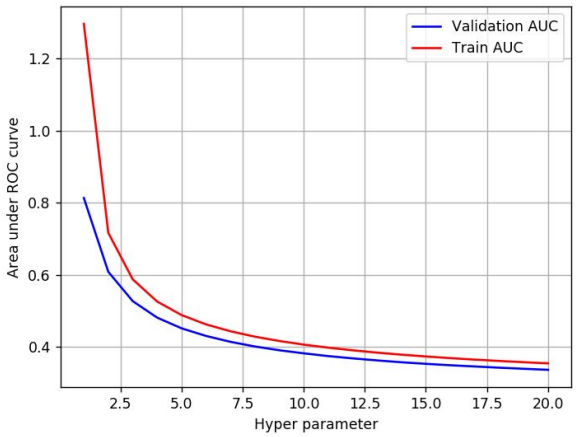
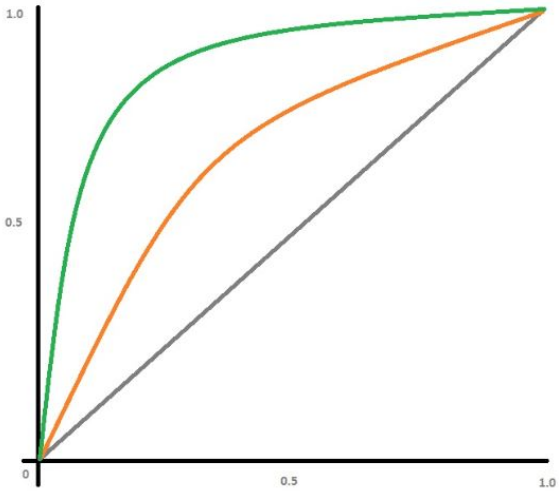
Assignment 6: Apply NB

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 Randon (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 figure



   - Once after you found the best hyper parameter, you need to train your model with it, an curve on both train and test.



- Along with plotting ROC curve, you need to print the [confusion matrix](#) with predicted an

	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 `MultinomialNB` (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.Mu corresponding feature names
5. You need to summarize the results at the end of the notebook, summarize it in the table form

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

```
!pip install chart_studio
```

```
Collecting chart_studio
  Downloading https://files.pythonhosted.org/packages/ca/ce/330794a6b6ca4b9182c38fc69
     |████████████████████████████████| 71kB 2.3MB/s
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from ch
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: plotly in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (fr
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/python3.6/dist
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist
Installing collected packages: chart-studio
Successfully installed chart-studio-1.1.0
```

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

import pandas as pd
import numpy as np
import nltk
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc

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

import pickle
from tqdm import tqdm
import os

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

1.1 Loading Data

```
!curl --header "Host: doc-0o-7s-docs.googleusercontent.com" --header "User-Agent: Mozilla/
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  118M    0  118M    0     0  99.2M      0 --:--:--  0:00:01 --:--:-- 99.2M
```

```
data  = pd.read_csv('preprocessed_data.csv', nrows=50000)
data.head(2)
```

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously
0	ca	mrs	grades_prek_2	
1	ut	ms	grades_3_5	

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

Double-click (or enter) to edit

```
#separating y from dataframe

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

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously
0	ca	mrs	grades_prek_2	

```
data.columns.values
```

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

```
# separating data into train and test.
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)
```

1.3 Make Data Model Ready: encoding eassay, and project_title

Bag of words preprocessing of eassay

```
# preprocessing eassay in bow.

print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

print("="*100)


vectorizer_eassy = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer_eassy.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_eassy.transform(X_train['essay'].values)
X_test_essay_bow = vectorizer_eassy.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
```

```
print(X_test_essay_bow.shape, y_test.shape)
print("="*100)
```

```
(33500, 8) (33500,)
(16500, 8) (16500,)
====================================================================================================
After vectorizations
(33500, 5000) (33500,)
(16500, 5000) (16500,)
====================================================================================================
```

TFIDF vectorizer

```
# tfidf vectorizer os essay
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
print("="*100)
vectorizer_tfidf = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer_tfidf.fit(X_train['essay'].values)
X_train_eassy_tfidf = vectorizer_tfidf.fit_transform(X_train['essay'].values)
X_test_eassy_tfidf = vectorizer_tfidf.fit_transform(X_test['essay'].values)
print("After vectorizations")
print(X_train_eassy_tfidf.shape, y_train.shape)
print(X_test_eassy_tfidf.shape, y_test.shape)
print("="*100)
```

```
(33500, 8) (33500,)
(16500, 8) (16500,)
====================================================================================================
After vectorizations
(33500, 5000) (33500,)
(16500, 5000) (16500,)
====================================================================================================
```

1.4 Make Data Model Ready: encoding numerical, categorical feature

encoding categorical features: School State

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

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

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer_school_state.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 51) (33500,)
(16500, 51) (16500,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id',
==================================================================================
```

encoding categorical features: teacher_prefix

```
vectorizer_teacher_prefix = CountVectorizer()
vectorizer_teacher_prefix.fit(X_train['teacher_prefix'].values) # fit has to happen only o

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

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

```
After vectorizations
(33500, 5) (33500,)
(16500, 5) (16500,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
==================================================================================
```

encoding categorical features: project_grade_category

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer_project_grade_category.transform(X_train['project_grade_cat
X_test_grade_ohe = vectorizer_project_grade_category.transform(X_test['project_grade_categ

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

```
After vectorizations
(33500, 4) (33500,)
(16500, 4) (16500,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
==================================================================================
```

encoding categorical features: clean_categories

```
vectorizer_clean_categories = CountVectorizer()
vectorizer_clean_categories.fit(X_train['clean_categories'].values) # fit has to happen on

# we use the fitted CountVectorizer to convert the text to vector
X_train_category_ohe= vectorizer_clean_categories.transform(X_train['clean_categories'].va
X_test_category_ohe = vectorizer_clean_categories.transform(X_test['clean_categories'].val

print("After vectorizations")
print(X_train_category_ohe.shape, y_train.shape)
print(X_test_category_ohe.shape, y_test.shape)
print(vectorizer_clean_categories.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 9) (33500,)
(16500, 9) (16500,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_langu
====================================================================================
```

encoding categorical features: clean_subcategories

```
vectorizer_clean_subcategories = CountVectorizer()
vectorizer_clean_subcategories.fit(X_train['clean_subcategories'].values) # fit has to hap

# we use the fitted CountVectorizer to convert the text to vector
X_train_subcategory_ohe = vectorizer_clean_subcategories.transform(X_train['clean_subcateg
X_test_subcategory_ohe = vectorizer_clean_subcategories.transform(X_test['clean_subcategor

print("After vectorizations")
print(X_train_subcategory_ohe.shape, y_train.shape)
print(X_test_subcategory_ohe.shape, y_test.shape)
print(vectorizer_clean_subcategories.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 30) (33500,)
(16500, 30) (16500,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'colleg
====================================================================================
```

Encoding numerical features: Price

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

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

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

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

Encoding numerical features : teacher_number_of_previously_poste

```
normalizer = Normalizer()

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

X_train_teacher_posted_projects_norm = normalizer.transform(X_train['teacher_number_of_pre
X_test_teacher_posted_projects_norm = normalizer.transform(X_test['teacher_number_of_previ
```

```
print("After vectorizations")
print(X_train_teacher_posted_projects_norm.shape, y_train.shape)
print(X_test_teacher_posted_projects_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

Concatinating all the features

Data set1 with BOW

```
from scipy.sparse import hstack
X_tr_set1 = hstack((X_train_essay_bow, X_train_state_ohe, X_train_teacher_ohe, X_train_gra
X_te_set1 = hstack((X_test_essay_bow, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_o

print("Final Data matrix")
print(X_tr_set1.shape, y_train.shape)
print(X_te_set1.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(33500, 5101) (33500,)
(16500, 5101) (16500,)
====================================================================================================
```

data set2 with tfidf

```
X_tr_set2 = hstack((X_train_eassy_tfidf, X_train_state_ohe, X_train_teacher_ohe, X_train_g
X_te_set2 = hstack((X_test_eassy_tfidf, X_test_state_ohe, X_test_teacher_ohe, X_test_grade

print("Final Data matrix")
print(X_tr_set1.shape, y_train.shape)
print(X_te_set1.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(33500, 5101) (33500,)
(16500, 5101) (16500,)
====================================================================================
```

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

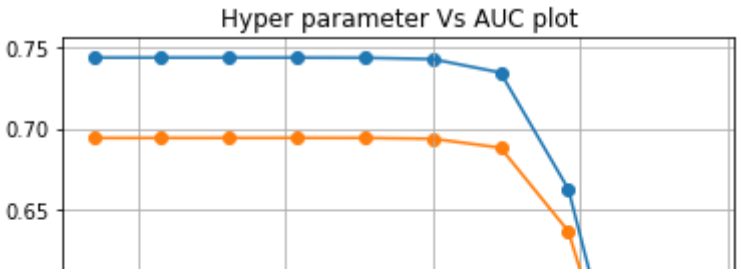
Apply NB on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

Appling NB on set1

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.h
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import RandomizedSearchCV

nb=MultinomialNB(class_prior=[0.5,0.5])
parameters = {'alpha':[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]}
clf = RandomizedSearchCV(nb, parameters, cv=10, scoring='roc_auc',return_train_score=True)
clf.fit(X_tr_set1, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_alpha'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
alpha =  results['param_alpha']



print(alpha)
```

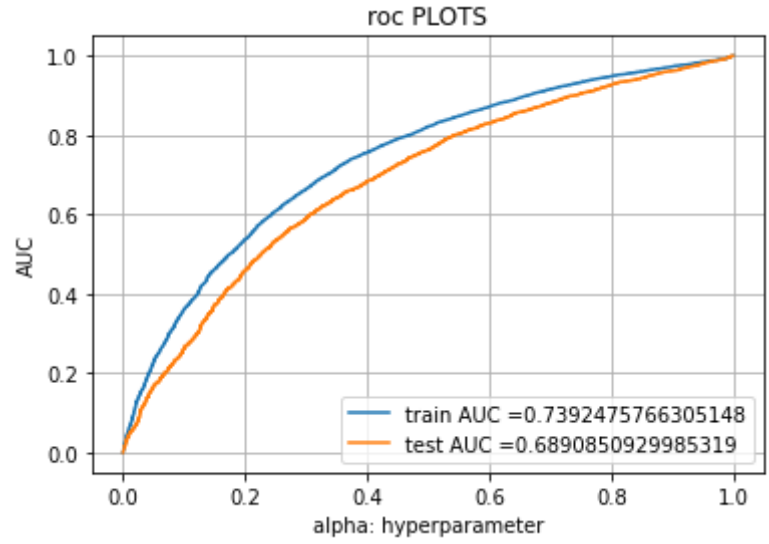
```
      0     1e-05
      1    0.0001
      2     0.001
      3      0.01
      4       0.1
      5         1
      6        10
      7       100
```

```
for i in range(len(alpha)):
  alpha[i]=np.log(alpha[i])


plt.plot(alpha , train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,

plt.plot(alpha, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darko

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

plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```




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

best_alpha= 0.01

nb = MultinomialNB(alpha=best_alpha)
nb.fit(X_tr_set1, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the
# not the predicted outputs

y_train_pred= clf.predict_proba(X_tr_set1)[:,1]
y_test_pred = clf.predict_proba(X_te_set1)[:,1]
print(y_train_pred[0:5])


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

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

```
[0.12627659 0.99991294 0.08816608 0.94613563 0.99803805]
```




```
# 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(
    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


print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

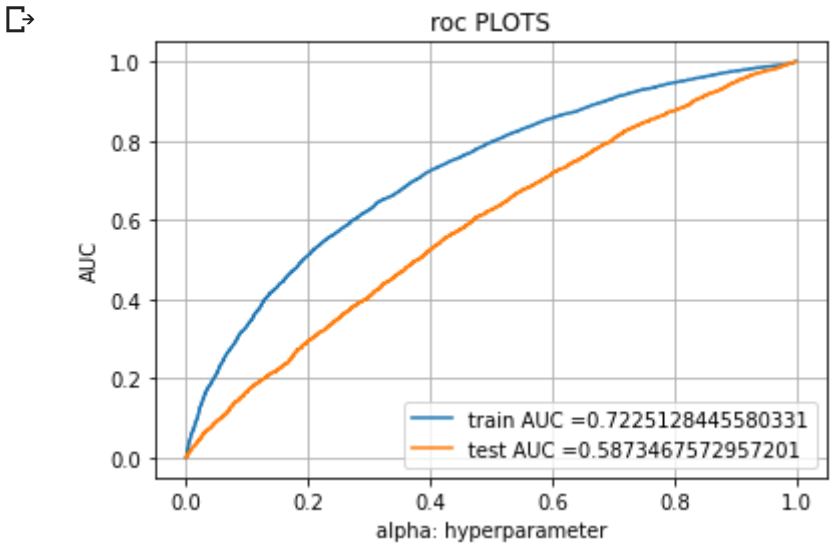
```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.46735636710965023 for threshold 0.487
Train confusion matrix
[[ 3662  1703]
 [ 8871 19264]]
Test confusion matrix
[[1635 1007]
 [4625 9233]]
```

Appling NB on set2

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.h
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import RandomizedSearchCV

nb=MultinomialNB(class_prior=[0.5,0.5])
parameters = {'alpha':[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]}
clf = RandomizedSearchCV(nb, parameters, cv=10, scoring='roc_auc',return_train_score=True)
clf.fit(X_tr_set2, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_alpha'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
alpha =  results['param_alpha']
```

```
for i in range(len(alpha)):
  alpha[i]=np.log(alpha[i])


plt.plot(alpha , train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,

plt.plot(alpha, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darko

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

plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```

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

best_alpha= 0.007

nb = MultinomialNB(alpha=best_alpha)
clf.fit(X_tr_set2, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the
# not the predicted outputs

y_train_pred= clf.predict_proba(X_tr_set2)[:,1]
y_test_pred = clf.predict_proba(X_te_set2)[:,1]


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

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




```
# 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(
    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


print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.4425935399007348 for threshold 0.512
Train confusion matrix
[[ 3679  1686]
 [ 9976 18159]]
Test confusion matrix
[[  880  1762]
 [ 3122 10736]]
```

finding the top 20 features from either from feature Set 1

```
# puting all features in one list
all_features =[]
all_features.extend(vectorizer_eassy.get_feature_names())
all_features.extend(vectorizer_school_state.get_feature_names())
all_features.extend(vectorizer_teacher_prefix.get_feature_names())
all_features.extend(vectorizer_project_grade_category.get_feature_names())
all_features.extend(vectorizer_clean_categories.get_feature_names())
all_features.extend(vectorizer_clean_subcategories.get_feature_names())
p=["price"]
t=['teacher_number_of_previously_posted_projects']
all_features.extend(p)
all_features.extend(t)
print(all_features)
all_features =np.array(all_features)
print(type(all_features))
```

```
['000', '10', '100', '100 free', '100 percent', '100 students', '100 students receive
<class 'numpy.ndarray'>
```

```
# https://stackoverflow.com/questions/50526898/how-to-get-feature-importance-in-naive-baye
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB(alpha=300)
nb = clf.fit(X_tr_set1, y_train)

neg_class_prob_sorted = nb.feature_log_prob_[0, :].argsort()[::-1]
pos_class_prob_sorted = nb.feature_log_prob_[1, :].argsort()[::-1]
```

```
print("neg_top_20_features ")
print(all_features[neg_class_prob_sorted[:20]])
print("=="*50)
print("pos_top_20_features")
print(all_features[pos_class_prob_sorted[:20]])
```

```
neg_top_20_features
['students' 'school' 'learning' 'my' 'classroom' 'not' 'learn' 'help'
 'they' 'the' 'my students' 'price' 'nannan' 'many' 'we' 'need' 'work'
 'come' 'year' 'love']
====================================================================================================
pos_top_20_features
['students' 'school' 'my' 'classroom' 'learning' 'the' 'not' 'they'
 'learn' 'my students' 'help' 'price' 'many' 'nannan' 'we' 'work'
 'reading' 'need' 'use' 'day']
```

3. Summary

as mentioned in the step 5 of instructions

```
from tabulate import tabulate
weather_data = [('BOw', 'Multinomial Naive Bayes', 0.01, 0.68),
                ('TFIDF', 'Multinomial Naive Bayes', 0.007, 0.58)
               ]
df = pd.DataFrame(weather_data, columns=['Vectorizer', 'Model', 'Hyper parameter', 'AUC'])


#Ref: https://pypi.org/project/tabulate/
print(tabulate(df, headers='keys', tablefmt='github'))
```

```
|    | Vectorizer   | Model                   |   Hyper parameter |   AUC |
|----|--------------|-------------------------|-------------------|-------|
|  0 | BOw          | Multinomial Naive Bayes |             0.01  |  0.68 |
|  1 | TFIDF        | Multinomial Naive Bayes |             0.007 |  0.58 |
```