Assignment 6: Apply NB

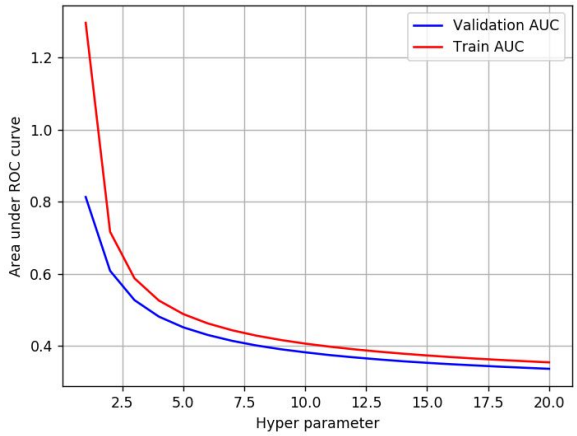
1. Minimum data points need to be considered for people having 4GB RAM is **50k** and for 8GB RAM is **100k**
2. When you are using ramdomsearchcv or gridsearchcv you need not split the data into X_train,X_cv,X_test. As the above methods use kfold. The model will learn better if train data is more so 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

      numerical features
      - price
      - teacher_number_of_previously_posted_projects
      while encoding the numerical features check this and this

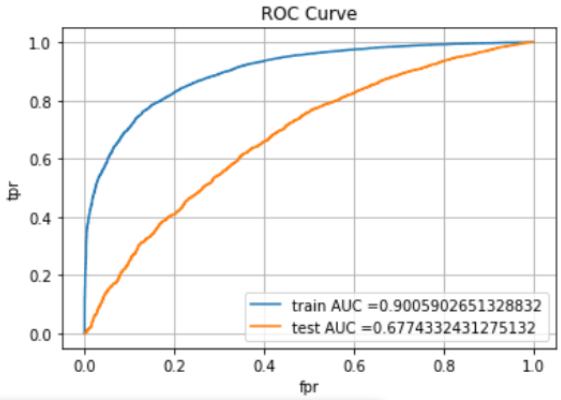
    - Set 1: categorical, numerical features + preprocessed_eassay (BOW)
    - Set 2: categorical, numerical features + preprocessed_eassay (TFIDF)
6. **The hyper paramter tuning(find best alpha:smoothing parameter)**
    - Consider alpha values in range: 10^-5 to 10^2 like [0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,100]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



Saved successfully!

ve, 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 format

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

2. Naive Bayes

1.1 Loading Data

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
```

Saved successfully!

```
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.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
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
import os
```

```
from google.colab import files
files=files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving train_data.csv to train_data.csv

```
from google.colab import files
files=files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving resources.csv to resources.csv

```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

```
n data", project_data.shape)
```

Saved successfully!

```
oject_data.columns.values)
project_data.head(2)
```

```
print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

```
Number of data points in train data (1541272, 4)
['id' 'description' 'quantity' 'price']
```

	id	description	quantity	price
0	p233245	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.00
1	p069063	Bouncy Bands for Desks (Blue support pipes)	3	14.95

1. Preprocessing: "essay"

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase


# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're",
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'hi
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', '
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', '
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over'
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any',
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', '
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'd
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn'
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn
            'won', "won't", 'wouldn', "wouldn't"]
```

Saved successfully!

```
from tqdm import tqdm
def preprocess_text(text_data):
    preprocessed_text = []
    # tqdm is for printing the status bar
    for sentance in tqdm(text_data):
        sent = decontracted(sentance)
        sent = sent.replace('\\r', ' ')
        sent = sent.replace('\\n', ' ')
        sent = sent.replace('\\"', ' ')
        sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
        # https://gist.github.com/sebleier/554280
        sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
        preprocessed_text.append(sent.lower().strip())
    return preprocessed_text


# merge two column text dataframe:
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)


print("printing some random essay")
print(15, project_data['essay'].values[15])
print('-'*50)
print(34, project_data['essay'].values[34])
print('-'*50)
print(147, project_data['essay'].values[147])
```

Saved successfully!

```
tudents face several challenges both in and out of the
--------------------------------------------------
34 My students mainly come from extremely low-income families, and the majority of th
--------------------------------------------------
147 My students are eager to learn and make their mark on the world.\r\n\r\nThey come
```

```
preprocessed_essays = preprocess_text(project_data['essay'].values)
```

```
100%|██████████| 109248/109248 [01:10<00:00, 1557.78it/s]
```

```
print("printing some random essay")
print(9, preprocessed_essays[9])
print('-'*50)
print(34, preprocessed_essays[34])
print('-'*50)
print(147, preprocessed_essays[147])
```

```
printing some random essay
9 95 students free reduced lunch homeless despite come school eagerness learn student
--------------------------------------------------
34 students mainly come extremely low income families majority come homes parents wor
```

```
--------------------------------------------------
147 students eager learn make mark world come title 1 school need extra love fourth g
```

```
project_data["essay"]=preprocessed_essays
```

2. Preprocessing Categorical Features: "teacher_prefix"

```
project_data['teacher_prefix'].value_counts()
```

```
Mrs.       57269
Ms.        38955
Mr.        10648
Teacher     2360
Dr.           13
Name: teacher_prefix, dtype: int64
```

```
# check if we have any nan values are there
print(project_data['teacher_prefix'].isnull().values.any())
print("number of nan values",project_data['teacher_prefix'].isnull().values.sum())
```

```
True
number of nan values 3
```

number of missing values are very less in number, we can replace it with Mrs.

mitted by Mrs.

Saved successfully!

```
project_data['teacher_prefix']=project_data['teacher_prefix'].fillna('Mrs.')
```

```
project_data['teacher_prefix'].value_counts()
```

```
Mrs.       57272
Ms.        38955
Mr.        10648
Teacher     2360
Dr.           13
Name: teacher_prefix, dtype: int64
```

```
#Remove '.' & convert all the chars to small
project_data['teacher_prefix'] = project_data['teacher_prefix'].str.replace('.','')
project_data['teacher_prefix'] = project_data['teacher_prefix'].str.lower()
project_data['teacher_prefix'].value_counts()
```

```
mrs        57272
ms         38955
mr         10648
teacher     2360
dr            13
Name: teacher_prefix, dtype: int64
```

3. Preprocessing Categorical Features: "project_title"

```
project_data['project_title'].head(5)
```

```
0     Educational Support for English Learners at Home
1                Wanted: Projector for Hungry Learners
2    Soccer Equipment for AWESOME Middle School Stu...
3                              Techie Kindergarteners
4                              Interactive Math Tools
Name: project_title, dtype: object
```

```
print("printing some random reviews")
print(9, project_data['project_title'].values[9])
print(34, project_data['project_title'].values[34])
print(147, project_data['project_title'].values[147])
```

```
printing some random reviews
9 Just For the Love of Reading--\r\nPure Pleasure
34 \"Have A Ball!!!\"
147 Who needs a Chromebook?\r\nWE DO!!
```

```
preprocessed_titles = preprocess_text(project_data['project_title'].values)
```

```
100%|██████████| 109248/109248 [00:03<00:00, 34161.47it/s]
```

Saved successfully!

```
print("printing some random reviews")
print(9, preprocessed_titles[9])
print(34, preprocessed_titles[34])
print(147, preprocessed_titles[147])
```

```
printing some random reviews
9 love reading pure pleasure
34 ball
147 needs chromebook
```

```
project_data['project_title']=preprocessed_titles
```

4. Preprocessing Categorical Features: "project_grade_category"

```
project_data['project_grade_category'].value_counts()
```

```
Grades PreK-2    44225
Grades 3-5       37137
Grades 6-8       16923
Grades 9-12      10963
Name: project_grade_category, dtype: int64
```

```
# we need to remove the spaces, replace the '-' with '_' and convert all the letters to sm
# https://stackoverflow.com/questions/36383821/pandas-dataframe-apply-function-to-column-s

project_data['project_grade_category'] = project_data['project_grade_category'].str.replac
project_data['project_grade_category'] = project_data['project_grade_category'].str.replac
project_data['project_grade_category'] = project_data['project_grade_category'].str.lower(
project_data['project_grade_category'].value_counts()
```

```
grades_prek_2    44225
grades_3_5       37137
grades_6_8       16923
grades_9_12      10963
Name: project_grade_category, dtype: int64
```

```
#Rename the column:
project_data.rename(columns={"project_grade_category":"cleaned_project_grade_category"},in
```

5. Preprocessing Categorical Features: "school_state"

```
project_data['school_state'].value_counts()
```

```
CA    15388
TX     7396
NY     7318
FL     6185
SC     3936
MI     3161
PA     3109
IN     2620
MO     2576
OH     2467
LA     2394
MA     2389
WA     2334
OK     2276
NJ     2237
AZ     2147
VA     2045
WI     1827
AL     1762
UT     1731
TN     1688
CT     1663
MD     1514
NV     1367
MS     1323
KY     1304
OR     1242
MN     1208
CO     1111
```

Saved successfully!

```
AR     1049
ID      693
IA      666
KS      634
NM      557
DC      516
HI      507
ME      505
WV      503
NH      348
AK      345
DE      343
NE      309
SD      300
RI      285
MT      245
ND      143
WY       98
VT       80
Name: school_state, dtype: int64
```

```
#convert all of them into small letters
project_data['school_state'] = project_data['school_state'].str.lower()
project_data['school_state'].value_counts()
```

```
ca    15388
tx     7396
ny     7318
fl     6185
nc     5091
il     4350
ga     3963
in     2620
mo     2576
oh     2467
la     2394
ma     2389
wa     2334
ok     2276
nj     2237
az     2147
va     2045
wi     1827
al     1762
ut     1731
tn     1688
ct     1663
md     1514
nv     1367
ms     1323
ky     1304
or     1242
mn     1208
co     1111
ar     1049
id      693
ia      666
```

Saved successfully!

```
ks      634
nm      557
dc      516
hi      507
me      505
wv      503
nh      348
ak      345
de      343
ne      309
sd      300
ri      285
mt      245
nd      143
wy       98
vt       80
Name: school_state, dtype: int64
```

6. Preprocessing Categorical Features: "clean_categories"

```
#Preprocessing Categorical Features: "clean_categories"--"project_subject_categories"
project_data['project_subject_categories'].value_counts()
```

```
Literacy & Language                           23655
Math & Science                                17072
Literacy & Language, Math & Science           14636
Health & Sports                               10177
Music & The Arts                               5180
Special Needs                                  4226
Literacy & Language, Special Needs             3961
                                               3771
                                    uage       2289
Applied Learning, Literacy & Language          2191
History & Civics                               1851
Math & Science, Special Needs                  1840
Literacy & Language, Music & The Arts          1757
Math & Science, Music & The Arts               1642
Applied Learning, Special Needs                1467
History & Civics, Literacy & Language          1421
Health & Sports, Special Needs                 1391
Warmth, Care & Hunger                          1309
Math & Science, Applied Learning               1220
Applied Learning, Math & Science               1052
Literacy & Language, History & Civics           809
Health & Sports, Literacy & Language            803
Applied Learning, Music & The Arts              758
Math & Science, History & Civics                652
Literacy & Language, Applied Learning           636
Applied Learning, Health & Sports               608
Math & Science, Health & Sports                 414
History & Civics, Math & Science                322
History & Civics, Music & The Arts              312
Special Needs, Music & The Arts                 302
Health & Sports, Math & Science                 271
History & Civics, Special Needs                 252
Health & Sports, Applied Learning               192
Applied Learning, History & Civics              178
Health & Sports, Music & The Arts               155
```

Saved successfully!

```
Music & The Arts, Special Needs                 138
Literacy & Language, Health & Sports             72
Health & Sports, History & Civics                43
Special Needs, Health & Sports                   42
History & Civics, Applied Learning               42
Special Needs, Warmth, Care & Hunger             23
Health & Sports, Warmth, Care & Hunger           23
Music & The Arts, Health & Sports                19
Music & The Arts, History & Civics               18
History & Civics, Health & Sports                13
Math & Science, Warmth, Care & Hunger            11
Applied Learning, Warmth, Care & Hunger          10
Music & The Arts, Applied Learning               10
Literacy & Language, Warmth, Care & Hunger        9
Music & The Arts, Warmth, Care & Hunger           2
History & Civics, Warmth, Care & Hunger           1
Name: project_subject_categories, dtype: int64
```

```
#remove spaces, 'the' & replace '&' with '_', and ',' with '_'
project_data['project_subject_categories'] = project_data['project_subject_categories'].st
project_data['project_subject_categories'] = project_data['project_subject_categories'].st
project_data['project_subject_categories'] = project_data['project_subject_categories'].st
project_data['project_subject_categories'] = project_data['project_subject_categories'].st
project_data['project_subject_categories'] = project_data['project_subject_categories'].st
project_data['project_subject_categories'].value_counts()
```

```
literacy_language                       23655
math_science                            17072
literacy_language_math_science          14636
health_sports                           10177
music_arts                               5180
                                         4226
                                         3961
appliedlearning                          3771
math_science_literacy_language           2289
appliedlearning_literacy_language        2191
history_civics                           1851
math_science_specialneeds                1840
literacy_language_music_arts             1757
math_science_music_arts                  1642
appliedlearning_specialneeds             1467
history_civics_literacy_language         1421
health_sports_specialneeds               1391
warmth_care_hunger                       1309
math_science_appliedlearning             1220
appliedlearning_math_science             1052
literacy_language_history_civics          809
health_sports_literacy_language           803
appliedlearning_music_arts                758
math_science_history_civics               652
literacy_language_appliedlearning         636
appliedlearning_health_sports             608
math_science_health_sports                414
history_civics_math_science               322
history_civics_music_arts                 312
specialneeds_music_arts                   302
health_sports_math_science                271
history_civics_specialneeds               252
```

Saved successfully!

```
health_sports_appliedlearning             192
appliedlearning_history_civics            178
health_sports_music_arts                  155
music_arts_specialneeds                   138
literacy_language_health_sports            72
health_sports_history_civics               43
history_civics_appliedlearning             42
specialneeds_health_sports                 42
specialneeds_warmth_care_hunger            23
health_sports_warmth_care_hunger           23
music_arts_health_sports                   19
music_arts_history_civics                  18
history_civics_health_sports               13
math_science_warmth_care_hunger            11
music_arts_appliedlearning                 10
appliedlearning_warmth_care_hunger         10
literacy_language_warmth_care_hunger        9
music_arts_warmth_care_hunger               2
history_civics_warmth_care_hunger           1
Name: project_subject_categories, dtype: int64
```

```
#Rename the "project_subject_categories" to "clean_categories"

project_data.rename(columns={"project_subject_categories":"cleaned_project_subject_categor
```

7. Preprocessing Categorical Features: "clean_subcategories"

```
#Preprocessing Categorical Features: "clean_subcategories"--"project_subject_subcategories
                                     egories'].value_counts()
```

Saved successfully!

```
Literacy                                  9486
Literacy, Mathematics                     8325
Literature & Writing, Mathematics         5923
Literacy, Literature & Writing            5571
Mathematics                               5379
                                          ...
Community Service, Financial Literacy        1
Extracurricular, Financial Literacy           1
Gym & Fitness, Social Sciences                1
College & Career Prep, Warmth, Care & Hunger  1
Gym & Fitness, Parent Involvement             1
Name: project_subject_subcategories, Length: 401, dtype: int64
```

```
project_data['project_subject_subcategories'] = project_data['project_subject_subcategorie
project_data['project_subject_subcategories'] = project_data['project_subject_subcategorie
project_data['project_subject_subcategories'] = project_data['project_subject_subcategorie
project_data['project_subject_subcategories'] = project_data['project_subject_subcategorie
project_data['project_subject_subcategories'] = project_data['project_subject_subcategorie
project_data['project_subject_subcategories'].value_counts()
```

```
literacy                              9486
literacy_mathematics                  8325
literature_writing_mathematics        5923
literacy_literature_writing           5571
```

```
mathematics                                5379
                                           ...
economics_other                               1
esl_teamsports                                1
civics_government_nutritioneducation          1
gym_fitness_socialsciences                    1
financialliteracy_foreignlanguages            1
Name: project_subject_subcategories, Length: 401, dtype: int64
```

```
#Rename the "project_subject_subcategories" to "clean_subcategories"
project_data.rename(columns={"project_subject_subcategories":"cleaned_project_subject_subc
```

1. Preprocessing Numerical Feature: "price", "quantity"

```
# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes-for-all-g
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_inde
price_data.head(2)
```

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21

```
# join two dataframes in python:
                                                    , price_data, on='id', how='left')
```

Saved successfully!

```
project_data.head(2)
```

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	mrs	in
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	mr	fl

1. New Feature Addition: "Number of words present in title"

```
word_count_in_title=[]

for a in project_data["project_title"] :
    b = len(a.split())
    word_count_in_title.append(b)

project_data["word_count_in_title"] = word_count_in_title
```

2. New Feature Addition: Number of words in essay

```
word_count_in_essay=[]

for a in project_data["essay"] :
    b = len(a.split())
    word_count_in_essay.append(b)

project_data["word_count_in_essay"] = word_count_in_essay

project_data.head(2)
```

Saved successfully!

			teacher_id	teacher_prefix	school_state
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	mrs	in
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	mr	fl

```
project_data.columns
```

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'cleaned_project_grade_category',
       'cleaned_project_subject_categories',
```

```
       'cleaned_project_subject_subcategories', 'project_title',
       'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'essay', 'price', 'quantity', 'word_count_in_title',
       'word_count_in_essay'],
      dtype='object')
```

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 subsecti
# 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 co
# 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
project_bkp=project_data.copy()
project_data = project_bkp.sample(n = 50000)
project_bkp.shape
```

```
(50000, 22)
```

```
## taking random samples of 50k datapoints due to lack of computation power
                                = 50000)
```



```
(50000, 22)
```

```
y_value_counts = project_data['project_is_approved'].value_counts()
print("Number of projects thar are approved for funding:     ", y_value_counts[1]," -> ",r
print("Number of projects thar are not approved for funding: ", y_value_counts[0]," -> ",r
```

```
Number of projects thar are approved for funding:      42449  ->  84.9 %
Number of projects thar are not approved for funding:  7551  ->  15.1 %
```

Observation:

Dataset is highly IMBALANCED. Approved Class (1) is the Majority class. And the Majority class portion in our sampled dataset: ~85% Unapproved class (0) is the Minority class. And the Minority class portion in our sampled dataset: ~15%

```
project_data.isna().any()
```

```
Unnamed: 0                                          False
id                                                  False
```

```
teacher_id                                    False
teacher_prefix                                False
school_state                                  False
project_submitted_datetime                    False
cleaned_project_grade_category                False
cleaned_project_subject_categories            False
cleaned_project_subject_subcategories         False
project_title                                 False
project_essay_1                               False
project_essay_2                               False
project_essay_3                                True
project_essay_4                                True
project_resource_summary                      False
teacher_number_of_previously_posted_projects  False
project_is_approved                           False
essay                                         False
price                                         False
quantity                                      False
word_count_in_title                           False
word_count_in_essay                           False
dtype: bool
```

```
y = project_data['project_is_approved'].values
project_data.drop(['project_is_approved'], axis=1, inplace=True)
project_data.head(1)
```

Unnamed: 0	id	teacher_id	teacher_prefix	school_st

Saved successfully!

29f9e2d549546bbc594035b8704 ms

```
X=project_data
```

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

```
# train test split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.25, stratify

print("x_train: ",X_train.shape)
print("x_test : ",X_test.shape)
print("y_train: ",y_train.shape)
print("y_test : ",y_test.shape)
print("x_cv: ",X_cv.shape)
print("y_cv: ",y_cv.shape)
```

```
x_train:  (28125, 21)
x_test :  (12500, 21)
y_train:  (28125,)
y_test :  (12500,)
x_cv:  (9375, 21)
y_cv:  (9375,)
```

1.3 Make Data Model Ready: encoding eassay, and project_title

```
# please write all the code with proper documentation, and proper titles for each subsecti
# 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 co
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label

print(X_train['essay'].values[1])
print(X_test['project_title'].values[2])
```

Saved successfully!

```
d play important necessary kindergarten children free
hear
```

```
#vectorizer = CountVectorizer()
#vectorizer.fit(X_train['essay'])
#print(vectorizer.transform(X_train['essay']).toarray())
#print(vectorizer.get_feature_names())


# For Selecting best feature.. lets create an empty array and add.

best_features=[]


#1.3.1(essay)
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4))
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_bow = vectorizer.transform(X_train['essay'].values)
X_cv_essay_bow = vectorizer.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer.transform(X_test['essay'].values)
```

```
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("="*50)

best_features.extend(vectorizer.get_feature_names())
```

```
After vectorizations
(28125, 82554) (28125,)
(9375, 82554) (9375,)
(12500, 82554) (12500,)
==================================================
```

Encoding "project_title"

```
#1.3.2(project_title)
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4))
vectorizer.fit(X_train['project_title'].values)                                    # fit has

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_bow = vectorizer.transform(X_train['project_title'].values)
X_cv_title_bow = vectorizer.transform(X_cv['project_title'].values)
X_test_title_bow = vectorizer.transform(X_test['project_title'].values)

print("After vectorizations")
                                                    in.shape)
Saved successfully!                                 pe)
                                                    .shape)
print("="*50)

best_features.extend(vectorizer.get_feature_names())
```

```
After vectorizations
(28125, 2089) (28125,)
(9375, 2089) (9375,)
(12500, 2089) (12500,)
==================================================
```

1.4 Make Data Model Ready: encoding numerical, categorical features

```
# please write all the code with proper documentation, and proper titles for each subsecti
# 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 co
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

Make Data Model Ready: Encoding Categorical Features

Encoding "school_state"

```
#1.3.3 (school_state)

vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train data

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

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
                                    pe)
                                    .shape)
                                    )
print("="*100)

best_features.extend(vectorizer.get_feature_names())
```



```
After vectorizations
(28125, 51) (28125,)
(9375, 51) (9375,)
(12500, 51) (12500,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id',
==========
```

Encoding "teacher_prefix":

```
#1.3.4 (teacher_prefix)

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

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

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.get_feature_names())
print("="*100)

best_features.extend(vectorizer.get_feature_names())
```

```
After vectorizations
(28125, 5) (28125,)
(9375, 5) (9375,)
(12500, 5) (12500,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
====================================================================================================
```

Encoding "cleaned_project_grade_category"


```
vectorizer.fit(X_train['cleaned_project_grade_category'].values) # fit has to happen only

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer.transform(X_train['cleaned_project_grade_category'].values)
X_cv_grade_ohe = vectorizer.transform(X_cv['cleaned_project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['cleaned_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.get_feature_names())
print("="*100)

best_features.extend(vectorizer.get_feature_names())
```

```
After vectorizations
(28125, 4) (28125,)
(9375, 4) (9375,)
(12500, 4) (12500,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
====================================================================================================
```

Encoding "cleaned_project_subject_categories"

```
#1.3.6 (cleaned_project_subject_categories)

vectorizer = CountVectorizer()
vectorizer.fit(X_train['cleaned_project_subject_categories'].values) # fit has to happen o

# we use the fitted CountVectorizer to convert the text to vector
X_train_subject_cat = vectorizer.transform(X_train['cleaned_project_subject_categories'].v
X_cv_subject_cat = vectorizer.transform(X_cv['cleaned_project_subject_categories'].values)
X_test_subject_cat = vectorizer.transform(X_test['cleaned_project_subject_categories'].val

print("After vectorizations")
print(X_train_subject_cat.shape, y_train.shape)
print(X_cv_subject_cat.shape, y_cv.shape)
print(X_test_subject_cat.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)

best_features.extend(vectorizer.get_feature_names())
```

```
After vectorizations
(28125, 50) (28125,)
(9375, 50) (9375,)
(12500, 50) (12500,)
['appliedlearning', 'appliedlearning_health_sports', 'appliedlearning_history_civics
====================================================================================
```

Saved successfully!

Encoding "cleaned_project_subject_subcategories"

```
#1.3.7 (cleaned_project_subject_subcategories)

vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(X_train['cleaned_project_subject_subcategories'].values) # fit has to happe

# we use the fitted CountVectorizer to convert the text to vector
X_train_sub_cat = vectorizer.transform(X_train['cleaned_project_subject_subcategories'].va
X_cv_sub_cat = vectorizer.transform(X_cv['cleaned_project_subject_subcategories'].values)
X_test_sub_cat = vectorizer.transform(X_test['cleaned_project_subject_subcategories'].valu

print("After vectorizations")
print(X_train_sub_cat.shape, y_train.shape)
print(X_cv_sub_cat.shape, y_cv.shape)
print(X_test_sub_cat.shape, y_test.shape)
print(vectorizer.get_feature_names())
```

```
print("="*100)

best_features.extend(vectorizer.get_feature_names())
```

```
After vectorizations
(28125, 165) (28125,)
(9375, 165) (9375,)
(12500, 165) (12500,)
['appliedsciences', 'appliedsciences_charactereducation', 'appliedsciences_college_ca
==========================================================================================
```

Make Data Model Ready: Encoding Numerical Features

Encoding "teacher_number_of_previously_posted_projects"

```
#1.4.1 (teacher_number_of_previously_posted_projects)

from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

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

X_train_pre_post_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_
X_cv_pre_post_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projec
X_test_pre_post_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_pr
```

Saved successfully!

```
print(X_train_pre_post_norm.shape, y_train.shape)
print(X_cv_pre_post_norm.shape, y_cv.shape)
print(X_test_pre_post_norm.shape, y_test.shape)
print(X_cv_pre_post_norm)
print("="*100)

best_features.extend(X_train['teacher_number_of_previously_posted_projects'])
```

```
After vectorizations
(28125, 1) (28125,)
(9375, 1) (9375,)
(12500, 1) (12500,)
[[1.]
 [1.]
 [1.]
 ...
 [0.]
 [1.]
 [1.]]
==========================================================================================
```

Encoding "quantity"

```
#1.4.2 (quantity)

from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['quantity'].values)
# 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['quantity'].values.reshape(-1,1))

X_train_quantity_norm = normalizer.transform(X_train['quantity'].values.reshape(-1,1))
X_cv_quantity_norm = normalizer.transform(X_cv['quantity'].values.reshape(-1,1))
X_test_quantity_norm = normalizer.transform(X_test['quantity'].values.reshape(-1,1))



print("After vectorizations")
print(X_train_quantity_norm.shape, y_train.shape)
print(X_cv_quantity_norm.shape, y_cv.shape)
print(X_test_quantity_norm.shape, y_test.shape)
print("="*100)

best_features.extend(X_train['quantity'])
```

```
After vectorizations
(28125, 1) (28125,)
(9375, 1) (9375,)
(12500, 1) (12500,)
====================================================================================================
```

Saved successfully!

Encoding "price"

```
#1.4.3(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)
best_features.extend(X_train['price'])
```

```
After vectorizations
(28125, 1) (28125,)
(9375, 1) (9375,)
(12500, 1) (12500,)
====================================================================================
```

Encoding "word_count_in_title"

```
#1.4.4 (word_count_in_title)

from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

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

X_train_word_count_norm = normalizer.transform(X_train['word_count_in_title'].values.resha
X_cv_word_count_norm = normalizer.transform(X_cv['word_count_in_title'].values.reshape(-1,
X_test_word_count_norm = normalizer.transform(X_test['word_count_in_title'].values.reshape
```



```
print("After vectorizations")
print(X_train_word_count_norm.shape, y_train.shape)
print(X_cv_word_count_norm.shape, y_cv.shape)
print(X_test_word_count_norm.shape, y_test.shape)
print("="*100)
best_features.extend(X_train['word_count_in_title'])
```

```
After vectorizations
(28125, 1) (28125,)
(9375, 1) (9375,)
(12500, 1) (12500,)
====================================================================================
```

Encoding "word_count_in_essay"

```
#1.4.5 (word_count_in_essay)

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

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

X_train_word_count_essay_norm = normalizer.transform(X_train['word_count_in_essay'].values
X_cv_word_count_essay_norm = normalizer.transform(X_cv['word_count_in_essay'].values.resha
X_test_word_count_essay_norm = normalizer.transform(X_test['word_count_in_essay'].values.r


print("After vectorizations")
print(X_train_word_count_essay_norm.shape, y_train.shape)
print(X_cv_word_count_essay_norm.shape, y_cv.shape)
print(X_test_word_count_essay_norm.shape, y_test.shape)
print("="*100)
best_features.extend(X_train['word_count_in_essay'])
```

```
After vectorizations
(28125, 1) (28125,)
(9375, 1) (9375,)
(12500, 1) (12500,)
====================================================================================
```

```
print(X_train_essay_bow.shape)
print(X_train_price_norm.shape)
print(X_cv_essay_bow.shape)
print(X_cv_price_norm.shape)
```

```
(28125, 82554)
(28125, 1)
(9375, 82554)
(9375, 1)
```

Saved successfully!

1.5 Concatinating all the features

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_bow, X_train_title_bow, X_train_state_ohe, X_train_teacher_oh
X_cr = hstack((X_cv_essay_bow, X_cv_title_bow, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grad
X_te = hstack((X_test_essay_bow, X_test_title_bow, X_test_state_ohe, X_test_teacher_ohe, X

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
(28125, 84923) (28125,)
(9375, 84923) (9375,)
(12500, 84923) (12500,)
====================================================================================
```

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

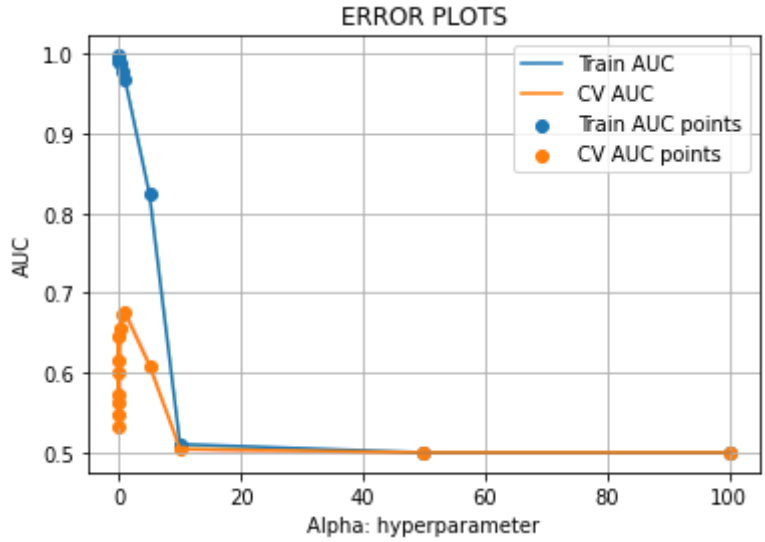
Apply NB on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

Set 1: Categorical, Numerical features + Preprocessed_eassay (BOW)

```
# please write all the code with proper documentation, and proper titles for each subsecti
# 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 co
# 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


# Ref: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearc
                                                    GridSearchCV
                                                    inomialNB

neigh = MultinomialNB()
parameters = {'alpha':[0.00001,0.0005, 0.0001, 0.005, 0.001, 0.05, 0.01, 0.1, 0.5, 1, 5 ,
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc', return_train_score=True) #
clf.fit(X_tr, y_train)

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

plt.plot(parameters['alpha'], train_auc, label='Train AUC') #Ref: https://stackoverflow.co
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc + train_auc

plt.plot(parameters['alpha'], cv_auc, label='CV AUC') #Ref: https://stackoverflow.com/a/48
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0

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


plt.legend()
plt.xlabel("Alpha: hyperparameter")
```

Saved successfully!

```
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

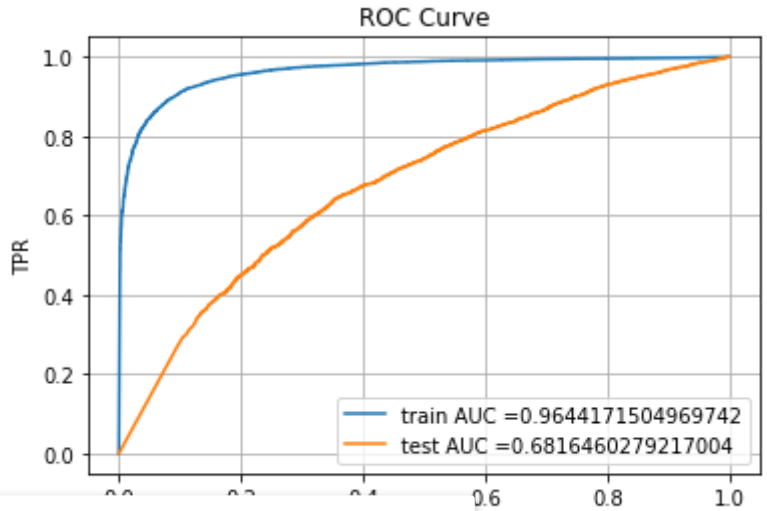



```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider X_tr shape is 28096, then your cr_loop will be 28125 - 28125%1000 = 28096
    # in this for loop we will iterate until the last 1000 multiplier
                                                                    :
                                                                    ct_proba(data[i:i+1000])[:,1])
                                                                    points

    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred


print(X_tr.shape)
print(X_te.shape)
```

```
(28125, 84923)
(12500, 84923)
```

```
#Ref: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#skl
from sklearn.metrics import roc_curve, auc

best_alpha_m1 =0.1
# for i in tqdm(parameters):
naiveBayesClf_set1 = MultinomialNB(alpha=best_alpha_m1, class_prior=[0.5, 0.5] )
naiveBayesClf_set1.fit(X_tr, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
    # not the predicted outputs

y_train_pred = batch_predict(naiveBayesClf_set1, X_tr)
```

```
y_train_pred = batch_predict(naiveBayesClf_set1, X_tr)
y_test_pred = batch_predict(naiveBayesClf_set1, 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)

m1_Auc = str(auc(train_fpr, train_tpr))

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()
```



Saved successfully!

```
def predict(proba, 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(
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    #print(predictions)
    return predictions


from sklearn.metrics import confusion_matrix

print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

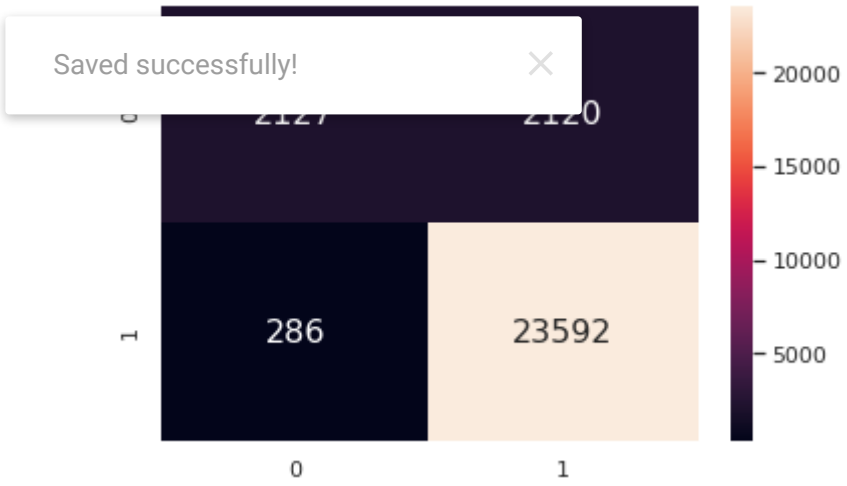
```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499993208408334 for threshold 0.0
[[ 2127  2120]
 [  286 23592]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.509
[[ 618 1270]
 [1572 9040]]
```

```
# https://stackoverflow.com/a/42265865/6000190
def plotConfusionMatrix(cm):
#     cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
#     cm_normalized = cm.astype('int') / 100
    sns.set(font_scale=1)#for label size
#     sns.heatmap(cm, annot=True,annot_kws={"size": 12})# font size
    sns.heatmap(cm, annot = True,annot_kws={"size": 16}, fmt='d')
```

```
# Normalize the confusion matrix by row (i.e by the number of samples in each class)
# https://scikit-learn.org/0.16/auto_examples/model_selection/plot_confusion_matrix.html

cm_train = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train
plotConfusionMatrix(cm_train)
```

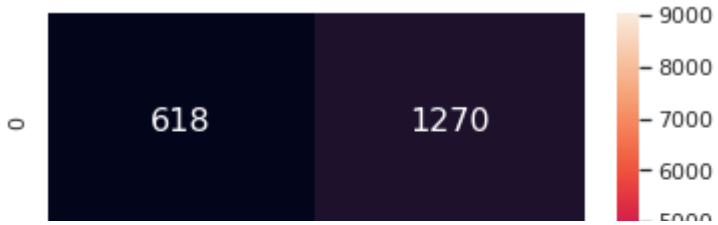
```
# sns.set(font_scale=1.4)#for label size
# sns.heatmap(cm_train, annot=True,annot_kws={"size": 16})# font size
```

```
the maximum value of tpr*(1-fpr) 0.2499993208408334 for threshold 0.0
```




```
cm_test = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)
plotConfusionMatrix(cm_test)
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.509
```



Finding Best Features

```
#Ref: https://imgur.com/mWvE7gj

max_ind_pos=np.argsort((naiveBayesClf_set1.feature_log_prob_)[1])[::1][0:10]
min_ind_pos=np.argsort((naiveBayesClf_set1.feature_log_prob_)[1])[::-1][0:10]
top_pos=np.take(best_features,max_ind_pos)
last_pos=np.take(best_features,min_ind_pos)

max_ind_neg=np.argsort((naiveBayesClf_set1.feature_log_prob_)[0])[::1][0:10]
min_ind_neg=np.argsort((naiveBayesClf_set1.feature_log_prob_)[0])[::-1][0:10]
top_neg=np.take(best_features,max_ind_neg)
last_neg=np.take(best_features,min_ind_neg)

print("")
print("Positive high prob features: ")
print("")
print(top_pos)
print("-"*50)
print("Positive low prob features: ")
```

Saved successfully!

```
print("-"*100)
print("-"*100)
print("-"*100)

print("")
print("Negative high prob features: ")
print("")
print(top_neg)
print("-"*50)
print("Negative low prob features: ")
print("")
print(last_neg)
```

```
Positive high prob features:

['music_arts_warmth_care_hunger' 'literacy_language_warmth_care_hunger'
 'music_arts_appliedlearning' 'appliedlearning_warmth_care_hunger' 'dr'
 'music_arts_history_civics' 'math_science_warmth_care_hunger'
 'help get students supplies' 'must change'
 'students supplies need successful']
--------------------------------------------------
Positive low prob features:
```

```
['students' 'school' 'learning' 'classroom' 'not' 'learn' 'help' '22' '1'
 '2']
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

Negative high prob features:

['make math come' 'school 60 students receiving'
 'individual reading levels' 'individual needs many'
 'school 75 students receiving' 'individual need' 'students fourth fifth'
 'want go college' 'individual copies' 'individual access']
------------------------------------------------
Negative low prob features:

['students' 'school' 'learning' 'classroom' 'not' 'learn' 'help' '22' '2'
 '1']
```

Set 2: Categorical, Numerical Features + preprocessed_eassay (TFIDF)

Saved successfully!

```
IDF:

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizerTfIDF = TfidfVectorizer(min_df=10)

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

print("Shape of matrix after one hot encodig ",X_train_essay_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (28125, 9667)
```

```
X_train_title_tfidf = vectorizerTfIDF.fit_transform(X_train['project_title'].values)
X_cv_title_tfidf = vectorizerTfIDF.transform(X_cv['project_title'].values)
X_test_title_tfidf = vectorizerTfIDF.transform(X_test['project_title'].values)


print("Shape of matrix after one hot encodig ",X_train_title_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (28125, 1359)
```

```
print(X_cv_essay_tfidf.shape)
print(X_test_title_tfidf.shape)
print(X_train_essay_tfidf.shape)
print(X_cv_price_norm.shape)
```

```
(9375, 9667)
(12500, 1359)
(28125, 9667)
(9375, 1)
```

```
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_tfidf, X_train_title_tfidf, X_train_state_ohe, X_train_teache
X_cr = hstack((X_cv_essay_tfidf, X_cv_title_tfidf, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_
X_te = hstack((X_test_essay_tfidf, X_test_title_tfidf, X_test_state_ohe, X_test_teacher_oh

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)
```

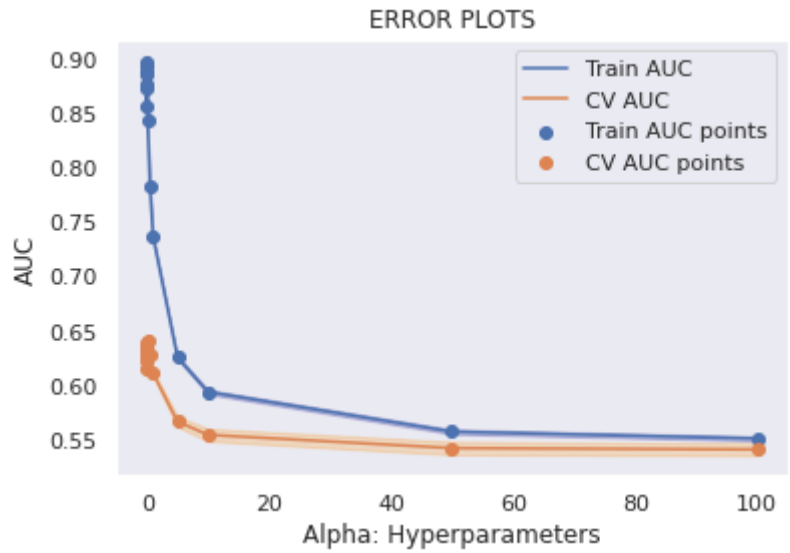
```
Final Data matrix
(28125, 11306) (28125,)
(9375, 11306) (9375,)
(12500, 11306) (12500,)
====================================================================================
```

```
neigh = MultinomialNB()
parameters = {'alpha': [0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,100
                                                    , cv=3, scoring='roc_auc', return_train_score=True ) #
```

Saved successfully!

```
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']

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# Ref: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc + train_auc

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# Ref: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0

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


plt.legend()
plt.xlabel("Alpha: Hyperparameters")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
```

```
plt.show()
```




```
#Extra work for study!!
from sklearn.model_selection import GridSearchCV
mnb_tfidf = MultinomialNB(class_prior=[0.5, 0.5])
parameters = {'alpha': [0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,100
clf = GridSearchCV(mnb_tfidf, parameters, cv= 3, scoring='roc_auc',verbose=1,return_train_
# clf.fit(x_cv_onehot_tfidf, y_cv)
clf.fit(X_tr, y_train)
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
                                                    ]
```

Saved successfully!

```
                                                    _['alpha']," BEST SCORE: ",clf.best_score_)
```

```
Fitting 3 folds for each of 14 candidates, totalling 42 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
BEST ALPHA:  0.1  BEST SCORE:  0.6412432760145467
[Parallel(n_jobs=1)]: Done  42 out of  42 | elapsed:    2.4s finished
```

- Minimum gap represents the best alpha value.

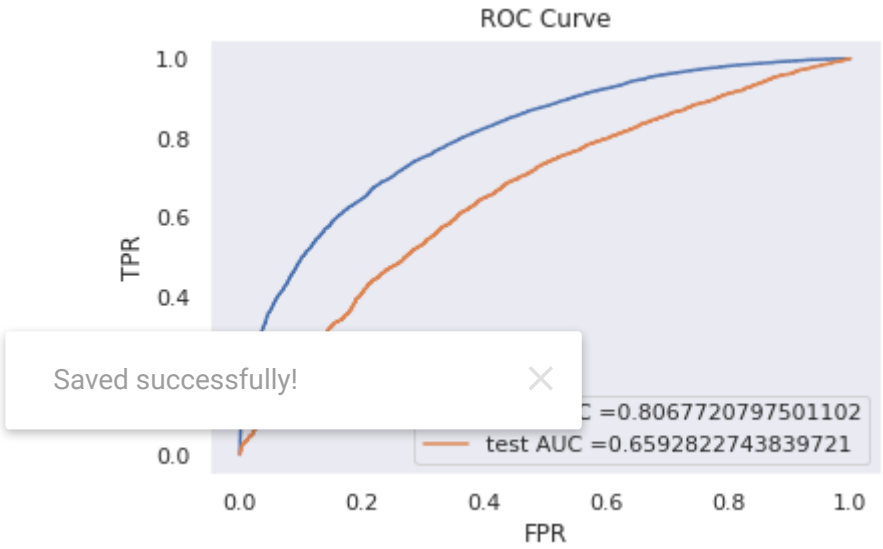
  And in this plot 19 is having less space between Train and Test Auc Graphs

```
print(X_tr.shape)
print(X_te.shape)
print(y_test.shape)
```

```
(28125, 11306)
(12500, 11306)
(12500,)
```

```
best_alpha_m2 = 0.1
# for i in tqdm(parameters):
naiveBayesClf = MultinomialNB(alpha=best_alpha_m2, class_prior=[0.5, 0.5])
```

```
naiveBayesClf.fit(X_tr, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
    # not the predicted outputs

y_train_pred = batch_predict(naiveBayesClf, X_tr)
y_test_pred = batch_predict(naiveBayesClf, 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)

m2_Auc = str(auc(train_fpr, train_tpr))

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()
```

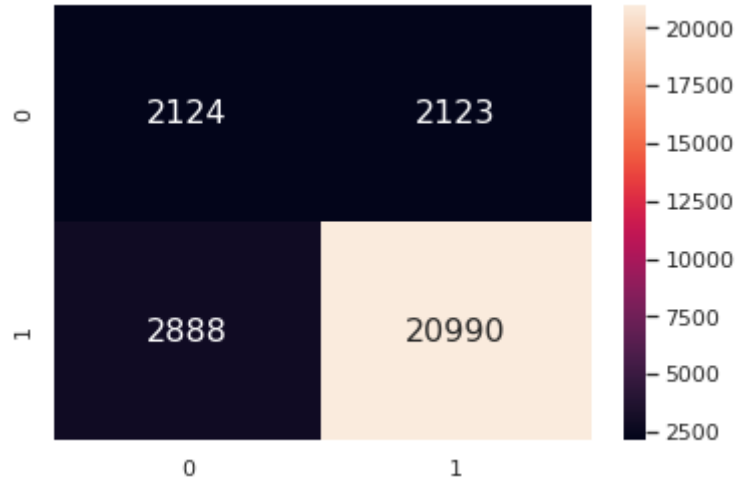



```
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999998613960883 for threshold 0.342
[[ 2124  2123]
 [ 2888 20990]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.556
[[1161  727]
 [3917 6695]]
```

```
cm_train = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train

plotConfusionMatrix(cm_train)
```

```
the maximum value of tpr*(1-fpr) 0.24999998613960883 for threshold 0.342
```




```
cm_test = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)

plotConfusionMatrix(cm_test)
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.556
```

Saved successfully!

```
max_ind_pos=np.argsort((naiveBayesClf.feature_log_prob_)[1])[::1][0:10]
min_ind_pos=np.argsort((naiveBayesClf.feature_log_prob_)[1])[::-1][0:10]
top_pos=np.take(best_features,max_ind_pos)
last_pos=np.take(best_features,min_ind_pos)

max_ind_neg=np.argsort((naiveBayesClf.feature_log_prob_)[0])[::1][0:10]
min_ind_neg=np.argsort((naiveBayesClf.feature_log_prob_)[0])[::-1][0:10]
top_neg=np.take(best_features,max_ind_neg)
last_neg=np.take(best_features,min_ind_neg)

print("")
print("Positive hight prob features: ")
print("")
print(top_pos)
print("="*50)
print("Positive low prob features: ")
print("")
print(last_pos)

print("*100)
```

```
print("="*100)
print("="*100)
print("="*100)

print("")
print("Negative high prob features: ")
print("")
print(top_neg)
print("="*50)
print("Negative low prob features: ")
print("")
print(last_neg)
```

```
Positive hight prob features:

['civil rights movement' 'building project' 'based real' 'able spend time'
 'area student' 'bounds' 'breakfast lunch' 'area students faced' 'adapt'
 'along words']
==================================================
Positive low prob features:

['class fun' 'class full energy' 'class full energetic'
 'class full students' 'class full' 'city elementary school' 'city kids'
 'city get' 'city high' 'city school 100']
====================================================================================================
====================================================================================================
====================================================================================================

Negative high prob features:

['actively engage' 'cheaper' 'building old' 'become clear school'
                                books represent'
                                es' 'abilities school' 'around sitting']
                                ==================
Negative low prob features:

['class fun' 'class full energetic' 'class full energy'
 'class full students' 'class full' 'city elementary school' 'city kids'
 'city get' 'city high' 'city school 100']
```

Saved successfully!

```
# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable


#!pip3 install prettytable

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


x.add_row(["BOW", "Naive Bayes", best_alpha_m1, m1_Auc])
x.add_row([" ", " ", " ", " "])
x.add_row(["TFIDF", "Naive Bayes", best_alpha_m2, m2_Auc])

print(x)
```

```
+------------+-------------+-----------------+--------------------+
| Vectorizer |    Model    | Hyper Parameter |        AUC         |
+------------+-------------+-----------------+--------------------+
|    BOW     | Naive Bayes |       0.1       | 0.9644171504969742 |
|            |             |                 |                    |
|   TFIDF    | Naive Bayes |       0.1       | 0.8067720797501102 |
+------------+-------------+-----------------+--------------------+
```

3. Summary

as mentioned in the step 5 of instructions

Naive Bayes For Donors Choose Dataset:

1) I have taken 50000 Data points for this assignment

2) BOW vectorizer is more efficient than TFIDF vectorizer. (AUC["BOW"]=96.44% & AUC["TFIDF"]=80.67%)

3) I have also found top 20 features from Feature Set 1 and Feature Set 2 using values of `feature_log_prob.

THANK YOU!!