Assignment 1

Classification and Regression

Prepared by:

|  |  |
| --- | --- |
| 7818 | Mohamed Hussein Elzohiry |
| 7721 | Alaa Mohamed Elkhouly |

**SUBJECT: Machine Learning PROFFESSOR: DR. Marwan Torki**

Problem Statement 1 (Classification):



Code’s flow:

Code Explanation:

**Lines 1 to 9:**

Imports necessary libraries

* **NumPy** & **Pandas** 🡪 data manipulation and handling.
* From **sklearn.model\_selection**:
  + **train\_test\_split 🡪** split the data into training, validation, and test sets
  + **KNeighborsClassifier 🡪** apply the K-Nearest Neighbors (K-NN) classification algorithm.
  + **Others 🡪** evaluating the model's performance

**Lines 10 to 12:**

Set a random seed for reproducibility

* Setting a random seed ensures reproducibility, so running the code multiple times will yield the same results for random sampling or splits.

**Lines 13 to 18:**

Load the dataset and replace 'magic\_gamma\_data.csv' with your actual dataset path.

**Lines 19 to 21:**

Check class distribution to identify imbalance

* checks the number of instances for each class (gamma and hadron) in the dataset.
* the output will show how many examples belong to each class (For the imbalance).

**Lines 22 to 26:**

* Balance the classes
* Separate the two classes: divides dataset into two subsets:
  + **gamma\_class** for the 'g' class
  + **hadron\_class** for the 'h' class

**Lines 27 to 32:**

Downsample the gamma class to match hadron class size:

* uses the sample method to randomly select a subset of gamma events equal in size to the number of hadron events, achieving balance.

Combine the balanced dataset

* creates balanced dataset “**balanced\_data**” by concatenating the downsampled gamma events and all hadron events.

**Lines 33 to 36:**

Split data into features and labels

* x contains the features (all columns except for class).
* y contains the labels (the class column).

**Lines 37 to 40:**

Split dataset into training, validation, and testing sets (70%, 15%, 15%)

* train\_test\_split first divides the data into:
  + training (70%)
  + temporary (X\_temp and y\_temp) (30%) sets.
* temporary set is split further into validation and testing sets, each 15% of the original data.
* The final split results in 70% for training, 15% for validation, and 15% for testing

**Lines 41 to 43:**

* Initializes an empty dictionary to store model’s performance metrics for each value of k tested
* allows comparison across different k values to find the optimal one

**Lines 44 to 70:**

Iterate through different values of k to find the best one

* Initialize the K-NN classifier with the current k value
* A loop iterates over k values from 1 to 10 to testing each as a possible number of neighbors for the K-NN classifier

For each k:

* A K-NN classifier is initialized with n\_neighbors=k.
* Train the model on the training set
* Validate the model (predictions are made) on the validation data
* Calculate and store performance metrics (accuracy, precision, recall, F1 score, and confusion matrix)
* Store metrics in performance\_metrics dictionary under the key for each k value.

**Lines 71 to 74:**

Find the best k based on validation F1 score

* selects the k value with the highest **F1 score** from the validation results stored in performance\_metrics, providing the optimal k for balanced performance across precision and recall.

**Lines 75 to 79:**

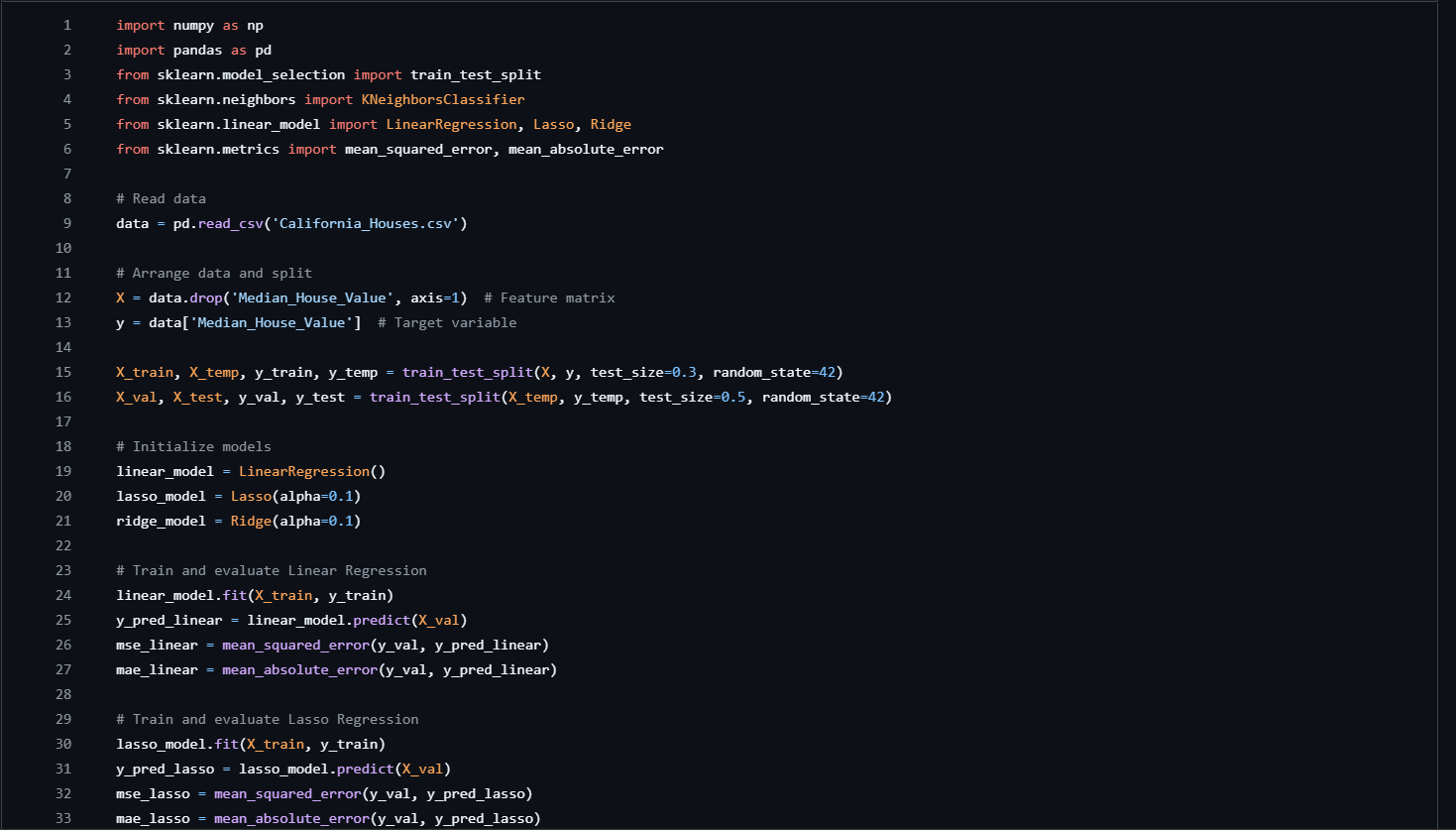
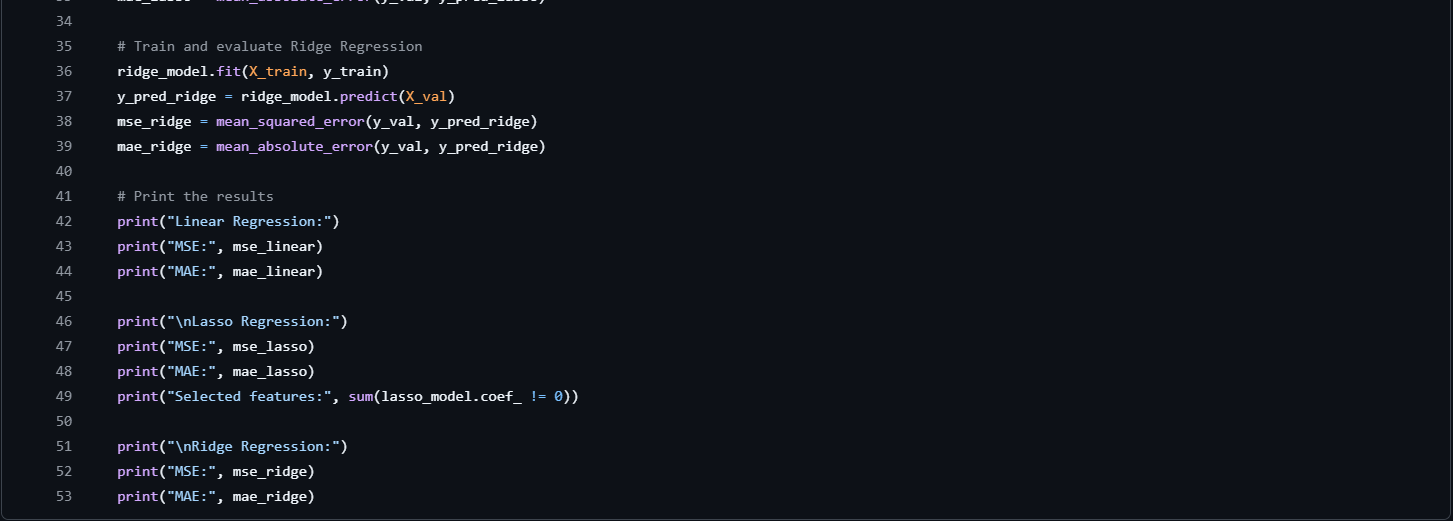
Retrain the best model on the training set and evaluate on the test set

* a **K-NN model** is retrained using this best k on the **entire training set**.
* Predictions are then made on the test set to assess the final model’s performance.

**Lines 80 to 86:**

Report test performance metrics

* Calculate the performance metrics on the test set
* These metrics reflect how well the model generalizes to unseen data.
* Problem Statement 2 (Regression):



Code Explanation:

**Lines 1 to 7:**

Imports necessary libraries

* **numpy** 🡪 for handling large arrays and numerical computations.
* **pandas** 🡪for data manipulation and analysis (provides the Data Frame structure for easy handling of datasets).
* From **sklearn.model\_selection**:
  + **train\_test\_split 🡪** split the data into training, validation, and test sets
  + **KNeighborsClassifier 🡪**  Although imported here, it’s not used in this code. It’s used for classification tasks rather than regression and could be removed. classification algorithm.
  + **LinearRegression, Lasso, Ridge 🡪** regression models
    - **LinearRegression 🡪** performs basic linear regression
    - **Lasso 🡪** Least Absolute Shrinkage and Selection Operator is a regression technique that adds regularization (penalty) to reduce model complexity by eliminating certain features
    - **Ridge 🡪** another regularization method that penalizes large coefficients
  + **mean\_squared\_error & mean\_absolute\_error 🡪** evaluating the model's accuracy by measuring the prediction error

**Lines 8 to 10:**

Reads data

loads the dataset from a CSV file named California\_Houses.csv into a Data Frame called data

**Lines 11 to 14:**

Splitting Features and Target Variable

* x: All columns except Median\_House\_Value are treated as features (independent variables).
* y: Median\_House\_Value is selected as the target variable (dependent variable) since it’s what the model aims to predict.

**Lines 15 to 17:**

Data splitting

* splits the data into training (70%) and temporary (30%) sets.
* The random\_state=42 ensures reproducibility by using the same random seed each time.
* temporary set is further split into validation and test sets, each with 15% of the total data, for validating and testing the model.
* Validation is used for tuning, and the test set is used to evaluate the model's performance on unseen data.

**Lines 18 to 22:**

Initialize models

* linear\_model: Basic linear regression model.
* lasso\_model: Lasso regression with a regularization strength (alpha) of 0.1, which adds a penalty to large coefficients, encouraging some to become zero and performing feature selection.
* ridge\_model: Ridge regression with a regularization strength (alpha) of 0.1, which penalizes large coefficients, reducing overfitting but without setting coefficients to zero.

**Lines 23 to 28:**

Train and evaluate Linear Regression

 **linear\_model.fit(X\_train, y\_train)**: Trains the model on the training data.

 **y\_pred\_linear = linear\_model.predict(X\_val)**: Predicts house values on the validation set.

 **mse\_linear and mae\_linear**: Compute the MSE and MAE of predictions against the actual values in the validation set, measuring model accuracy. Lower values indicate a better fit:

* **MSE** (Mean Squared Error): Measures average squared differences between predicted and actual values.
* **MAE** (Mean Absolute Error): Measures the average absolute difference between predicted and actual values, less sensitive to outliers.

**Lines 29 to 34:**

Train and evaluate Lasso Regression

 **lasso\_model.fit(X\_train, y\_train)**: Trains the Lasso regression model on the training data.

 **y\_pred\_lasso = lasso\_model.predict(X\_val)**: Predicts house values on the validation set using Lasso.

 **mse\_lasso and mae\_lasso**: Compute the error metrics for Lasso predictions. The Lasso model can drive some coefficients to zero, effectively excluding some features.

**Lines 35 to 40:**

Train and evaluate Ridge Regression

 **ridge\_model.fit(X\_train, y\_train)**: Trains the Ridge regression model.

 **y\_pred\_ridge = ridge\_model.predict(X\_val)**: Predicts house values on the validation set using Ridge regression.

 **mse\_ridge and mae\_ridge**: Calculate error metrics, similar to the previous models. Ridge penalizes large coefficients, reducing their magnitude but not setting them to zero.

**Lines 41 to 53:**

**Print the results**

 The **print statements** output the performance metrics (MSE and MAE) for each model.

 **sum(lasso\_model.coef\_ != 0)**: This counts the number of non-zero coefficients in the Lasso model, showing how many features Lasso considered important.