1, 0.5, 1, 5, 10, 50, 100]
for i in tqdm(alpha):
    neigh = MultinomialNB(alpha=i, class_prior=[0.5,0.5])
    neigh.fit(X_tr, y_train)

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

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

plt.plot(alpha, train_auc, label='Train AUC')
plt.plot(alpha, cv_auc, label='CV AUC')

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

plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████| 14/14 [00:01<00:00,  7.77it/s]
```

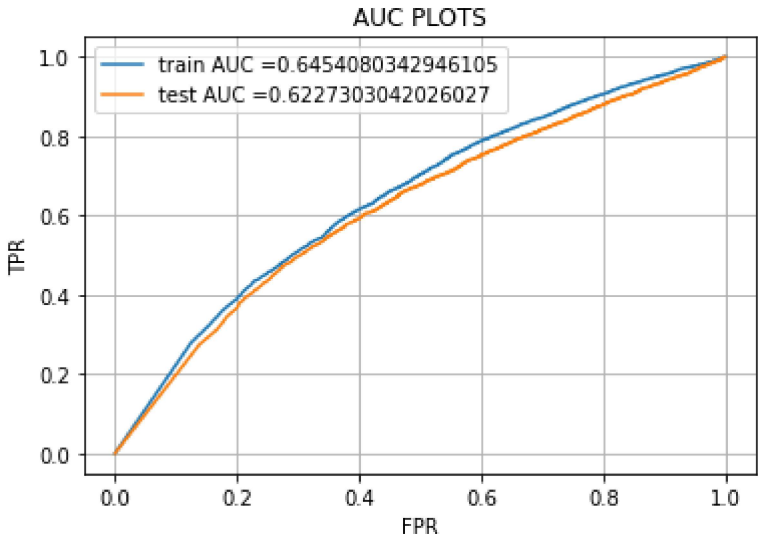



## 1.5.1.1.2 Testing the performance of the model on test data, plotting ROC Curves

```
best_alpha = 100
```

```
from sklearn.metrics import roc_curve, auc
```

```
neigh = MultinomialNB(alpha=best_alpha, class_prior=[0.5,0.5])
neigh.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the pos
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, X_te)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("AUC PLOTS")
plt.grid()
plt.show()
```




```
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

best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print(best_t)
```

```
the maximum value of tpr*(1-fpr) 0.3708092657426503 for threshold 1.0
0.9999999998403837
```

```
def predict_with_best_t(proba, threshould):
```

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

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

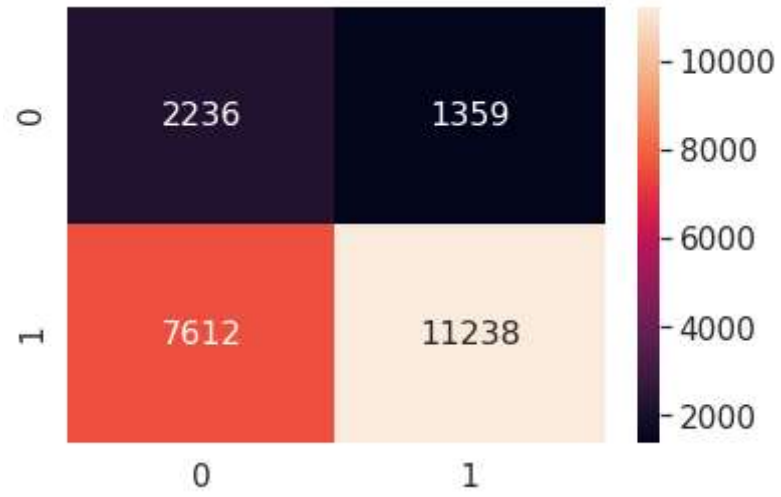
```
Train confusion matrix
[[ 2236  1359]
 [ 7612 11238]]
Test confusion matrix
[[1603 1039]
 [5728 8130]]
```

## Train Confusion Matrix

```
conf_matr_df_train_1 = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(y_train_pre
```

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f22a090ae90>
```




## Test Confusion Matrix

```
conf_matr_df_test_1 = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred,


sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_1, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f22bc66f2d0>
```


## 1.5.2 Appling NB: TFID featurization

```
preprocessed_essays = data['essay'].values


from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
text_tfidf = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (50000, 12122)
```

### 1.5.2.1 Hyper parameter Tuning

```
tfid_vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
tfid_vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfid = tfid_vectorizer.transform(X_train['essay'].values)
X_cv_essay_tfid = tfid_vectorizer.transform(X_cv['essay'].values)
X_test_essay_tfid = tfid_vectorizer.transform(X_test['essay'].values)

print("After tfid vectorizations")
```

```
print("After tfid_vectorizations")
print(X_train_essay_tfid.shape, y_train.shape)
print(X_cv_essay_tfid.shape, y_cv.shape)
print(X_test_essay_tfid.shape, y_test.shape)
print("="*100)
```

```
After tfid_vectorizations
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
====================================================================================================
```

```
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_tfid, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe,
X_cr = hstack((X_cv_essay_tfid, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_clean_
X_te = hstack((X_test_essay_tfid, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_t

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

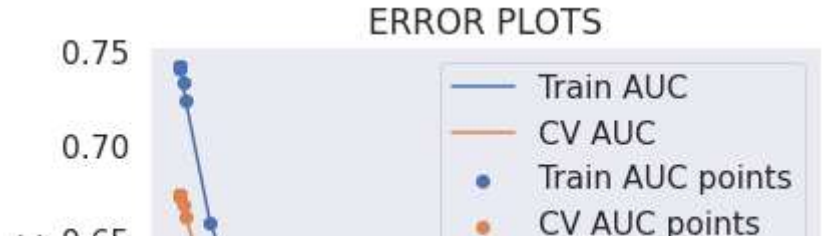
```
Final Data matrix
(22445, 5101) (22445,)
(11055, 5101) (11055,)
(16500, 5101) (16500,)
====================================================================================================
```

### 1.5.2.1.1 Method 1: Simple for loop

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the
    # 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


import matplotlib.pyplot as plt
```

```
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values,
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
alpha = [0.00001, 0.0005, 0.0001, 0.005, 0.001, 0.05, 0.01, 0.1, 0.5, 1, 5, 10, 50, 100]
for i in tqdm(alpha):
    neigh = MultinomialNB(alpha=i, class_prior=[0.5,0.5])
    neigh.fit(X_tr, y_train)

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

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

plt.plot(alpha, train_auc, label='Train AUC')
plt.plot(alpha, cv_auc, label='CV AUC')

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

plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████| 14/14 [00:01<00:00,  7.34it/s]
```




### 1.5.2.1.2 Testing the performance of the model on test data, plotting ROC Curves


```
best_alpha = 100
```

```
from sklearn.metrics import roc_curve, auc


neigh = MultinomialNB(alpha=best_alpha, class_prior=[0.5,0.5])
neigh.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the pos
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, X_te)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("AUC PLOTS")
plt.grid()
plt.show()
```


```
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

best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print(best_t)
```

```
 the maximum value of tpr*(1-fpr) 0.2864217691090263 for threshold 1.0
 0.9998887528535866
```

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


from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
 Train confusion matrix
 [[ 1851  1744]
  [ 8364 10486]]
 Test confusion matrix
 [[1307 1335]
  [6116 7742]]
```

## Train Confusion Matrix

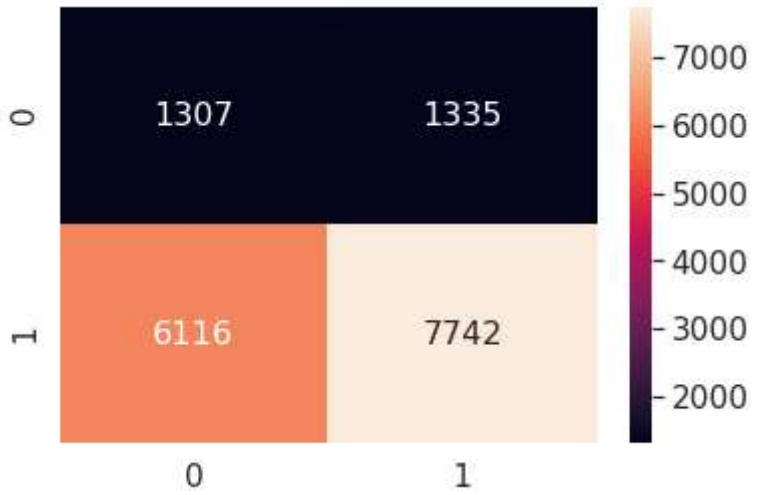
```
conf_matr_df_train_1 = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(y_train_pre


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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f229c060350>
```


## Test Confusion Matrix

```
conf_matr_df_test_1 = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred,

sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_1, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f22b895d3d0>
```




# 1.6 Select best 20 features of both Positive and negative class for both the sets of data

## Set1 Bow

```
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_cr = hstack((X_cv_essay_bow, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_clean_c
X_te = hstack((X_test_essay_bow, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_te



nb_bow = MultinomialNB(alpha = 0.001,class_prior=[0.5,0.5])

nb_bow.fit(X_tr, y_train)


     MultinomialNB(alpha=0.001, class_prior=[0.5, 0.5], fit_prior=True)


# Collecting feature names for BOW set1
# adding to end of list by concatening features
# Code snippet taken from here https://stackabuse.com/append-vs-extend-in-python-lists/


bow_features_names1 = []


for cnt in vectorizer_clean.get_feature_names() :
    bow_features_names1.append(cnt)

for cnt1 in vectorizer_clean_sub.get_feature_names() :
    bow_features_names1.append(cnt1)

for cnt2 in vectorizer_state.get_feature_names() :
    bow_features_names1.append(cnt2)

for cnt3 in vectorizer_grade.get_feature_names() :
    bow_features_names1.append(cnt3)

for cnt4 in vectorizer_teacher.get_feature_names() :
    bow_features_names1.append(cnt4)



for cnt6 in vectorizer_bow.get_feature_names() :
    bow_features_names1.append(cnt6)




bow_features_names1.append("price")

bow_features_names1.append("quantity")


bow_features_names1.append("prev_proposed_projects")

bow_features_names1.append("title_word_count")
```

```
bow_features_names1.append("title_word_count")

bow_features_names1.append("essay_word_count")

len(bow_features_names1)
```

```
5104
```

## Top 20 Positive features BOW

```
pos_class_prob_sorted = nb_bow.feature_log_prob_[1,:].argsort()[::-1][:5070]
len(pos_class_prob_sorted)
for i in pos_class_prob_sorted[:20]:
    print(bow_features_names1[i])
```

```
spend
relax
meetings
knit
cases
supports
my students special
technological
it not
mind
graders they
price
love
motivated learn
project improve
we
usually
movement
time students
continuously
```

## Top 20 Negative features BOW

```
neg_class_prob_sorted = nb_bow.feature_log_prob_[0, :].argsort()[::-1][:7206]
for i in neg_class_prob_sorted[0:20]:
    print(bow_features_names1[i])
```

```
spend
relax
knit
meetings
```

```
cases
my students special
it not
technological
graders they
supports
mind
price
motivated learn
love
movement
usually
we
class would
wide
wy
```

## 1.7 SUMMARY

Finally results are summarized as we can see in below table. Words like learn is present in negative class but not in positive class Few words are similar but their relative ordering is different between the two sets

```
from prettytable import PrettyTable

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

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

x.add_row(["BOW", "Naive Bayes", 100, 0.64])
x.add_row(["TFIDF", "Naive Bayes", 100, 0.54])

print(x)
```

```
+------------+-------------+-----------------+-----------+
| Vectorizer |    Model    | Hyper Parameter |  Test AUC |
+------------+-------------+-----------------+-----------+
|    BOW     | Naive Bayes |       100       |    0.64   |
|   TFIDF    | Naive Bayes |       100       |    0.54   |
+------------+-------------+-----------------+-----------+
```

We conclude that Naive bayes gives better AUC than KNN also it is very fast as compared to KNN.

Naive Bayes is super interpretable because of probability values, we can get feature importance very easily as seen above

There is strong possibility that Naive bayes can overfit if alpha has not been found properly.