Classification And Regression Trees Implementation

Sharath Simha

Abstract

The purpose of this project is to understand, develop and evaluate a category of popular supervised machine learning algorithms referred to as "Classification and Regression Trees (CARTs). They attempt to predict output for new input based on previous modeling from a set of known observations. The difference between Classification and Regression lies with the predicted output being a set of finite values with the former, and infinite numerical values with the latter.

Introduction

When successful, CART algorithms can find application in a variety of real-world situations. It is therefore important to evaluate them, and thoroughly understand their relative strengths and weaknesses. Further, due to their very nature, supervised learning algorithms tend to "overfit" to the training data. Therefore, one of the main motivations of this project is to understand the conditions under which overfitting occurs and develop techniques to minimize them.

In this project, four CART algorithms – Classical Decision Learner (Quinlan, 1985), Random Tree Learner, Bag Learner and an "Insane Learner" are developed and evaluated. Based on conceptual understanding of the learners, the following hypotheses are proposed:

- Classical Decision Learners are expected to deliver higher out-of-sample prediction accuracies while being relatively slower to train.
- Random Decision Learners are expected to deliver better training speed with some compromise in prediction accuracies.
- Bag Learners are expected to be significantly slower to train and query due to multiple learners, however, should result in reduced overfitting.

Methods

The Classic Decision, Random Tree and Bag Learners are each implemented as individual Python classes and provide a consistent interface for creating, training, and querying. Additionally, a fourth "Insane Learner" is also provided to demonstrate the power of combining and forming complex learners. Technical details on importing, constructing, and using the learners are in Appendix A.

Through a series of 3 experiments (described below), the Decision, Random, and Bag Learners were evaluated against certain selected metrics. A container program (see Appendix B) is provided to conveniently run all 3 experiments by supplying the input file as a command line argument. Alternatively, the reader can choose to utilize the learner APIs (Appendix A) to setup other interesting experiments.

- All three experiments conduct multiple (25-100) trials of evaluating the learner(s) in question, and the metrics are then averaged.
- For every trial, the source dataset is randomly split into smaller datasets with 60% being used to train the model, and 40% for testing.
- Experiment #1 evaluates the Decision Learner through 100 trials. In each trial, leaf size is varied from 1 to 50, and the model is first trained and then tested with both in-sample and out-of-sample data. Overfitting is analyzed by plotting average Root Mean Squared Error (RMSE) ('Root-mean-square deviation', Wikipedia) against leaf size for both in and out-of-sample metrics.
- Experiment #2 evaluates the impact of Bagging on Overfitting using a Decision learner. A fixed bag size is selected (details later). 25 trials are run, with leaf size varied from 1 to 50 in each. Areas of overfitting are analyzed as before.
- Experiment #3 performs a quantitative comparison of the Decision and Random Tree Learners using Mean Absolute Error ('Mean absolute error', Wikipedia) and Time-To-Train (details in a later section). The experiment is run over 100 trials with leaf size varied from 1 to 50 in each.

Experiment 1

Does Overfitting occur with respect to leaf size?

Yes

For which values of leaf size does overfitting occur?

Overfitting occurred for leaf sizes **6 and below**, where Out-Of-Sample RMSE reversed direction and started increasing while In-Sample RMSE continued to decline, as indicated below, and graphically depicted in Figure 1.

Leaf Size	In-Sample RMSE	Out-Of-Sample RMSE	Direction of Overfit
1	0.000138	0.007875	↑
2	0.002417	0.007512	
3	0.003295	0.007241	
4	0.003339	0.007230	
5	0.004509	0.006822	
6	0.004621	0.006771	
7	0.004626	0.006769	
8	0.004628	0.006769	
9	0.004658	0.006761	
…	…	…	

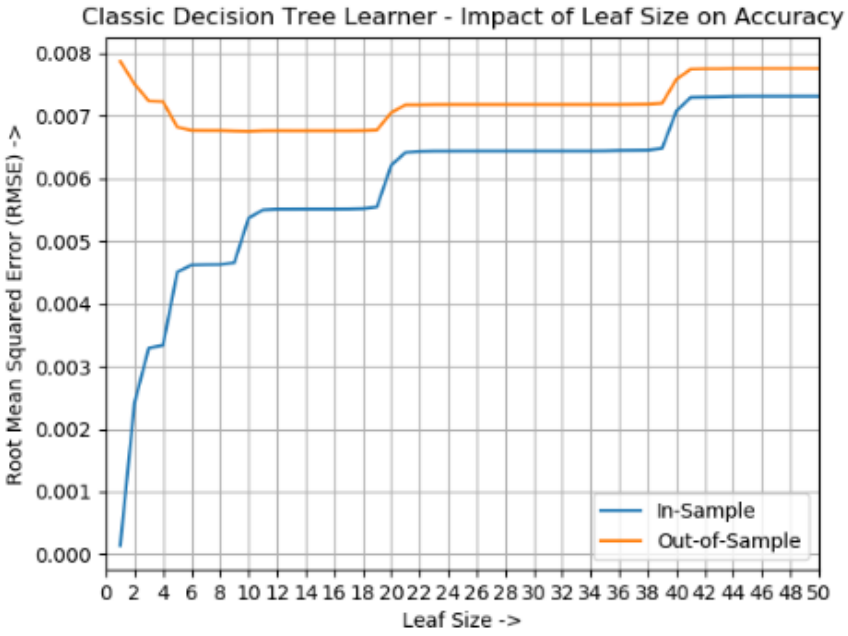

Figure 1— Classic Decision Learner – Impact of Leaf Size on Prediction Accuracy

Experiment #2

Choice of Bag Size (**17**): Before this experiment, the Bag Learner was evaluated offline by varying bag sizes in multiple trials and choosing the bag size that produced the least out-of-sample RMSE (17 bags).

In this experiment, with bag size set to 17, both in-sample and out-of-sample RMSE was tracked with randomized datasets for each bag/learner using a "with replacement" technique – and results averaged across 25 episodes.

Can bagging reduce overfitting with respect to leaf size?

Yes. It was observed in the results that as leaf size reduced to 1, and even as the in-sample RMSE plummeted, the out-of-sample RMSE remained mostly flat, with very minor fluctuations in RMSE around the 5th decimal place. Bagging is an ensemble with multiple learners under the hood. Therefore, even if the individual learners overfit by themselves (which we observed with the Decision Learner at lower leaf sizes), when their average is taken in bagging, the result is a "smoothing" effect and a significant reduction in overfitting.

Leaf Size	In-Sample RMSE	Out-Of-Sample RMSE	Direction of Overfit
1	0.00257	0.00573	Miniscule Decrease
2	0.00280	0.00574	Miniscule Increase
3	0.00295	0.00570	Miniscule Decrease
4	0.00323	0.00573	Miniscule Increase
5	0.00358	0.00572	↑
6	0.00377	0.00568	
7	0.00384	0.00571	
…	...	...	

Can bagging eliminate overfitting with respect to leaf size?

Not entirely. While bagging significantly reduced overfitting as seen above, it did not 100% eliminate it, since some small fluctuations still exist below leaf size 5. The "with replacement" bagging technique ensures at least some repeated "x" points in each bag. Therefore, at lower leaf sizes, each bag will be biased towards the repeated points in its bag. Averaging reduces that bias but may not eliminate it.

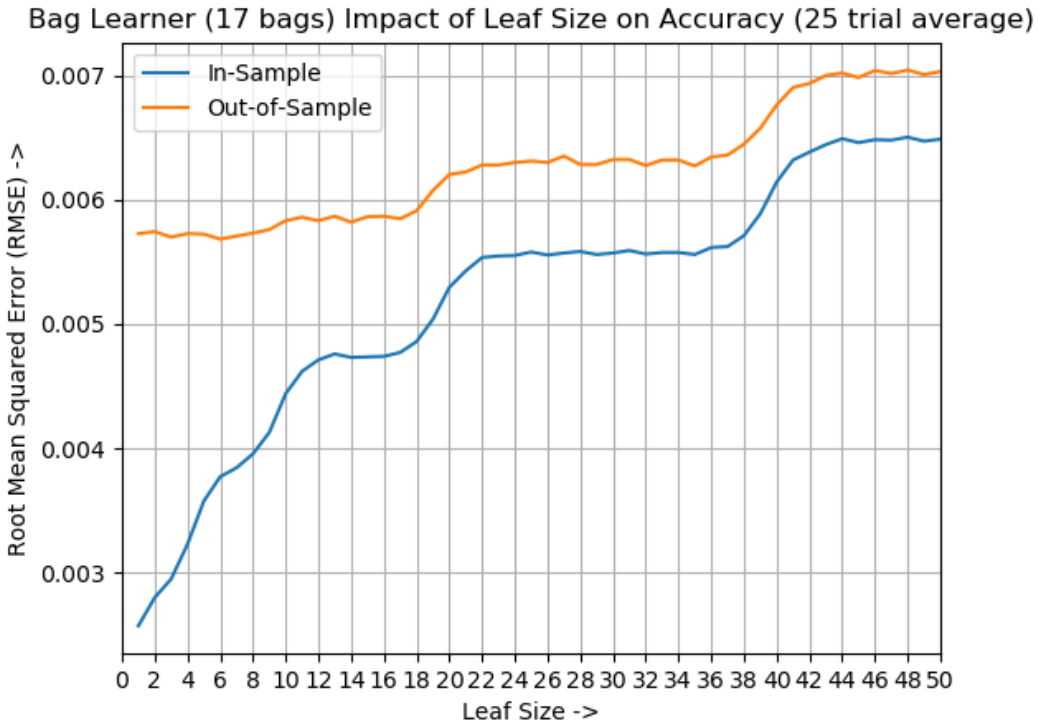

Figure 2— Bag Learner – Impact of Leaf Size on Prediction Accuracy

Experiment #3

The two metrics proposed for comparing Decision and Random Tree learners are:

- Mean Absolute Error (MAE) and
- Time-To-Train

These metrics were chosen because they quantitatively exemplify the main differences between the two learners. Random learners are essentially Decision learners with the only difference being their (random) feature selection, which is expected to result in faster time-to-train but likely with lower prediction accuracy.

In this experiment, the MAE and Time-To-Train metrics were averaged and charted across 100 trials, with leaf size being varied from 1 to 50 in each trial.

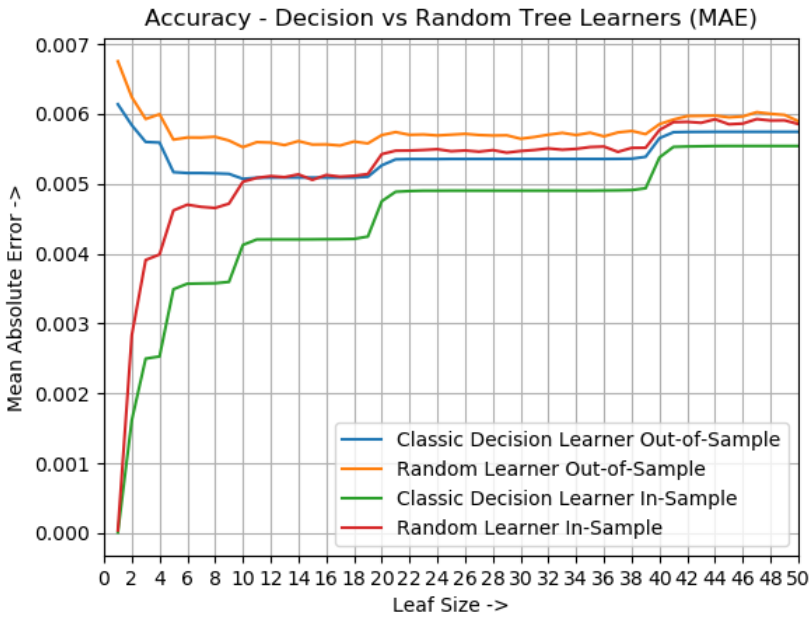

Figure 3— Comparison of Accuracy Between Decision and Random Learners

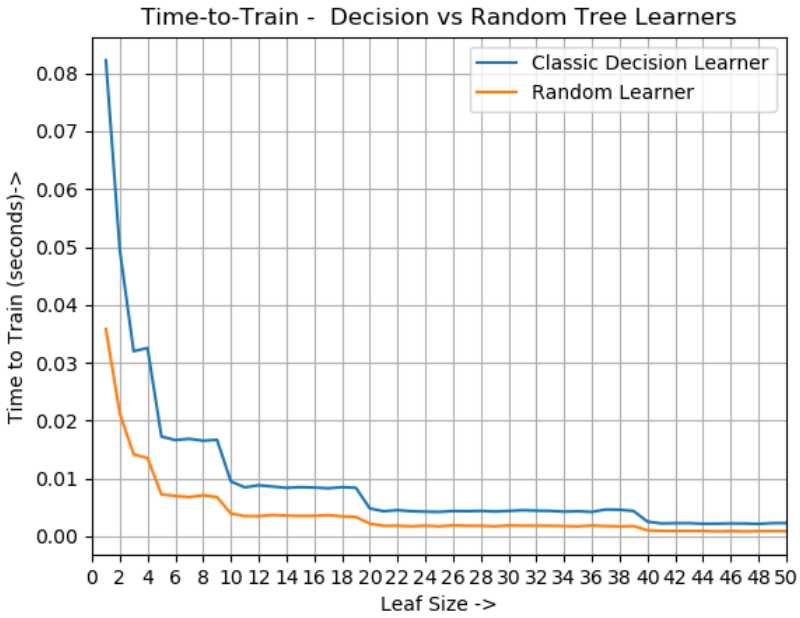

Figure 4— Comparison of Time-To-Train Between Decision and Random Learners

In which ways is one method better than the other?

- Classic Decision Learners are better in terms of prediction accuracy.
- Random Tree Learners are better/faster in terms of time-to-train.

Which learner had better performance and why do you think that was the case?

- Classic Decision Learners have better MAE because the decision tree is built based on the "best" features (highest correlation with observed labels). A more accurate model translates to better prediction accuracy. In contrast, random learners randomly choose features, resulting in lower accuracy.
- Random Learners are better for time-to-train because they don't incur the cost of determining the most correlating feature at each tree level – which is one of the costliest steps in model building, rather simply pick a random feature.

Is one learner likely to always be superior to another (why or why not)?

For both measures, it is *likely* that one is superior to the other. However, supervised learning is not an exact science, rather dependent on quality and diversity of underlying observations. If the feature set doesn't correlate well with labels or are not distinguishing enough, means/medians for large tree chunks are taken in which case accuracy of a random learner may get very close to a decision learner, making the trade-off with training time more attractive. With time-to-train, for small leaf sizes, the random learner is almost always faster since feature selection is likely required. As leaf size increases, there's increased node aggregation (no feature correlation) making performance gains negligible. Time-to-train is an offline performance metric, hence more affordable.

Summary

Key Findings and Insights

Aspect	**Decision Tree**	**Random Tree**	**Bagging**
Overfitting	Significant at low leaf sizes	Significant at low leaf sizes	Miniscule (best)
Prediction Accuracy	Good except in region of overfitting	Generally lowest	Highest
Training Time	High	Lowest	Highest (due to multiple learners)
Query Time	Not Evaluated (Future Work)		

REFERENCES

1. 'Decision tree learning' 2023 Wikipedia. Available at: https://en.wikipedia.org/wiki/Decision_tree_learning. (Accessed: 2-Feb-2023)
2. J.R. QUINLAN. (1985). Induction of Decision Trees. Machine Learning 1.
3. 'Root-mean-square deviation' (2023) Wikipedia. Available at: https://en.wikipedia.org/wiki/Root-mean-square_deviation (Accessed: 11 February 2023)
4. 'Correlation coefficient' (2021) Wikipedia. Available at: https://en.wikipedia.org/wiki/Correlation_coefficient (Accessed: 11 February 2023)
5. 'Mean absolute error' (2023) Wikipedia. Available at: https://en.wikipedia.org/wiki/Mean_absolute_error (Accessed: 11 February 2023)

Appendix A Technical Details of Learners

Import and Construction

Learner	**Class Name and Import**	**Constructor Arguments**	**Description**
Classic Decision	from DTLearner import DTLearner	leaf_size	Maximum nodes for aggregation into leaf
Random Tree	from RTLearner import RTLearner	leaf_size	Maximum nodes for aggregation into leaf
Bag	from BagLearner import BagLearner	learner	Maximum nodes for aggregation into leaf
		kwargs	Any arguments needed for the underlying Learner
		bags	Number of learners
"Insane"	from InsaneLearner import InsaneLearner		

Method Interface

Method	**Description**	**Arguments**	**Description**
author	Returns author		Returns author
add_evidence	Performs model training	data_x, data_y [both NumPy arrays]	Training observations and labels. Expects prior data cleansing
Query	Returns Predictions for provided input	points [Numpy array]	Input observations for which predictions are needed. Returns an Numpy array of predictions for each "x" in points

Note: all methods also support an optional "verbose" argument for printing

Appendix B Container Program for Running all Experiments.

Program Name: *testlearner.py*

Input Data File Location: *<project directory>/Data*

Execution (from project directory)

PYTHONPATH=./..:. python testlearner.py Data/<data file name>

Output: None. Charts generated in project directory

Function Names *(for configuration of trials, leaf size, verbose, show_plot)*

- run_experiment_1
- run_experiment_2
- run_experiment_3