Implement Decision Trees From Scratch

Noboru Hayashi NHAYASH3@JHU.EDU

1. Introduction

This paper describes the process to build decision tree models from scratch using Python to perform classification and regression. Specifically with the algorithms of ID3 (Iterative Dichotomiser 3) and CART (Classification and Regression Trees), decision tree models are built. Along with the implementation of these machine learning algorithms, the demonstration of classification and regression with the actual datasets will be shown as well.

Also the associated tuning process for ID3 and CART, respectively Pruning and Early-Stopping, are demonstrated with a tuning dataset extracted by 5-fold approach. And as the measurement of the performance, average accuracies and MSEs for 5-Fold Cross Validations are picked for classification and regression tasks.

2. Implementation

2.1. ID3 (ID3.py)

2.1.1 Node class

A data structure to store the corresponding value, next node pointer, children pointer, isPruned identifier, label value after pruning, and the tree depth.

2.1.2 ID 3 Classifier Class

A Classifier class stores the source data and corresponding labels, a list of feature names. After initialization, the label set and the counts of labels are stored, and based on the label data, entropy is calculated with the use of _calculate_entropy()

2.1.3 _calculate_entropy()

Given the IDs of data to scan, this method calculates entropy of the dataset.

2.1.4 _calculate_information_gain()

A method to calculate a specific feature's IG (information gain) given the range of data IDs.

2.1.5 _get_max_ig_feature()

With the use of _calculate_infromation_gain(), this method calculates IG for each feature, and returns the ID of the feature with the max IG.

2.1.6 fit() / _fit_helper()

A fit() method calls a recursive helper method: _fit_helper() which uses the feature with max IG as the split, and creates child nodes for each distinct value for that feature. Then recursively call itself on this child node, until no more feature or data sample is left for the downstream.

2.1.7 print_tree()

A printer method to show the tree structure.

2.1.8 predict()

A method to traverse the decision tree given the feature values of the datapoint to predict.

2.1.9 reduced_error_pruning() / _pruning_helper()

A method to prune the decision tree given the set of data. A helper function first traverses the tree to the bottom most parent node, and calculates the accuracies for pre-pruning/post-pruning tree, and if the accuracy is improved, prunes the tree by marking the corresponding node's identifier isPruned.

2.2 CART (CART.py)

2.2.1 CART Regressor Node Class

A class of regression trees, stores feature data, X and target values y, with other info: features_names, node_type, depth, a string rule, and stopping threshold value.

Within initialization:

1. A root node is created with depth 0,

2. Mean value of y (as the predicted value at the root node), and MSE are calculated,
3. pointers to the left and the right trees are initialized.
4. Other supportive values (split_feature_id, split_feature_val, data length n) are initialized.

2.2.2 _calc_mse()

A method to calculate Mean Square Error given a list of y values and y hat value.

2.2.2 _get_midpoints()

This method returns a list of values representing the midpoints of the unique values for the feature pointed in the input argument.

2.2.3 _get_best_split()

A method to get the best split by calculating midpoints for each feature, then splitting the data and calculating MSE after the split. If the MSE is improved, return the index number and value of the best split feature.

2.2.4 fit()

This method recursively operates tree splitting and calls itself on the left and the right node, until no more split is operated by _get_best_split()

2.2.5 print_info() / print_tree()

A printer method to visually display the tree structure

2.2.6 predict()

Given the feature values of one data point, this tree traverses through the regression tree, and returns the prediction.

2.3 Utils

2.3.1 K_fold()

A function for operating 5-fold cross validation, which samples uniformly from the dataset and returns the list of folds with the length of K. For example, if there are 1000 examples, this function generates a list of 5 data lists, each containing 200 examples.

2.3.2 Read_file, Evaluation, Data_handler

Helper functions for data loading, preprocessing and the model evaluation. Implemented in project 1.

3. Results

3.1 Classification with ID3

In the following section, the results of the prediction using the fitted decision tree with and without pruning will be demonstrated. The demonstrations are based on the 3 datasets for classification as below:

- Breast Cancer
- Car Evaluation
- Congressional Vote

3.1.1 Preparing Folds

After loading and preprocessing the data such as handling missing values, 20% of dataset is extracted as a tuning data for pruning, and in order to operate 5 Fold cross validation, from the remaining 80% of data, 5 folds are generated with the below code snippet:

```
folds = k_fold.k_fold(data)

pruning = folds[-1]

pruning_X = [x[:-1] for x in pruning]

pruning_labels = [x[-1] for x in pruning]

validation = folds[0] + folds[1] + folds[2] + folds[3]

validation_folds = k_fold.k_fold(validation)
```

3.1.2 Fit and Predict

With the use of 5 folds and a tuning data set, fittings and predictions are operated. The average accuracies of pruned and unpruned version of the trees for three classification dataset as below:

	Breast Cancer	Car Evaluation	Congressional Vote
Unpruned	93.39%	72.38%	93.96%
Pruned	93.75%	72.52%	93.97%

From the result above, we can see the improvement of the accuracies on the test dataset after pruning with the tuning data. And in general, the classification tree built by ID3 performs well on Breast cancer and congressional vote dataset.

3.2 Regression with CART

For regression, the results of fitting regression trees with and without early stopping strategy will be demonstrated. The experiments are performed using the three regression datasets as following:

- Abalone
- Computer Hardware
- Forest Fires

3.2.1 Preparing Dataset

Similar to the classification tasks with ID3, the dataset used to train a tree model needs to be preprocessed beforehand. For example, handling categorical features. Then 20% of the source data is pulled as a tuning data to tune the hyperparameter: stopping threshold. And the remaining 80% are split into 5 folds to run 5 fold cross validation.

3.2.2 Threshold Tuning

With the use of tuning set, the best threshold with the lowest MSE is searched as below:

```
thresholds = [0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]

best_mse = None

best_threshold = None
```

```
    for t in thresholds:

        tuning_train_X = [x[:-1] for x in tuning_train]

        tuning_train_y = [x[-1] for x in tuning_train]

        tuning_test_X = [x[:-1] for x in tuning_test]

        tuning_test_y = [x[-1] for x in tuning_test]

        model = CARTRegressorNode(tuning_train_X, tuning_train_y, feature_names,
stopping_threshold=t)

        model.fit()

        preds = [model.predict(x) for x in tuning_test_X]

        mse = evaluation_metric('mse', tuning_test_y, preds)

        print(f'Threshold: {t}, MSE: {mse}')

        if not best_mse or mse < best_mse:

            best_mse = mse

            best_threshold = t
```

Example Output for Forest Fires:

…

Threshold: 100, MSE: 338.55210172958374

Threshold: 1000, MSE: 259.52695180966464

Threshold: 10000, MSE: 278.06099033894617

...

Best Threshold for Stopping is: 1000

And the results of the hyperparameter tunings for three datasets as below:

	Abalone	Computer Hardware	Forest Fires
Best Stopping Threshold	10	1000	1000

3.2.3 Fit and Predict

After the best threshold value for early stopping is searched, the average MSEs among 5 fold cross validations of the fitting with/without early-stopping are calculated. The results for three regression datasets as below:

5 CV average:	Abalone	Computer Hardware	Forest Fires
Unstopped Tree	6.72	4518.88	8048.84
Stopped Tree	6.72	4518.88	8048.84

From the experiments on the training dataset, no improvement or change can be observed for MSEs of unstopped/stopped trees. So I examined the performance on tuning dataset, the result is as below:

MSE on Tuning Set	Abalone	Computer Hardware	Forest Fires
Unstopped Tree	8.96	3315.11	416.49
Stopped Tree	8.45	3184.96	259.53

From the results on the tuning set we can see that the MSEs decrease with early stopping.

4. Conclusion

In this paper, I described my approach to implement decision trees with ID3 and CART algorithms from scratch. Also the results of model tuning and the comparisons of the performances that ID3 with/without pruning, and the CART with/without early-stopping show are demonstrated.