Team members: Arsany Osama (2205122) , Mark Magdy (2205040)

K-Nearest Neighbors (KNN) Classification Report

This report provides an overview of the KNN classification code and includes relevant information about the steps involved. Each section is accompanied by explanatory comments to help understand the code.

1. Data Loading and Preprocessing

The code begins by importing the necessary libraries and loading the dataset from a CSV file (`magic04.csv`) using pandas. The dataset is then displayed using `print(data)`.

2. Column Renaming

The column names of the dataset are set using the `data.columns` attribute. The column names are assigned as `['f1', 'f2', 'f3', 'f4', 'f5', 'f6', 'f7', 'f8', 'f9', 'f10', 'class']`. The dataset with renamed columns is displayed using `print("Data after renaming columns:\n",data)`.

3. Handling Missing Values

The code removes any rows with missing values using `data.dropna(inplace=True)`.

4. Data Scaling

The features of the dataset are then scaled using the `StandardScaler` from `sklearn.preprocessing`. The `fit_transform` method is applied to the feature columns (`f1` to `f10`) to standardize their values. The scaled dataset is displayed using `print("Data after scaling:\n",data)`.

5. Class Balancing

The dataset is balanced by randomly sampling an equal number of instances for each class. The gamma class (`'g'`) and hadron class (`'h'`) are separated into two different dataframes (`gamma_data` and `hadron_data`). The gamma class is then randomly sampled (`gamma_data_equal`) to match the size of the hadron class. The balanced dataset is created by concatenating `gamma_data_equal` and `hadron_data`, resulting in the `balanced_data` dataframe. The balanced dataset is displayed using `print("Data after balancing:\n",balanced_data)`.

6. Train-Validation-Test Split

The balanced dataset is split into training, validation, and testing sets using the `train_test_split` function from `sklearn.model_selection`. The `test_data` is extracted by specifying a test size of 15% (`test_size=0.15`). The remaining data is further split into `train_data` and `val_data` with a test size of 15% (`test_size=0.15`). The lengths of the resulting sets are displayed using `print("Testing: ",len(test_data))`, `print("Validation: ",len(val_data))`, and `print("Training: ",len(train_data))`.

7. **Model Tuning**

The code proceeds to tune the model parameters to find the best performing K value. A list of K values (`k_values`) is provided, and for each value, the KNN classifier is trained on the training set and evaluated on the validation set. The accuracy score is computed using `accuracy_score`. The best K value and its corresponding accuracy are tracked and updated if a higher accuracy is achieved. The best K value is displayed using `print("Best k value:", best_k)`.

8. **Training the Best Model**

The KNN classifier is trained on the combined training and validation sets using the best K value (`best_k`). The concatenated training and validation feature sets (`pd.concat([X_train, X_val])`) and label sets (`pd.concat([y_train, y_val])`) are used for training.

9. **Evaluation on Testing Set**

The best model is evaluated on the testing set. The predicted labels (`y_pred`) are computed using the trained KNN classifier. Several evaluation metrics, including accuracy, precision, recall, F1-score, and confusion matrix, are calculated using functions from `sklearn.metrics`. The evaluation results are displayed using `print`.

10. **Prediction on Random Data**

The code randomly selects three data points from the original dataset (`random_data = data.sample(n=3)`). The features of the random data are prepared for prediction (`X_random = random_data.drop('class', axis=1)`). The KNN classifier predicts the labels for the random data points, and the predicted labels are converted to class labels. The actual labels for the random data points are retrieved. Finally, the predicted and actual labels for each random data point, along with the prediction correctness, are printed.

Regression Report

1. Dataset Overview

The code begins by reading the dataset from the file "California_Houses.csv" using the `pd.read_csv()` function from the pandas library. The dataset is then printed using the `print(data)` statement.

Next, some information about the dataset is provided. The code prints the following:

- Dataset information using the `data.info()` function.

- Descriptive statistics of the dataset using the `data.describe()` function.

2. Data Cleaning

The code performs some data cleaning operations:

⇨ **Null Values:**

The code checks if there are any null values in the dataset using the `data.isnull().sum()` function. The number of null values for each column is printed using `print(data.isnull().sum())`. It is determined that there are no null values in the dataset.

⇨ **Duplicates:**

The code checks if there are any duplicated values in the dataset using the `data.duplicated()` function. The resulting boolean array indicating whether each row is a duplicate is printed using `print(data.duplicated())`. The total number of duplicated rows is printed using `print(data.duplicated().sum())`. It is determined that there are no duplicate values in the dataset.

3. Data Splitting

The code performs linear regression using the scikit-learn library.

Then splits the dataset into training, validation, and testing sets using the `train_test_split()` function. The training set contains 70% of the data, while the validation and testing sets each contain 15% of the data. The random_state parameter is set to 42 for reproducibility.

The resulting datasets are printed using the following statements:

- Training set: `(x_train.shape, y_train.shape)`

- Validation set: `(x_val.shape, y_val.shape)`

- Test set: `(x_test.shape, y_test.shape)`

4. Model Training

A linear regression model is created using `LinearRegression()` and trained on the training set using the `fit()` function. The trained model is stored in the `model` variable.

5. Model Evaluation (Validation)

The code predicts the target variable (`y_val_predict`) for the validation set using the trained model. The predicted values are printed using `print(y_val_predict)`.

The mean squared error (MSE) between the ground truth (`y_val`) and the predicted values (`y_val_predict`) is calculated using `mean_squared_error()` and stored in the `mse_val` variable. The MSE value is printed using `print(f"Mean Square Error(Validation): {mse_val}")`.

6. Model Testing

The code predicts the target variable (`y_test_predict`) for the testing set using the trained model. The predicted values are printed using `print(y_test_predict)`.

The score of the model on the testing set is calculated using the `score()` function, which measures the coefficient of determination (R^2) of the prediction. The score is printed using `print("Score: ", model.score(x_test, y_test))`.

The MSE between the ground truth (`y_test`) and the predicted values (`y_test_predict`) is calculated and stored in the `mse_test` variable. The MSE value is printed using `print(f"Mean Square Error(Test): {mse_test}")`.

7. Model Application

The code demonstrates applying the trained linear regression model to a new data point (`df`). The feature values of the data point are defined in a dictionary (`dic`) and converted to a DataFrame.

The model predicts the target variable (`y`) for the new data point using `model.predict(df)`. The predicted value is printed using `print("Result (Linear Regression): ", y)`.

The code performs Lasso regression using the scikit-learn library.

9. Model Training

A Lasso regression model is created using `Lasso(max_iter=5000)` and trained on the training set using the `fit()` function. The trained model is stored in the `model` variable.

10. Model Evaluation (Validation)

The code predicts the target variable (`y_val_predict`) for the validation set using the trained model. The predicted values are printed using `print(y_val_predict)`.

The mean squared error (MSE) between the ground truth (`y_val`) and the predicted values (`y_val_predict`) is calculated and stored in the `mse_val` variable. The MSE value is printed using `print(f"Mean Square Error(Validation): {mse_val}")`.

11. Model Testing

The code predicts the target variable (`y_test_predict`) for the testing set using the trained model. The predicted values are printed using `print(y_test_predict)`.

The MSE between the ground truth (`y_test`) and the predicted values (`y_test_predict`) is calculated and stored in the `mse_test` variable. The MSE value is printed using `print(f"Mean Square Error(Test): {mse_test}")`.

12. Model Application

The code demonstrates applying the trained Lasso regression model to the new data point (`df`). The model predicts the target variable (`y`) for the new data point using `model.predict(df)`. The predicted value is printed using `print("Result (Lasso Regression): ", y)`.