**Assignment**

In this assignment students need to predict whether a person makes over

50K per year or not from classic adult dataset using XGBoost. The

description of the dataset is as follows:

**Data Set Information:**

Extraction was done by Barry Becker from the 1994 Census

database. A set of reasonably clean records was extracted using the

following conditions: ((AAGE>16) && (AGI>100) &&

(AFNLWGT>1)&& (HRSWK>0))

Attribute Information:

Listing of attributes: >50K, <=50K.

age: continuous.

workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov,

Local-gov, State-gov, Without-pay, Never-worked.

fnlwgt: continuous.

education: Bachelors, Some-college, 11th, HS-grad, Prof-school,

Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th,

Doctorate, 5th-6th, Preschool.

education-num: continuous.

marital-status: Married-civ-spouse, Divorced, Never-married,

Separated, Widowed, Married-spouse-absent, Married-AF-spouse.

occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-

managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct,

Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv,

Protective-serv, Armed-Forces.

relationship: Wife, Own-child, Husband, Not-in-family, Other-relative,

Unmarried.

race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.

sex: Female, Male.

capital-gain: continuous.

capital-loss: continuous.

hours-per-week: continuous.

native-country: United-States, Cambodia, England, Puerto-Rico,

Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan,

Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy,

Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France,

Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia,

Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-

Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.

**Following is the code to load required libraries and data:**

import numpy as np

import pandas as pd

train\_set = pd.read\_csv('http://archive.ics.uci.edu/ml/machine-

learning-databases/adult/adult.dat a', header = None)

test\_set = pd.read\_csv('http://archive.ics.uci.edu/ml/machine-learning-

databases/adult/adult.test' , skiprows = 1, header = None)

col\_labels = ['age', 'workclass', 'fnlwgt', 'education', 'education\_num',

'marital\_status', 'occupation','relationship', 'race', 'sex', capital\_gain',

'capital\_loss', 'hours\_per\_week', 'native\_country', 'wage\_class']

train\_set.columns = col\_labels

test\_set.columns = col\_labels

**Task:** Deploy this assignment in any cloud platform.(Try to look for free cloud platform)

**Assignment:** Submit assignment’s deployable link only.

**import numpy as np**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**import sklearn**

**from xgboost.sklearn import XGBClassifier**

**url='./adult.data'**

***#columns name according to adult.names***

**cols=["age","workclass","fnlwgt","education","education-num","marital-status", "occupation", "relationship", "race", "sex", "capital-gain", "capital-loss", "hours-per-week","native-country","wage\_class"]**

**train\_set=pd.read\_csv(url, names=cols) *# reading and setting names***

**train\_set.head()**

**test\_set = pd.read\_csv('./adult.test',skiprows = 1, header = None, names=cols) *#coz row one is multi index***

**test\_set.head()**

**EDA**

**train\_set.isnull().sum()**

***# no null values in any frame***

**train\_set.hist(figsize=(10,10))**

**plt.show()**

**train\_set.dtypes**

***#only 6 features(out of 14 are numeric)***

**train\_set.workclass.value\_counts()**

***#train\_set.workclass.unique()***

***# there is a '?' need to be handle***

**for feature in cols:**

**if train\_set[feature].dtype == 'object':**

**print('\_'\*10,feature, end=" ",)**

**print('\_'\*10)**

**print(train\_set[feature].value\_counts())**

**train\_set.columns**

**import re**

**train\_set.rename(columns=lambda name: re.sub(r"\-",'\_',name), inplace=True)**

**filter1 = train\_set["workclass"]=="?"**

***#index= train\_set.where(filter1).index***

**index = train\_set[train\_set["native\_country"]== ' ?'].index**

***#index.value\_counts() # to display all the indexes having value '?'***

**index.value\_counts().sum() *# sum of those indexes***

***# type(index)***

***#now apply the filter for the '?' value in those above 3 columns***

**indexes\_list = [] *# will contain three index series of desired syntexes***

**def filter\_questionMark(l):**

**for item in l:**

**print('\_'\*10,item, '\_'\*10)**

**index = train\_set[train\_set[item]== ' ?'].index**

**indexes\_list.append(index)**

***#index.value\_counts() # to display all the indexes having value '?'***

**print("Total `?` in {0}: {1}".format(item, index.value\_counts().sum())) *# sum of those indexes***

**print("Total % of `?` in {0}: {1}".format(item, round((index.value\_counts().sum()/train\_set.shape[0])\*100,2)))**

**filter2= ['workclass','occupation', 'native\_country']**

**filter\_questionMark(filter2)**

**print("Original Shape of Train Set:", train\_set.shape)**

**print("Rows in Train Set:", train\_set.shape[0])**

**print('\n Drowping `?` indexes from `workclass` featutre....')**

**train\_set= train\_set.drop(index)**

**print("Total New rows:",train\_set.shape[0])**

**print('\n Drowping `?` indexes from `native\_country` featutre....')**

**index1= train\_set[train\_set["native\_country"]== ' ?'].index**

**train\_set = train\_set.drop(index1)**

**print("Total New rows:",train\_set.shape[0])**

**print('\n Drowping `?` indexes from `occupation` featutre....')**

**index2 = train\_set[train\_set["occupation"]== ' ?'].index**

**train\_set = train\_set.drop(index2)**

**print("Total New rows:",train\_set.shape[0])**

***# check for '?' again***

**filter\_questionMark(filter2)**

#### Applying one hot coding to all the categorical variables

**df1 = train\_set.copy()**

**objectFeature = []**

**for i in list(df1.columns):**

**if (df1[i].dtypes == 'object'):**

**objectFeature.append(i)**

**df1=pd.get\_dummies(df1[objectFeature[:-1]])**

**intFeature = []**

**for i in list(train\_set.columns):**

**if (train\_set[i].dtypes == 'int64'):**

**intFeature.append(i)**

**df2 = train\_set[intFeature]**

**print(df1.shape)**

**print(df2.shape)**

**Concatinating the two frames**

**X\_train = pd.concat([df1, df2], axis=1)**

**X\_train.shape *# final dataframe to be get Trained***

**y=train\_set[objectFeature[-1]]**

### Using LabelEncoder over the target variable

**from sklearn import preprocessing**

**le = preprocessing.LabelEncoder()**

***# le.fit(y)***

***# le.classes\_ # will display only two classes---> array([' <=50K', ' >50K'], dtype=object)y\_train***

**y\_train=le.fit\_transform(y)**

**y\_train**

# Applying XGB : the sklearn way

**from xgboost.sklearn import XGBClassifier**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.metrics import accuracy\_score**

**params = {**

**'objective': 'binary:logistic',**

**'max\_depth': 2,**

**'learning\_rate': .5,**

**'silent': True, *# would be boolean in sklearn***

**'n\_estimators': 100**

**}**

**x\_train, x\_test, y\_t1, y\_t2 =train\_test\_split(X\_train, y\_train, test\_size=.25)**

**x\_train.shape, x\_test.shape, y\_t1.shape, y\_t2.shape**

#### Model Training : making model 1

**xgb = XGBClassifier(\*\*params).fit(x\_train,y\_t1)**

**y\_pred= xgb.predict(x\_test)**

**accuracy\_score(y\_t2, y\_pred)**

#### Making model no .2

**eval\_set = [(x\_train,y\_t1), (x\_test, y\_t2)]**

**xgb1 = XGBClassifier(\*\*params).fit(x\_train,y\_t1,**

**early\_stopping\_rounds=15,**

**eval\_metric=["error", "logloss"],**

**eval\_set=eval\_set,**

**verbose=True)**

**y\_pred1=xgb1.predict(x\_test)**

**accuracy\_score(y\_t2, y\_pred1)**

**Ploting Classifying errors and log loss with respect to each iteration**

***# retrieve performance metrics***

**results = xgb1.evals\_result()**

**epochs = len(results['validation\_0']['error'])**

**x = range(0, epochs)**

***# plot log loss***

**fig, ax = plt.subplots()**

**ax.plot(x, results['validation\_0']['logloss'], label='Train')**

**ax.plot(x, results['validation\_1']['logloss'], label='Test')**

**ax.legend()**

**plt.ylabel('Log Loss')**

**plt.xlabel('Epochs')**

**plt.title('XGBoost Log Loss')**

**plt.show()**

***# plot classification error***

**fig, ax = plt.subplots()**

**ax.plot(x, results['validation\_0']['error'], label='Train')**

**ax.plot(x, results['validation\_1']['error'], label='Test')**

**ax.legend()**

**plt.ylabel('Classification Error')**

**plt.xlabel('Epochs')**

**plt.title('XGBoost Classification Error')**

**plt.show()**

### model no. 3

**params = {**

**'objective': 'binary:logistic',**

**'max\_depth': 20,**

**'learning\_rate': .01,**

**'silent': True, *# would be boolean in sklearn***

**'n\_estimators': 200**

**}**

**eval\_set = [(x\_train,y\_t1), (x\_test, y\_t2)]**

**xgb1 = XGBClassifier(\*\*params).fit(x\_train,y\_t1,**

**early\_stopping\_rounds=15,**

**eval\_metric=["error", "logloss"],**

**eval\_set=eval\_set,**

**verbose=True)**

***#plot***

***# retrieve performance metrics***

**results = xgb1.evals\_result()**

**epochs = len(results['validation\_0']['error'])**

**x = range(0, epochs)**

***# plot log loss***

**fig, ax = plt.subplots()**

**ax.plot(x, results['validation\_0']['logloss'], label='Train')**

**ax.plot(x, results['validation\_1']['logloss'], label='Test')**

**ax.legend()**

**plt.ylabel('Log Loss')**

**plt.xlabel('Epochs')**

**plt.title('XGBoost Log Loss')**

**plt.show()**

***# plot classification error***

**fig, ax = plt.subplots()**

**ax.plot(x, results['validation\_0']['error'], label='Train')**

**ax.plot(x, results['validation\_1']['error'], label='Test')**

**ax.legend()**

**plt.ylabel('Classification Error')**

**plt.xlabel('Epochs')**

**plt.title('XGBoost Classification Error')**

**plt.show()**

**The XGBoost way**

***#Import Xgboost***

**import xgboost as xgb**

**dtrain = xgb.DMatrix(x\_train, label = y\_t1)**

**dtest = xgb.DMatrix(x\_test, label = y\_t2)**

***#creating watchlist of training***

***# to see out output***

**watchlist = [(dtrain,'train'),(dtest, 'eval')]**

**params = {**

**'objective': 'binary:logistic',**

**'max\_depth': 2,**

**'learning\_rate': 1.0,**

**'silent': True, *# would be boolean in sklearn***

**'booster' : 'gbtree',**

**'max\_depth' : 7,**

**'eval\_metric' : 'auc'**

**}**

***# using bydeafault eta [default=0.3, alias: learning\_rate]***

**num\_rounds = 100**

**model\_xgb = xgb.train(params, dtrain, num\_rounds, evals = watchlist, early\_stopping\_rounds = 15, verbose\_eval = True)**

**features\_contribution = pd.Series(model\_xgb.get\_fscore()).sort\_values(ascending=False)**

**plt.figure(figsize=(20,10))**

**features\_contribution[:50].plot(kind='bar', title='Feature Importances')**

***#features\_contribution[:50].plot(kind='line')***

**plt.ylabel('Feature Importance Score')**

**plt.show()**