**CODE:**

import numpy as np

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

from sklearn.linear\_model import LinearRegression

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVR

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor

from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor

from sklearn.cluster import AgglomerativeClustering

from sklearn.decomposition import PCA

from sklearn.manifold import TSNE

from sklearn.covariance import EllipticEnvelope

from sklearn.ensemble import IsolationForest

from sklearn.neighbors import LocalOutlierFactor

from sklearn.metrics import accuracy\_score, mean\_squared\_error

import warnings

warnings.filterwarnings("ignore")

def ml\_function(X, y, task, algorithm, hyperparameters):

    if task == "classification":

        X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

        if algorithm == "LogisticRegression":

            model = LogisticRegression(\*\*hyperparameters)

        elif algorithm == "SVC":

            model = SVC(\*\*hyperparameters)

        elif algorithm == "RandomForestClassifier":

            model = RandomForestClassifier(\*\*hyperparameters)

        elif algorithm == "DecisionTreeClassifier":

            model = DecisionTreeClassifier(\*\*hyperparameters)

        elif algorithm == "KNeighborsClassifier":

            model = KNeighborsClassifier(\*\*hyperparameters)

        else:

            raise ValueError("Invalid algorithm specified for classification task.")

        model.fit(X\_train, y\_train)

        y\_pred = model.predict(X\_test)

        print("Accuracy:", accuracy\_score(y\_test, y\_pred))

    elif task == "regression":

        if algorithm == "LinearRegression":

            model = LinearRegression(\*\*hyperparameters)

        elif algorithm == "SVR":

            model = SVR(\*\*hyperparameters)

        elif algorithm == "DecisionTreeRegressor":

            model = DecisionTreeRegressor(\*\*hyperparameters)

        elif algorithm == "KNeighborsRegressor":

            model = KNeighborsRegressor(\*\*hyperparameters)

        else:

            raise ValueError("Invalid algorithm specified for regression task.")

        model.fit(X, y)

        y\_pred = model.predict(X)

        print("MSE:", mean\_squared\_error(y, y\_pred))

    elif task == "clustering":

        if algorithm == "KMeans":

            model = KMeans(\*\*hyperparameters)

        elif algorithm == "AgglomerativeClustering":

            model = AgglomerativeClustering(\*\*hyperparameters)

        else:

            raise ValueError("Invalid algorithm specified for clustering task.")

        model.fit(X)

        print("MSE:", mean\_squared\_error(X, model.predict(X)))

    elif task == "dimensionality\_reduction":

        if algorithm == "PCA":

            model = PCA(\*\*hyperparameters)

        elif algorithm == "tSNE":

            model = TSNE(\*\*hyperparameters)

        else:

            raise ValueError("Invalid algorithm specified for dimensionality reduction task.")

        X\_reduced = model.fit\_transform(X)

        print("New shape of X:", X\_reduced.shape)

    elif task == "anomaly\_detection":

        if algorithm == "EllipticEnvelope":

            model = EllipticEnvelope(\*\*hyperparameters)

        elif algorithm == "IsolationForest":

            model = IsolationForest(\*\*hyperparameters)

        elif algorithm == "LocalOutlierFactor":

            model = LocalOutlierFactor(\*\*hyperparameters)

        else:

            raise ValueError("Invalid algorithm specified for anomaly detection task.")

        y\_pred = model.fit\_predict(X)

        print("Number of anomalies:", np.sum(y\_pred == -1))

    else:

        raise ValueError("Invalid task specified.")

    return model