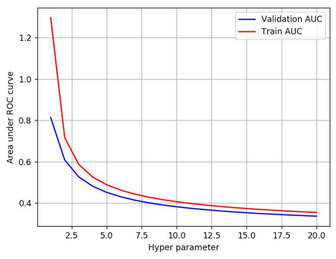
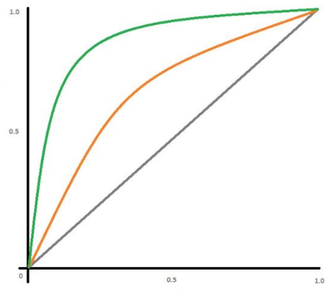
Assignment 6: Apply NB

1. **Apply Multinomial NB on these feature sets**
    - Set 1: categorical, numerical features + preprocessed_eassay (BOW)
    - Set 2: categorical, numerical features + preprocessed_eassay (TFIDF)
2. **The hyper paramter tuning(find best alpha:smoothing parameter)**
    - Find the best hyper parameter which will give the maximum AUC value
    - find the best hyper paramter using k-fold cross validation(use GridsearchCV or RandomsearchCV)/simple cross validation data (write for loop to iterate over hyper parameter values)
    -
3. **Representation of results**
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



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



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

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

4. fine the top 20 features from either from feature Set 1 or feature Set 2 using absolute values of `feature_log_prob_ ` parameter of `MultinomialNB` (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html) and print their corresponding feature names
5. You need to summarize the results at the end of the notebook, summarize it in the table format

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

2. Naive Bayes

```
%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 sklearn.neighbors import KNeighborsClassifier



from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

1.1 Loading Data

```
data  = pd.read_csv('/content/drive/My Drive/Colab Notebooks/preprocessed_data.csv',encoding='utf7')
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 109248 entries, 0 to 109247
Data columns (total 10 columns):
school_state                                109248 non-null object
teacher_prefix                              109248 non-null object
project_grade_category                      109248 non-null object
teacher_number_of_previously_posted_projects    109248 non-null int64
project_is_approved                         109248 non-null int64
clean_categories                            109248 non-null object
clean_subcategories                         109248 non-null object
essay                                       109248 non-null object
price                                       109248 non-null float64
project_title                               109248 non-null object
dtypes: float64(1), int64(2), object(7)
memory usage: 8.3+ MB
```

```
feature_Bow = list()
feature_Tf_idf = list()


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_posted_projects	clean_categories	clean_subcategories	essay	price	project_title
0	ca	mrs	grades_prek_2	53	math_science	appliedsciences health_lifescience	i fortunate enough use fairy tale stem kits cl...	725.05	educational support english learners home

1.2 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=0.33, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

1.3 Make Data Model Ready: encoding eassay, and project_title

project_title - Bag of Words

```
vectorizer = CountVectorizer(min_df=10,binary=True)
vectorizer.fit(X_train['project_title'].values)

project_title_bag_of_words_X_train = vectorizer.transform(X_train['project_title'].values)
project_title_bag_of_words_X_cv = vectorizer.transform(X_cv['project_title'].values)
project_title_bag_of_words_X_test = vectorizer.transform(X_test['project_title'].values)

feature_Bow = feature_Bow + vectorizer.get_feature_names()

print("="*50)
print("Bag of Words")
print("Shape of matrix after vectorizations (project_title)")
print("="*50)
print("X_train  :",project_title_bag_of_words_X_train.shape)
print("X_cv     :",project_title_bag_of_words_X_cv.shape)
print("X_test   :",project_title_bag_of_words_X_test.shape)
print("="*50)
print(vectorizer.get_feature_names())
print("="*50)
```

```
==================================================
Bag of Words
Shape of matrix after vectorizations (project_title)
==================================================
X_train  : (49041, 2095)
X_cv     : (24155, 2095)
X_test   : (36052, 2095)
==================================================
['04', '05', '10', '100', '101', '123', '16', '1st', '2016', '2017', '21st', '2nd', '3d', '3doodler', '3doodlers', '3rd', '4th', '5th', '60', '6th', '7th', '8th', 'abc', 'abcs', 'abilities', 'aboard', 'about', 'above', 'academic', 'academics', 'aca
==================================================
```

Project_title Tf-idf

```
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['project_title'].values)

project_title_tfidf_X_train = vectorizer.transform(X_train['project_title'].values)
project_title_tfidf_X_cv = vectorizer.transform(X_cv['project_title'].values)
project_title_tfidf_X_test = vectorizer.transform(X_test['project_title'].values)

feature_Tf_idf = feature_Tf_idf + vectorizer.get_feature_names()

print("="*50)
print("Tfidf Vectorizer")
print("Shape of matrix after vectorizations (project_title)")
print("="*50)
print("X_train  :",project_title_tfidf_X_train.shape)
print("X_cv     :",project_title_tfidf_X_cv.shape)
print("X_test   :",project_title_tfidf_X_test.shape)
print("="*50)
#print(vectorizer.get_feature_names())
#print("="*50)
```

```
==================================================
Tfidf Vectorizer
Shape of matrix after vectorizations (project_title)
==================================================
X_train  : (49041, 2095)
X_cv     : (24155, 2095)
X_test   : (36052, 2095)
==================================================
```

Essay - Bag of Words

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

essay_bag_of_words_X_train = vectorizer.transform(X_train['essay'].values)
essay_bag_of_words_X_cv = vectorizer.transform(X_cv['essay'].values)
essay_bag_of_words_X_test = vectorizer.transform(X_test['essay'].values)

feature_Bow = feature_Bow + vectorizer.get_feature_names()

print("="*50)
print("Bag of Words")
print("Shape of matrix after vectorizations (essay)")
print("="*50)
print("X_train  :",essay_bag_of_words_X_train.shape)
print("X_cv     :",essay_bag_of_words_X_cv.shape)
print("X_test   :",essay_bag_of_words_X_test.shape)
print("="*50)
print(vectorizer.get_feature_names())
print("="*50)
```

```
==================================================
Bag of Words
Shape of matrix after vectorizations (essay)
==================================================
X_train  : (49041, 12157)
X_cv     : (24155, 12157)
X_test   : (36052, 12157)
==================================================
['00', '000', '00pm', '10', '100', '1000', '100th', '101', '102', '103', '105', '10th', '11', '110', '1100', '115', '11th', '12', '120', '1200', '125', '12th', '13', '130', '1300', '135', '13th', '14', '140', '1400', '14th', '15', '150', '1500', '1
==================================================
```

Essay - Tf-idf

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['essay'].values)

essay_tfidf_X_train = vectorizer.transform(X_train['essay'].values)
essay_tfidf_X_cv = vectorizer.transform(X_cv['essay'].values)
essay_tfidf_X_test = vectorizer.transform(X_test['essay'].values)

feature_Tf_idf = feature_Tf_idf + vectorizer.get_feature_names()

print("="*50)
print("Tfidf Vectorizer")
print("Shape of matrix after vectorizations (essay)")
print("="*50)
print("X_train  :",essay_tfidf_X_train.shape)
print("X_cv     :",essay_tfidf_X_cv.shape)
print("X_test   :",essay_tfidf_X_test.shape)
print("="*50)
#print(vectorizer.get_feature_names())
#print("="*50)
```

```
==================================================
Tfidf Vectorizer
Shape of matrix after vectorizations (essay)
==================================================
X_train  : (49041, 12157)
X_cv     : (24155, 12157)
X_test   : (36052, 12157)
==================================================
```

1.4 Make Data Model Ready: encoding numerical, categorical features

school_state

```
# we use count vectorizer to convert the values into one hot encoded features

from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(binary=True)
vectorizer.fit(X_train['school_state'].values)

states_one_hot_X_train = vectorizer.transform(X_train['school_state'].values)
states_one_hot_X_cv = vectorizer.transform(X_cv['school_state'].values)
states_one_hot_X_test = vectorizer.transform(X_test['school_state'].values)

feature_Bow = feature_Bow + vectorizer.get_feature_names()
feature_Tf_idf = feature_Tf_idf + vectorizer.get_feature_names()

print("="*50)
print("Shape of matrix after one hot encoding(school_state) ")
print("="*50)
print("X_train  :",states_one_hot_X_train.shape)
print("X_cv     :",states_one_hot_X_cv.shape)
print("X_test   :",states_one_hot_X_test.shape)
print("="*50)
print(vectorizer.get_feature_names())
print("="*50)
```

```
==================================================
Shape of matrix after one hot encoding(school_state)
==================================================
X_train  : (49041, 51)
X_cv     : (24155, 51)
X_test   : (36052, 51)
==================================================
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', '
==================================================
```

teacher_prefix

```
# we use count vectorizer to convert the values into one hot encoded features

vectorizer = CountVectorizer(binary=True)
vectorizer.fit(X_train['teacher_prefix'].values)

teacher_prefix_one_hot_X_train = vectorizer.transform(X_train['teacher_prefix'].values)
teacher_prefix_one_hot_X_cv = vectorizer.transform(X_cv['teacher_prefix'].values)
teacher_prefix_one_hot_X_test = vectorizer.transform(X_test['teacher_prefix'].values)

feature_Bow = feature_Bow + vectorizer.get_feature_names()
feature_Tf_idf = feature_Tf_idf + vectorizer.get_feature_names()

print("="*50)
print("Shape of matrix after one hot encoding(teacher_prefix) ")
print("="*50)
print("X_train  :",teacher_prefix_one_hot_X_train.shape)
print("X_cv     :",teacher_prefix_one_hot_X_cv.shape)
print("X_test   :",teacher_prefix_one_hot_X_test.shape)
print("="*50)
print(vectorizer.get_feature_names())
print("="*50)
```

```
==================================================
Shape of matrix after one hot encoding(teacher_prefix)
==================================================
X_train  : (49041, 5)
X_cv     : (24155, 5)
X_test   : (36052, 5)
==================================================
['dr', 'mr', 'mrs', 'ms', 'teacher']
==================================================
```

project_grade_category

```
# we use count vectorizer to convert the values into one hot encoded features

vectorizer = CountVectorizer(binary=True)
vectorizer.fit(X_train['project_grade_category'].values)

project_grade_one_hot_X_train = vectorizer.transform(X_train['project_grade_category'].values)
project_grade_one_hot_X_cv = vectorizer.transform(X_cv['project_grade_category'].values)
project_grade_one_hot_X_test = vectorizer.transform(X_test['project_grade_category'].values)

feature_Bow = feature_Bow + vectorizer.get_feature_names()
feature_Tf_idf = feature_Tf_idf + vectorizer.get_feature_names()

print("="*50)
print("Shape of matrix after one hot encoding(project_grade_category) ")
print("="*50)
print("X_train  :",project_grade_one_hot_X_train.shape)
print("X_cv     :",project_grade_one_hot_X_cv.shape)
print("X_test   :",project_grade_one_hot_X_test.shape)
print("="*50)
print(vectorizer.get_feature_names())
print("="*50)
```

```
==================================================
Shape of matrix after one hot encoding(project_grade_category)
==================================================
X_train  : (49041, 4)
X_cv     : (24155, 4)
X_test   : (36052, 4)
==================================================
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
==================================================
```

clean_categories

```
# we use count vectorizer to convert the values into one hot encoded features

vectorizer = CountVectorizer(binary=True)
vectorizer.fit(X_train['clean_categories'].values)

clean_categories_one_hot_X_train = vectorizer.transform(X_train['clean_categories'].values)
clean_categories_one_hot_X_cv = vectorizer.transform(X_cv['clean_categories'].values)
clean_categories_one_hot_X_test = vectorizer.transform(X_test['clean_categories'].values)

feature_Bow = feature_Bow + vectorizer.get_feature_names()
feature_Tf_idf = feature_Tf_idf + vectorizer.get_feature_names()

print("="*50)
print("Shape of matrix after one hot encoding(clean_categories) ")
print("="*50)
print("X_train  :",clean_categories_one_hot_X_train.shape)
print("X_cv     :",clean_categories_one_hot_X_cv.shape)
print("X_test   :",clean_categories_one_hot_X_test.shape)
print("="*50)
print(vectorizer.get_feature_names())
print("="*50)
```

```
==================================================
Shape of matrix after one hot encoding(clean_categories)
==================================================
X_train  : (49041, 9)
X_cv     : (24155, 9)
X_test   : (36052, 9)
==================================================
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
==================================================
```

clean_subcategories

```
# we use count vectorizer to convert the values into one hot encoded features

vectorizer = CountVectorizer(binary=True)
vectorizer.fit(X_train['clean_subcategories'].values)

clean_subcategories_one_hot_X_train = vectorizer.transform(X_train['clean_subcategories'].values)
clean_subcategories_one_hot_X_cv = vectorizer.transform(X_cv['clean_subcategories'].values)
clean_subcategories_one_hot_X_test = vectorizer.transform(X_test['clean_subcategories'].values)

feature_Bow = feature_Bow + vectorizer.get_feature_names()
feature_Tf_idf = feature_Tf_idf + vectorizer.get_feature_names()

print("="*50)
print("Shape of matrix after one hot encoding(clean_categories) ")
print("="*50)
print("X_train  :",clean_subcategories_one_hot_X_train.shape)
print("X_cv     :",clean_subcategories_one_hot_X_cv.shape)
print("X_test   :",clean_subcategories_one_hot_X_test.shape)
print("="*50)
print(vectorizer.get_feature_names())
print("="*50)
```

```
==================================================
Shape of matrix after one hot encoding(clean_categories)
==================================================
X_train  : (49041, 30)
X_cv     : (24155, 30)
X_test   : (36052, 30)
==================================================
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym
==================================================
```

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))

previously_posted_X_train = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
previously_posted_X_cv = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
```

```
previously_posted_X_cv = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
previously_posted_X_test = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))



previously_posted_X_train = previously_posted_X_train.reshape(-1,1)
previously_posted_X_cv = previously_posted_X_cv.reshape(-1,1)
previously_posted_X_test = previously_posted_X_test.reshape(-1,1)

feature_Bow = feature_Bow + ['teacher_number_of_previously_posted_projects']
feature_Tf_idf = feature_Tf_idf + ['teacher_number_of_previously_posted_projects']

print("="*50)
print("Shape of matrix after vectorizations(teacher_number_of_previously_posted_projects) ")
print("="*50)
print("X_train  :",previously_posted_X_train.shape)
print("X_cv     :",previously_posted_X_cv.shape)
print("X_test   :",previously_posted_X_test.shape)
print("="*50)
```

```
==================================================
Shape of matrix after vectorizations(teacher_number_of_previously_posted_projects)
==================================================
X_train  : (49041, 1)
X_cv     : (24155, 1)
X_test   : (36052, 1)
==================================================
```

price

```
normalizer = Normalizer()

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

price_X_train = normalizer.transform(X_train['price'].values.reshape(1,-1))
price_X_cv = normalizer.transform(X_cv['price'].values.reshape(1,-1))
price_X_test = normalizer.transform(X_test['price'].values.reshape(1,-1))

feature_Bow = feature_Bow + ['price']
feature_Tf_idf = feature_Tf_idf + ['price']

price_X_train = price_X_train.reshape(-1,1)
price_X_cv = price_X_cv.reshape(-1,1)
price_X_test = price_X_test.reshape(-1,1)

print("="*50)
print("Shape of matrix after vectorizations(price) ")
print("="*50)
print("X_train  :",price_X_train.shape)
print("X_cv     :",price_X_cv.shape)
print("X_test   :",price_X_test.shape)
print("="*50)
```

```
==================================================
Shape of matrix after vectorizations(price)
==================================================
X_train  : (49041, 1)
X_cv     : (24155, 1)
X_test   : (36052, 1)
==================================================
```

Combining all feature vecters into one Vecter

```
from scipy.sparse import hstack
#Bag_of_words
X_train_Bow = hstack((states_one_hot_X_train,teacher_prefix_one_hot_X_train,project_grade_one_hot_X_train,clean_categories_one_hot_X_train,
clean_subcategories_one_hot_X_train,previously_posted_X_train,price_X_train,essay_bag_of_words_X_train,project_title_bag_of_words_X_train)).

X_cv_Bow = hstack((states_one_hot_X_cv, teacher_prefix_one_hot_X_cv,project_grade_one_hot_X_cv,clean_categories_one_hot_X_cv,
clean_subcategories_one_hot_X_cv,previously_posted_X_cv,price_X_cv,essay_bag_of_words_X_cv,project_title_bag_of_words_X_cv)).tocsr()

X_test_Bow = hstack((states_one_hot_X_test, teacher_prefix_one_hot_X_test,project_grade_one_hot_X_test,clean_categories_one_hot_X_test,
clean_subcategories_one_hot_X_test,previously_posted_X_test,price_X_test,essay_bag_of_words_X_test,project_title_bag_of_words_X_test)).tocsr



idf
ain_tfidf = hstack((states_one_hot_X_train,teacher_prefix_one_hot_X_train,project_grade_one_hot_X_train,clean_categories_one_hot_X_train,
n_subcategories_one_hot_X_train,previously_posted_X_train,price_X_train,essay_tfidf_X_train,project_title_tfidf_X_train)).tocsr()
_tfidf = hstack((states_one_hot_X_cv, teacher_prefix_one_hot_X_cv,project_grade_one_hot_X_cv,clean_categories_one_hot_X_cv,
n_subcategories_one_hot_X_cv,previously_posted_X_cv,price_X_cv,essay_tfidf_X_cv,project_title_tfidf_X_cv)).tocsr()

st_tfidf = hstack((states_one_hot_X_test, teacher_prefix_one_hot_X_test,project_grade_one_hot_X_test,clean_categories_one_hot_X_test,
n_subcategories_one_hot_X_test,previously_posted_X_test,price_X_test,essay_tfidf_X_test,project_title_tfidf_X_test)).tocsr()
```

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

Apply NB on different kind of featurization as mentioned in the instructions

For Every model that you work on make sure you do the step 2 and step 3 of instrucations

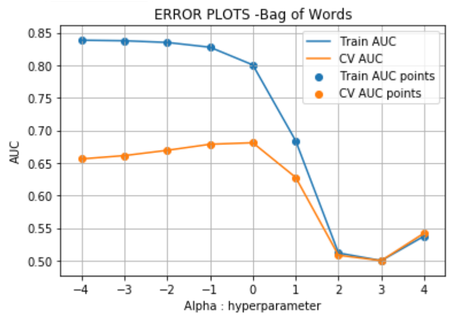
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred


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

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

BAG of Words

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.naive_bayes import MultinomialNB
import math

train_auc = []
cv_auc = []
y_train = y_train.reshape(y_train.size, 1)
alpha = [10**x for x in range(-4,5)]
alpha_log = [math.log(10**x,10) for x in range(-4,5)]

for i in tqdm(alpha):
    classifer = MultinomialNB(alpha = i,class_prior=[0.5,0.5])
    classifer.fit(X_train_Bow.toarray(), y_train)

    y_train_pred = batch_predict(classifer, X_train_Bow)
    y_cv_pred = batch_predict(classifer, X_cv_Bow)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

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

```
plt.legend()
plt.xlabel("Alpha : hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS -Bag of Words")
plt.grid()
plt.show()
```

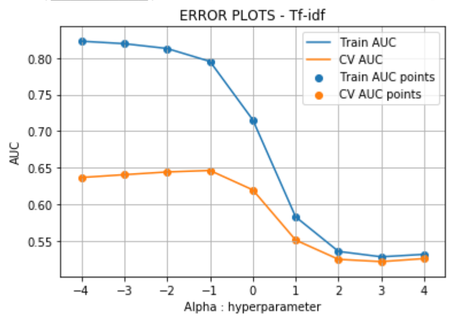
```
100%|██████████| 9/9 [00:36<00:00,  3.90s/it]
```



TF-idf

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

train_auc = []
cv_auc = []
y_train = y_train.reshape(y_train.size, 1)
alpha = [10**x for x in range(-4,5)]
alpha_log = [math.log(10**x,10) for x in range(-4,5)]

for i in tqdm(alpha):
    classifer = MultinomialNB(alpha = i,class_prior=[0.5,0.5])
    classifer.fit(X_train_tfidf.toarray(), y_train)
    y_train_pred = batch_predict(classifer, X_train_tfidf)
    y_cv_pred = batch_predict(classifer, X_cv_tfidf)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))



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

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

plt.legend()
plt.xlabel("Alpha : hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS - Tf-idf")
plt.grid()
plt.show()
```

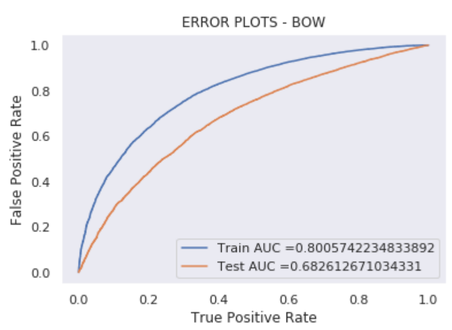
```
100%|██████████| 9/9 [00:33<00:00,  3.77s/it]
```




```
# Here I'm selecting 40 as a best alpha  by seeing above plot
best_alpha = 1
```

Bag of Words with best Alpha

```
#https://datascience.stackexchange.com/questions/65219/find-the-top-n-features-from-feature-set-using-absolute-values-of-feature-log-p
def most_features(sorted_prob_class,features_lst):
  Most_imp_words = []

  for index in sorted_prob_class[-20:-1]:
    Most_imp_words.append(features_lst[index])

  return Most_imp_words


classifer = MultinomialNB(alpha = best_alpha,class_prior=[0.5,0.5])
classifer.fit(X_train_Bow.toarray(), y_train)

y_train_pred = batch_predict(classifer, X_train_Bow)
y_test_pred = batch_predict(classifer, X_test_Bow)

train_fpr_bow, train_tpr_bow, tr_thresholds_bow = roc_curve(y_train, y_train_pred)
test_fpr_bow, test_tpr_bow, te_thresholds_bow = roc_curve(y_test, y_test_pred)


plt.plot(train_fpr_bow, train_tpr_bow, label="Train AUC ="+str(auc(train_fpr_bow, train_tpr_bow)))
plt.plot(test_fpr_bow, test_tpr_bow, label="Test AUC ="+str(auc(test_fpr_bow, test_tpr_bow)))
plt.legend()
plt.xlabel("True Positive Rate")
plt.ylabel("False Positive Rate")
plt.title("ERROR PLOTS - BOW")
plt.grid()
plt.show()
```

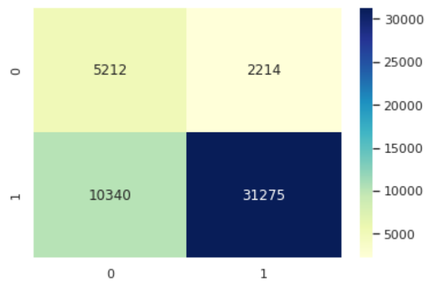



```
sorted_prob_class = classifer.feature_log_prob_[0, :].argsort()

imp_features = most_features(sorted_prob_class[-20:-1],feature_Tf_idf)
print("="*100)
print("importent features for class 0 " ,imp_features)
print("="*100)

sorted_prob_class = classifer.feature_log_prob_[1, :].argsort()

imp_features = most_features(sorted_prob_class[-20:-1],feature_Tf_idf)
print("="*100)
print("importent features for class 1 " ,imp_features)
print("="*100)
```

```
====================================================================================================
importent features for class 0  ['aid', 'character', 'all', 'significance', 'soft', '92', 'hesitate', 'adventures', 'full', 'reinforced', 'diorama', 'refuge', 'fee', 'horseshoe', '200', 'feeders', 'he', 'ordered']
====================================================================================================
====================================================================================================
importent features for class 1  ['character', 'scripts', '92', 'soft', 'all', 'significance', 'hesitate', 'adventures', 'full', 'diorama', 'reinforced', 'fee', 'refuge', 'horseshoe', '200', 'feeders', 'he', 'ordered']
====================================================================================================
```

```
import seaborn as sns; sns.set()
best_t = find_best_threshold(tr_thresholds_bow, train_fpr_bow, train_tpr_bow)
print("="*50)
print("Train confusion matrix for BOW")
print("="*50)

#https://seaborn.pydata.org/generated/seaborn.heatmap.html
uniform_data = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
ax = sns.heatmap(uniform_data,cmap="YlGnBu",annot=True,fmt="d")
```

```
The maximum value of tpr*(1-fpr) 0.5274689281555345 for threshold 0.472
==================================================
Train confusion matrix for BOW
==================================================
```

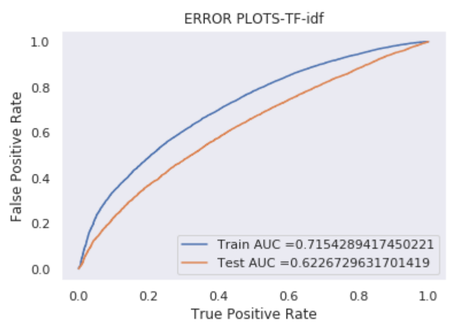



```
print("="*50)
print("Test confusion matrix for BOW")
print("="*50)

uniform_data = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
ax = sns.heatmap(uniform_data,cmap="YlGnBu",annot=True,fmt="d")
```

```
==================================================
Test confusion matrix for BOW
==================================================
```


TF-Idf with best Alpha

```
classifer = MultinomialNB(alpha = best_alpha,class_prior=[0.5,0.5])
classifer.fit(X_train_tfidf.toarray(), y_train)

y_train_pred = batch_predict(classifer, X_train_tfidf)
y_test_pred = batch_predict(classifer, X_test_tfidf)


train_fpr_tfidf, train_tpr_tfidf, tr_thresholds_tfidf = roc_curve(y_train, y_train_pred)
test_fpr_tfidf, test_tpr_tfidf, te_thresholds_tfidf = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr_tfidf, train_tpr_tfidf, label="Train AUC ="+str(auc(train_fpr_tfidf, train_tpr_tfidf)))
plt.plot(test_fpr_tfidf, test_tpr_tfidf, label="Test AUC ="+str(auc(test_fpr_tfidf, test_tpr_tfidf)))
plt.legend()
plt.xlabel("True Positive Rate")
plt.ylabel("False Positive Rate")
plt.title("ERROR PLOTS-TF-idf")
plt.grid()
plt.show()
```




```
sorted_prob_class = classifer.feature_log_prob_[0, :].argsort()

imp_features = most_features(sorted_prob_class[-20:-1],feature_Tf_idf)
print("="*100)
print("importent features for class 0 " ,imp_features)
print("="*100)

sorted_prob_class = classifer.feature_log_prob_[1, :].argsort()

imp_features = most_features(sorted_prob_class[-20:-1],feature_Tf_idf)
print("="*100)
print("importent features for class 1 " ,imp_features)
print("="*100)
```

```
====================================================================================================
importent features for class 0  ['allows', 'adventure', 'ahead', 'alouds', 'air', 'properly', 'algebra', '101', 'along', 'apples', 'age', 'answers', 'another', 'ants', 'again', 'african', 'allow', 'aid']
====================================================================================================
====================================================================================================
importent features for class 1  ['alouds', 'adventure', 'animals', 'ahead', 'air', 'apples', 'along', 'properly', 'algebra', '101', 'age', 'answers', 'ants', 'another', 'again', 'african', 'allow', 'aid']
====================================================================================================
```

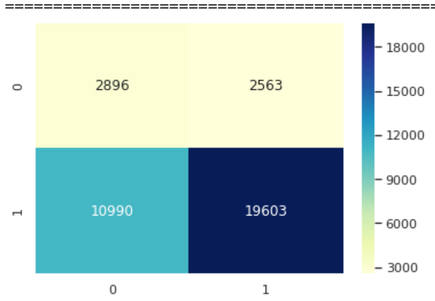
```
best_t = find_best_threshold(tr_thresholds_tfidf, train_fpr_tfidf, train_tpr_tfidf)
print("="*50)
print("Train confusion matrix for TF-idf")
print("="*50)
uniform_data = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
ax = sns.heatmap(uniform_data,cmap="YlGnBu",annot=True,fmt="d")
```

```
The maximum value of tpr*(1-fpr) 0.4279507343212775 for threshold 0.589
==================================================
Train confusion matrix for TF-idf
==================================================
```



```
print("="*50)
print("Test confusion matrix for TF-idf")
print("="*50)


uniform_data = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
ax = sns.heatmap(uniform_data,cmap="YlGnBu",annot=True,fmt="d")
```

```
==================================================
Test confusion matrix for TF-idf
==================================================
```



3. Summary

as mentioned in the step 5 of instructions

```
rom prettytable import PrettyTable
```

```
 = PrettyTable()
.field_names = ["Vectorizer", "Model", "HyperParameter", "AUC_Train","AUC_Test"]
.add_row(["BOW", "MultinomialNB", best_alpha, str(auc(train_fpr_bow, train_tpr_bow)),str(auc(test_fpr_bow, test_tpr_bow))])
.add_row(["TF-idf", "MultinomialNB", best_alpha,  str(auc(train_fpr_tfidf, train_tpr_tfidf)),str(auc(test_fpr_tfidf, test_tpr_tfidf))])

rint(x)
```

```
+------------+---------------+----------------+--------------------+--------------------+
| Vectorizer |     Model     | HyperParameter |     AUC_Train      |      AUC_Test      |
+------------+---------------+----------------+--------------------+--------------------+
|    BOW     | MultinomialNB |       1        | 0.8005742234833892 | 0.682612671034331  |
|   TF-idf   | MultinomialNB |       1        | 0.7154289417450221 | 0.6226729631701419 |
+------------+---------------+----------------+--------------------+--------------------+
```