Project 1 Report

Sander Marx and Kornél Papp

The code can be found in the project1.ipynb jupyter notebook file. We worked together throughout the project sharing ideas and giving feedback to each other. Sander focused more on the implementation of the pruning and Kornél spent more time on the model selection, evaluation and report.

Task 1.1, 1.2

We started the project with the object-oriented programming paradigm in mind, creating two classes: Node and Dtree. The instance of Dtree represents the decision tree, which is built from instances of the Node class. A node can be either a decision or leaf node.

The Dtree class has the following methods:

- Set_root(root) – Sets the root of the tree
- entropy(y) – Calculates and returns the entropy impurity
- gini(y) – Calculates and returns the gini impurity
- make_tree(dataset, is_gini) – Builds a decision tree based on either the GINI index or entropy impurity (depends on the is_gini parameter)
- most_common(lst) – Finds the most common value for label or majority label
- best_split(X, y, is_gini) – Returns the best split based on either the GINI index or entropy impurity (depends on the is_gini parameter)
- make_predictions(x, node) – Checks the thresholds and the feature value and makes a prediction based on them recursively
- predict(y) – Returns an array of predictions by calling the make_predictions function starting with the root and moving down.
- print_tree(node, spacing) – Prints the tree with indentations (for development purposes)

Helper function related to task 1.1, 1.2:

learn(X,y, impurity_measure, prune, prune_data_ratio) – This function returns a decision tree based on features X and labels y. The impurity measure can be either entropy or gini, and the pruning can be false or true. If it is true, we prune the tree with the prune_tree(tree, X_p, Y_p, node) function while taking an amount of the training data as prune data, which corresponds with the prune_data_ratio.

Task 1.3

Helper function related to task 1.3:

Prune_tree(tree, X_p, Y_p, node) –This recursive function takes the tree to be pruned and its pruning data, then it prunes the decision tree by replacing subtrees with leaf nodes that have the majority label, provided that the change maintains or improves the model's accuracy on the given pruning validation set X_p, Y_p. It is used in the learn(X,y, impurity_measure, prune) function, explained under task 1.2, 1.3.

Task 1.4

Because the dataset contains the same number of data points with label 0 and 1, it is very balanced. We used accuracy as our performance measure, as it is very intuitive, easy to interpret and works well with balanced classes. Accuracy is the ratio between correctly labeled points and all points.

We separated the data into 3 parts: training data (80%), validation data (10%) and test data (10%). The training data was used for teaching the model, which could then be further divided into two parts for the pruning (20% of training data). The validation data was used to select the best setting between:

- entropy without pruning
- entropy with pruning
- gini without pruning
- gini with pruning

For this we simply calculated and printed the accuracy of each tree on the validation data. The selected setting happened to be entropy with pruning, with the highest validation accuracy of 86.875%.

Finally, to evaluate the model we calculated its accuracy on the test dataset, which was 92.1875%. This gives an estimate on how often the model guesses a label correctly on unseen data.

For the sake of completeness, we printed the accuracies on the training data as well.

Helper functions related to task 1.4:

measure_accuracy(tree, measure, is_pruning, data_type, X, Y) – Using the sklearn.metrics accuracy_score function it prints and returns the accuracy of the decision tree.

get_best_setting(settings) – By providing the different settings with their accuracy measures (on the validation data) it prints and returns the best setting among them.

evaluate_best_setting(best_setting, gini_tree, gini_tree_pr, entropy_tree, entropy_tree_pr) – This function is used for evaluating the best setting by measuring its accuracy on the test data and printing out the results.

1.5

There are certainly optimizations that our implementation lacks, as DecisionTreeClassifier from sklearn runs in ~0.0001 second, while our implementation needs around 1.8 seconds.

As for the accuracy they had very similar results. Both sklearn DecisionTreeClassifier had 91.875% accuracy on test data, and our implementation scored 92.1875%. We made sure to use the same test and training data by providing the same random_state (2).