```
##########################################
#                                        #
#              main.py                   #
#                                        #
##########################################

import pandas as pd
import numpy as np
from utils import *
import argparse
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
import json

#This is a particular class to include dictionaries in the parsers
class StoreDictKeyPair(argparse.Action):
    def __init__(self, option_strings, dest, nargs=None, **kwargs):
        self._nargs = nargs
        super(StoreDictKeyPair, self).__init__(option_strings, dest, nargs=nargs, **kwargs)
    def __call__(self, parser, namespace, values, option_string=None):
        my_dict = {}
        for kv in values:
            k,v = kv.split("=")
            my_dict[k] = v
        setattr(namespace, self.dest, my_dict)


# All the args arguments:
parser = argparse.ArgumentParser(description='Main-Code')
parser.add_argument('--model', '-m', type=str, default='SVC',help = "model to run on the dataset, options are \n{'SVC',
'LogisticRegression', 'SGDClassifier', 'RandomForestClassifier', 'AdaBoostClassifier','DecisionTreeClassifier}") # model choice
parser.add_argument('--dataset','-d', type=str, default='kidney', help="dataset type, options are {'kidney','banknote'}, default is
'kidney'") # dataset choice
parser.add_argument("--parameters","-p" , dest="my_dict", action=StoreDictKeyPair, nargs="+",
metavar="KEY=VAL",help="model paramaters, dictionary of the parameters and their values, example for SVC : -p kernel=rbf
C=10 degree=2 gamma=auto") # parameters choice, it's a dictionary
parser.add_argument('--pca', type=bool, default=True, help="Apply PCA, if True applies PCA algorithm") #Choice to keep the
original features or to use the PCA coefficients instead
parser.add_argument('-v', '--cut_off_variance',type=float, default=0.95, help="PCA cut-off variance or the threshold ratio of
explained variance. Default is 0.95") #Ratio threshold of variance explained


data_paths = {'kidney':'../data/kidney_disease/kidney_disease.csv',
          'banknote':'../data/banknote/data_banknote_authentication.txt'} # Dictionary of datasets paths

# Dictionary of models and their corresponding parameters for the gridsearch
models_dict = {'SVC':{'model':SVC, #Support vector Classifier
                 'parameters':{'kernel':['linear', 'rbf', 'sigmoid', 'poly'],
                               'C'     :[1, 10],
                               'degree': [2, 3],
                               'gamma' : ['scale', 'auto']
                              }
                },
           'LogisticRegression':{'model':LogisticRegression, # Logistic Regression
                         'parameters':{'C':[1, 10],
                               'fit_intercept' : [True,False],
                               'intercept_scaling' : [1,10],
                                }
                         },
            'SGDClassifier':{'model':SGDClassifier, # Stochastic gradient descent classifier
                         'parameters':{'loss':['hinge', 'log', 'modified_huber', 'squared_hinge', 'perceptron'],
                               'penalty':['l1', 'l2'],
```

```
                                'fit_intercept' : [True,False],
                                }
                        },
            'DecisionTreeClassifier':{'model':RandomForestClassifier,
                        'parameters':{'criterion':['gini', 'entropy'],
                                'max_depth' : [2,5,10,None],
                                }
                        },
            'AdaBoostClassifier':{'model':AdaBoostClassifier,
                        'parameters':{'n_estimators':[50, 100, 150],
                                'algorithm':['SAMME', 'SAMME.R'],
                                'learning_rate' : [0.1,0.5,1]
                                }
                        },
            'RandomForestClassifier':{'model':RandomForestClassifier,
                        'parameters':{'n_estimators':[50, 100, 150],
                                'criterion':['gini', 'entropy'],
                                'max_depth' : [2,5,10,None],
                                'bootstrap' : [True,False],
                                }
                        }
            }
def main():
    """
    @author : Yassir BENDOU
    main function of the program
    """
    args = parser.parse_args()  #Initiate the parser
    data_str  = args.dataset    # dataset choice
    model_str = args.model      # model choice
    params    = args.my_dict    # get the parameters dictionary provided by the user
    print('params',params)
    print(f'{data_str} dataset : Extracting data...')
    data = pd.read_csv(data_paths[data_str]) # Fetch the data
    print('Cleaning data')
    data = clean_noisy_data(data,classes = 2)  # Clean the dataset
    num_variables, categ_variables = detect_type(data.iloc[:,:-1]) # Detect numerical and categorical data
    data.iloc[:,:-1] = replace_missing(data.iloc[:,:-1], num_variables, categ_variables) # Replace missing values
    data.iloc[:,:-1] = center_encode(data, num_variables, categ_variables)  # scale values and ordinal ordinalencode them

    if args.pca:
        dataset = feature_selection(data,cut_off_variance=args.cut_off_variance)  # Compress the data using PCA

    X,y = dataset.values[:,:-1],dataset.values[:,-1]
    X_train, X_test, y_train, y_test = split_data(X,y)

    model_choice = models_dict[model_str]
    model = model_choice['model']
    if params == None : # If the user doesn't provide any parameters for the model, we run the gridsearch and pick the best ones
        print(f'Chosen model and its different possible parameters: {model_choice}')
        print(f'running grid search to find the best parameters for the {data_str} dataset')
        parameters = model_choice['parameters']
        params,best_score = select_params(model, parameters, X_train, y_train)
        print(f'Best parameters for {model_str} are {params}. \nBest Score is {best_score}')

    print(f'Training with parameters : {params}')
    score_val,clf = training(model, params, X_train, y_train) # Train the model
    print(f'Cross Validation Score : {np.round(100*score_val,2)}%')
    score_test = test_evaluate(clf,X_test,y_test) # Run the model on test data
    print(f'F1 score on test data : {np.round(100*score_test,2)}%')

if __name__ == '__main__':
    main()
```