import numpy as np

import pandas as pd

from sklearn. model selection import train\_test\_split, RandomizedSearchcv

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn. ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.metrics import accuracy\_score, roc\_auc\_score

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import GridSearchcV

from sklearn import metrics

import

matplotlib.pyplot as plt

##! pip install xgboost ##

import xgboost as xgb

from xgboost import XGBClassifier

loan\_train = pd.read\_csv('./loan-train.csv')

print (loan train. shape) # (614, 13)

loan\_train. head()

total\_null = loan\_train. isnull). sum). sort\_values (ascending=False)

total\_null.head (10)

loan\_train[‘ Gender'] = loan\_train|Gender']. fillna (loan\_train[ Gender'].dropna). mode(). values[0])

Loan\_train [‘Married'] = loan\_train Married']. fillna(loan\_train Married ].dropna().mode(). values[0] )

Loan\_ train[‘Dependents'] = loan\_train[‘Dependents'].fillna(Loan\_train[‘Dependents'].dropna().mode(). values [0])

loan\_train Self\_Employed] = loan\_train[‘Self\_Employed]. fillna(Loan\_train[‘ Self\_Employed’].dropna().mode().values[0])

Loan\_train| LoanAmount'] = loan\_train[‘LoanAmount'].fillna(loan\_train[‘LoanAmount'].dropna().mean())

Loan train[‘Loan\_Amount\_Term'] = loan\_train['Loan\_Amount\_Term'].fillna(Loan\_train[‘Loan\_Amount\_Term'].dropna). mode().values[0] )

Loan\_train [‘Credit\_History’] = loan\_train[‘Credit\_History’].fillna(

print (set (loan\_train[‘Gender']. values. tolist()))

print(set(loan\_train[‘Dependents']. values. tolist()))

print (set(loan\_train[‘Married']. values. tolist)))

print (set(loan\_train[‘Education']. values. tolist)))

print (set(loan\_train[ Self\_Employed']. values.tolist()))

print (set(loan\_train| Loan\_Status']-values.tolist)))

print (set (loan\_train[Property\_Area' ]-values. tolist)))

Loan\_train[‘ Loan\_Status'] = loan\_train['Loan\_Status'].map({'N' :0,'Y':1}).astype(int)

Loan\_ train = pd.get\_dummies(loan\_train, columns = ['Gender', "Dependents', 'Married', 'Education',

‘set employed’,’Property\_Area’ ])

standardScaler = StandardScaler()

columns\_to\_scale = ['ApplicantIncome', 'CoapplicantIncome’, 'LoanAmount', ‘Loan\_Amount\_Term']

Loan\_train[columns\_to\_scale] = standardScaler.fit\_transformloan\_train[columns\_to\_scale])

у = loan\_train['Loan\_Status']

Х = loan\_train.drop(['Loan\_Status', ‘Loan\_ID'], axis = 1)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.2, random\_state = 42)

#XGBOOST

gom\_param\_grid = {‘n\_estimators'; range (1,1000, 10),’max\_depth': range (1, 20), ‘learning\_rate': [.1,.4,

，45，.5，.55，.61, ‘colsample\_bytree': [.6, 7, 8, 9, 1],}

xgb\_classifier = XGBClassifter()

xgb\_random = RandomizedSearchCV(param\_distributions=gbm\_param\_grid, estimator = xgb\_classifter, scoring = "accuracy", verbose = 0, n\_iter = 100, cv = 4)

xgb\_random. fit(x\_train,y\_train)

print (f’ Best parameters: {xgb\_random. best\_params\_}’)

Y\_pred = xgb\_random. predict(X\_test)

print (f’Accuracy: {np.sum(y\_pred==y\_test)/len(y\_test)}’)

param\_grid = {‘max depth’ : range(4,25), ‘min samples\_leat’ : range(10,100, 10), ‘min samples split' : range(10,100, 10), ‘criterion' : ['gint', 'entropy'] }

n\_folds = 5

dt = DectsionTreeClassif ter(random\_state=np. random. rand int (0,100))

dt\_grid = GridSearchCV(dt, param\_grid, cv = n\_folds, return\_train\_score=True, verbose=0)

dt\_grid. fit(X\_train,y\_train)

print (dt\_grid. best\_params\_)

Y\_pred\_best=dt\_grid. predict(X\_test)

acc = metrics. accuracy\_score(y\_test,y\_pred\_best)

print(acc)

#0.7804878048780488

#Random Forest Classifier

rf\_param\_grid = {

n\_estimators': range (1,1000, 100),

rt = RandomForestClassifier()

rf\_random = RandomizedSearchCV(param\_distributions=rf\_param\_grid,estimator = rf, scoring = "accuracy", verbose = 0, n\_iter = 100, CV = 4)

rf\_random. fit(X\_train,y\_train)

best\_params = rf\_random. best\_params\_

print (f'Best parameters: {best\_params}')

y\_pred1 = rf\_random. predict (X\_test)

print (f’Accuracy: {np. sum(y\_pred1==y\_test)/len(y\_test)}')

# SUPPORT VECTOR CLASSIFIER

svm\_param\_grid = {‘kernel' : ['linear', ‘poly’, ‘rbf’, ‘signoid’], ‘C’: range(1, 11)}

svm = SVC()

svm\_randon = RandonizedSearchCV(paran\_distributions-svm\_paran\_grid, estimator = svm, scoring = "accuracy", verbose = 0, n\_iter = 100, cv = 4)

svm\_ random. fit (Xtrain, y\_train)

best\_params = sva\_random.best\_params\_

print (f'Best parameters: {best\_params}')

y\_pred\_best=svm\_randon.predict(Xtest)

acc = metrics. accuracy\_score\_test, y\_pred\_best)

print(acc)

def feature\_imp(df, model):

feat = pd.DataFrame(columns=|'feature', 'importance' 1)

feat["feature"] = df.columns

feat|"importance"| = model. best\_estimator\_. feature\_importances\_

return feat. sort\_values (by=" importance", ascending=False)