Mälardalen University

Course: DVA262 — Machine Learning Concepts

Objective: The purpose of the Lab is to apply your understanding of the Decision Tree algorithm by implementing the Decision Tree algorithm from scratch in the C++ programming using the Lab skeleton.

Lab 2: The Decision Tree Algorithm Implementation

Execution and demonstration

This lab is executed a group of two students or individually. The completed lab should be demonstrated to a lab assistant.

Useful aids

- The course literature
- Lecture slides and notes
- Google & Internet

Part 1: Decision Tree for Classification

You will implement the Decision Tree algorithm for the classification problem in this task.

Dataset: The classification problem will be based on the **Iris dataset,** which is already available in the project folder.

The Visual Interface: The main form (i.e., MainForm.h) calls all the class objects and functions that are already implemented in the skeleton. However, you may need to edit some parts of the code to make it consistent with your code to display the results correctly.

- *Loading and splitting the dataset*
  The code for loading and preparing the dataset is already implemented, so no additional code is required for this part. Use the following function in your code:

  - You can use the function loadAndPreprocessDataset() to prepare the dataset, which is located in the class name DataLoader.cpp under DataUtils folder.

- Use the splitDataset() function in the class name DataPreprocessor.cpp under the DataUtils folder to split the dataset into training and test sets.

Your Task: You will mainly work with two classes: DecisionTreeClassification.cpp and EntropyFunctions.cpp located under the Classification and Utils folder, respectively.

- *Creating class member functions*

The DecisionTreeClassification.cpp class may contain some or all of the following member functions:

- Constructor: initialises the parameters for DT.
- fit() function: assigns training dataset and labels.
- growTree() function: this is primarily function to build decision tree from training data.
- informationGain() function: to measure the quality of a split.
- mostCommonLable() function: to find most common lable from the Label set.
- predict() function: given the test data points this function predicts the classes.
- traverseTree() function: used in the predict() function to traverse the tree for making prediction.

Task 1: In this Lab work you must complete the Decision Tree implementation by completing the growTree(), informationGain(), mostCommonLable(), predict(), and traverseTree() functions.

Use the Node function to construct the feature, threshold, left and right nodes, and value and use the isLeafNode function to check if the node is a leaf node. These functions are in the class name Node.cpp under the Utils folder.

The EntropyFunctions.cpp class contains functions for entropy measuring which are not implemented.

Task 2: You must complete the two entropy() functions to compute the supported criteria as Shannon information gain.

- *Evaluating*

When you are done with the implementation it is time to evaluate your algorithm by measuring accuracy, precision, recall and F1 score. The code for these metrics is already done, so no additional code is required.

- You can use accuracy(), precision(), recall() and f1Score() functions, which are located in the class name Metrics.cpp under the Evaluation folder for this evaluation.

Task 3: What value of *min_samples_split*, *max_depth* and *n_feats* have you used? Try different values for these parameters. What are optimal values that give you best results? Discuss your findings during the demonstration.

Note: You can change the values that are passed as parameters in the DecisionTreeClassification() function definition at the DecisionTreeClassification.h file.

- *Visualisation*

A visual representation (for training and test sets results) is needed. However, the code for this task is already done, so use the following functions:

- Use the confusionMatrix() function located in the class name Metrics.cpp under the Evaluation folder, to calculate the confusion matrix for the specified number of class labels.
- Use the DisplayConfusionMatrixInGrid() function located in the MainForm.h to display the confusion matrix in the grid.

An example of the result from the Decision Tree classification is shown in the Figure 1. The visual interface is same for all classifiers.

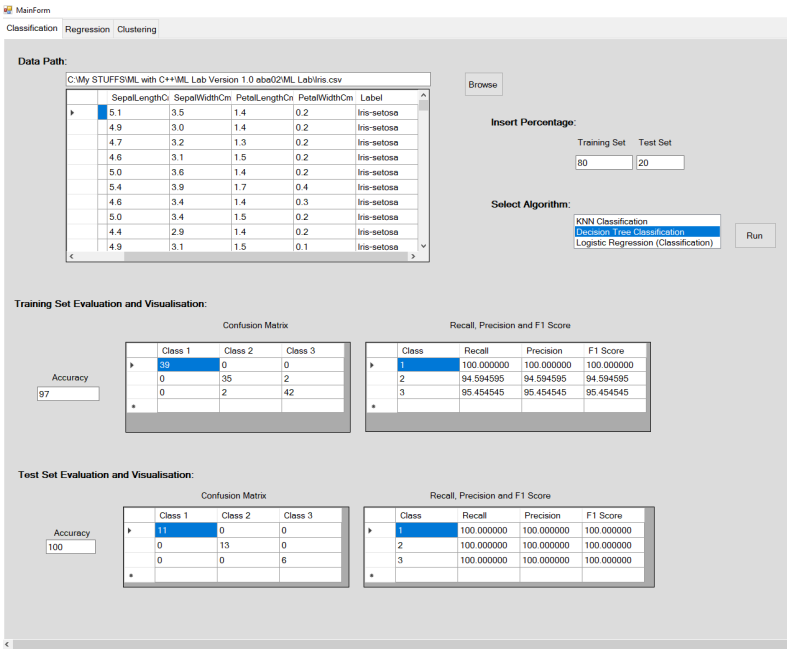

Figure 1: Example result of Decision Tree classification in the visual interface.

Part 2: Decision Tree for Regression

In this task, you will implement the Decision Tree algorithm for the regression problem.

Dataset

The regression problem will be based on the Boston Housing dataset, which is already available in the project folder.

The Visual Interface: Similar to the classification task, the main form (i.e., MainForm.h) calls all the class objects and functions that are already implemented. But you may need to edit some parts of the code to make it consistent with your code to display the results correctly.

Your Task: You will mainly work with the DecisionTreeRegression.cpp class located under the Regression folder.

- *Loading and splitting the dataset*

The code for loading and preparing (preprocessing) the dataset is already done, so no additional code is required for this part. Use the following function in your code:

- You can use the function readDatasetFromFilePath() to prepare the dataset, which is located in the class name DataLoader.cpp under DataUtils folder.
- Use the splitDataset() function in the class name DataPreprocessor.cpp under the DataUtils folder, to split the dataset into training and test sets.

- Creating class member functions

The DecisionTreeRegression.cpp class may have some or all of the following member functions:

- Constructor: initialises the parameters for DT.
- fit() function: assigns training dataset and labels.
- growTree() function: this is primarily function to build decision tree from training data.
- meanSquaredError() function: to measure the quality of a split. This function is used in growTree() function as supported criteria to find the best split and minimizes the L2 loss.
- mean() function: to calculate the mean as the predicted regression value.

- predict() function: given the test data points this function predicts the regression value.
- traverseTree() function: used in the predict() function traverse the tree to making prediction.

Task 4: In this part of the work you must complete the Decision Tree implementation by completing the growTree(), meanSquaredError(), mean(), predict(), and traverseTree() functions.

Same as DecisionTreeClassification.cpp, you need to use the Node.cpp class.

- *Evaluating*

Evaluating the regression model's performance using mean Absolute Error (MAE), Root Mean Squared Error (RMSE) and R Squared. The code for these metrics is already done, so no additional code is required.

- You can use meanAbsoluteError(), rootMeanSquaredError(), and rSquared() functions, which are located in the class name Metrics under the Evaluation folder for this evaluation.

Task 5: What values of *min_samples_split*, *max_depth* and *n_feats* have you used? Try different values for these parameters. What are optimal values that give you the best results? Discuss your findings during the demonstration.

Note: You can change the values that are passed as parameters in the DecisionTreeRegression() function definition at the DecisionTreeRegression.h file.

- *Visualisation*

A visual representation (for training and test data sets) is needed. The code for this task is already done, so use the following functions for that:

- Use the VisualizeParityPlot() function, located in the MainForm.h, to display the parity plot for the actual and predicted test labels.

An example of the result from the Decision regression is shown in the Figure 2. The visual interface is same for all regression algorithms.

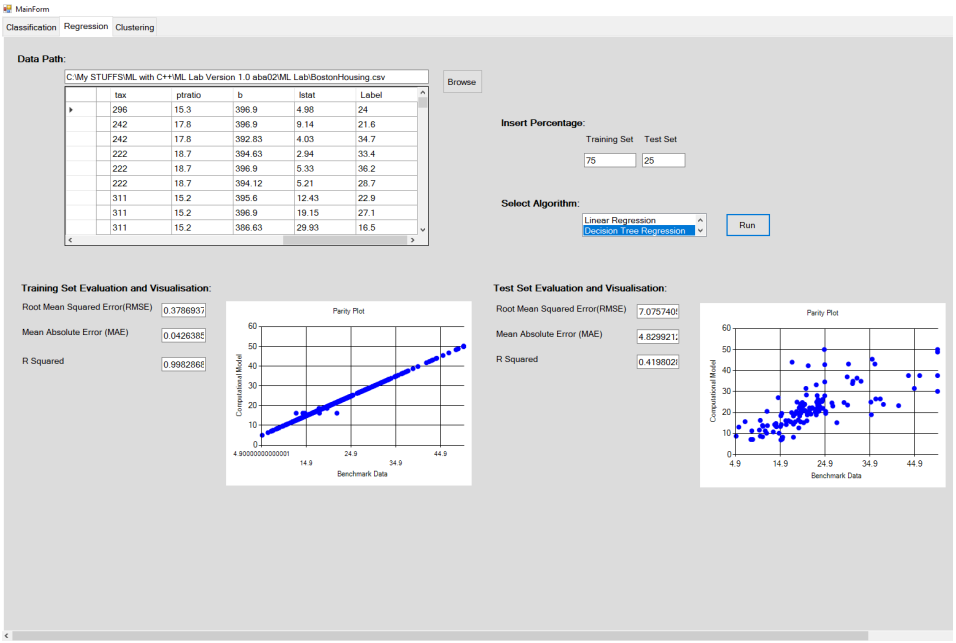

Figure 2: Example results of Decision Tree Regression in the visual interface.

Some Hints:

- Include the necessary header files at the beginning of your code.
- For regression you need to calculate the best split based on mean squared error not information gain.
- Use the entropy function for the information gain, which is in the class name EntropyFunctions.cpp under the Utils folder.