Exploring the Impact of Decision Tree Depth

Alic Szecsei
University of Iowa
alic-szecsei@uiowa.edu

Diego Castaneda
University of Iowa
diego-castaneda@uiowa.edu

Willem DeJong
University of Iowa
willem-dejong@uiowa.edu

ABSTRACT

For classification problems, decision trees are a very easily understandable machine learning tool that does not require much computation to use after construction. However, they are easily susceptible to overfitting. A common means to combat this problem is by pruning the decision tree, according to some metric. While this is commonly chosen to be a χ^2 analysis, a computationally trivial metric is simply the height of the decision tree. This has the benefit of reducing the amount of computation initially performed: when a decision tree reaches a specified height, the construction algorithm constructs a leaf node. This prevents branches of the decision tree being computed which would only be pruned in a secondary phase. The process of model selection is designed to determine the optimal height of the decision tree so as to avoid overfitting; the impact of tree height on this model selection process varies wildly from one dataset to another. We used four datasets from the UCI Machine Learning Repository and trained a modified ID3 decision tree learning algorithm on them to analyze the accuracies of the resulting decision trees. We determined that the datasets for which the decision tree process works best are those which have several unrelated attributes; datasets with interconnected attributes require much larger trees and tend to have higher error rates. These datasets also have very high variance in error rates, depending heavily on which data points were chosen for training.

CCS CONCEPTS

• **Computing methodologies → Machine learning**; *Supervised learning by classification*; *Classification and regression trees*; Cross-validation.

KEYWORDS

decision trees, model selection

1 BACKGROUND AND MOTIVATION

Machine learning algorithms fall into one of two categories: classification and regression. In regression, input data is consumed and transformed by a function which can produce values along a numeric range; in classification, data is categorized into a finite number of possible values, and machine learning algorithms seek only to label data sets. These classification problems can use both numeric and categorical data attributes. For purely categorical data, decision trees are a way to build machine learning algorithms that are understandable by both humans and machines. Rather than a sort of "black box" in which numbers are passed in and answers are returned, there is a logical tree hierarchy that is familiar to anyone who has played the game Twenty Questions.

Decision trees require both a way to construct them, and a way to use them to categorize data. This categorization is a trivial tree traversal, and so many of the innovations regarding decision trees have to do with their construction. Most decision tree construction algorithms use a two-phase approach: first a *growing* phase, followed by a *pruning* phase. In the growing phase, the decision tree is built out, trying to fit the provided training data as closely as possible. To combat over-fitting, the pruning phase determines which branches of the tree are too "noisy" using χ^2 tests and removes them.

Additionally, decision tree models can be constrained by size to combat overfitting. Russell and Norvig [6] showcase an implementation of restricting a decision tree to be beneath a maximum size by generating the tree in breadth-first fashion, and stopping when the maximum number of nodes has been reached. As stated in Garofalakis, Hyun, Rastogi, and Shim [3], there is no point in creating a branch when it is guaranteed to be pruned later.

The amount of pruning which occurs is heavily dependent on the data set chosen. As such, the exact values which optimize our trained decision trees are relatively unimportant. However, our goal was to examine how impactful depth-based pruning was, and how much it assisted with reducing overfitting.

2 METHODS

2.1 Dataset Selection

Our first step was selecting datasets to use as the foundation for our programs. We used four UCI data sets [2]:

1. `MUSHROOMS`, a dataset to classify mushrooms as edible or inedible
2. `BALANCE`, a dataset to classify whether or not a scale with objects of varying weights and distances was balanced
3. `CARS`, a dataset of car evaluations
4. `TICTACTOE`, a dataset of Tic-Tac-Toe board states.

These datasets were selected as they have a variety of both number of attributes and number of examples. `MUSHROOMS` has 8124 examples with 22 attributes, for example, while `BALANCE` only has 625 examples and 5 attributes. In addition, some of our datasets were calculated while others were collected from real-world data. We tried to choose a diverse selection of datasets so that our algorithm testing results would not be overly skewed towards certain types of datasets.

2.2 Decision Tree Generation

The foundation of our decision tree algorithm was the one provided by Russell and Norvig [6] which, in turn, is based on the ID3 algorithm [5].

A two-stage approach was used: first, a decision tree generator was developed, called `dtl`. This program would take in a flag to determine which of our data sets was represented by the input. As the ordering of attributes and classes was inconsistent between datasets, this allowed us to significantly simplify the process of importing data.

Function DecisionTreeLearning(*examples*, *attributes*, *parent_examples*, *depth*)

```
if examples is empty then
    return Plurality-Value(parent_examples)
else if depth = 0 then
    return Plurality-Value(examples)
else if all examples have the same classification c then
    return a leaf node c
else if attributes is empty then
    return Plurality-Value(examples)
end
$A \leftarrow argmax_{a \in attributes}$ Importance(a, examples);
tree ← a new decision tree with root test A;
foreach value $v_k$ of A do
    exs ← {e : e ∈ examples and e.A = $v_k$};
    subtree ←
      DecisionTreeLearning(exs, attributes - A, examples, depth - 1);
    add a branch to tree with label ($A = v_k$) and subtree
      subtree;
end
return tree;
```

2.3 Classification

A second program, `classify`, was used to classify data using a decision tree generated from `dtl`. This program would take in two arguments: The first is a path to a learned decision tree and the second is a path to the corresponding dataset to classify. To communicate a decision tree between `classify` and `dtl`, a JSON representation of the decision tree was used. This data format allowed us to produce human-readable representations of the decision trees, which we could use as a sanity check during development. The classifier is also used heavily by the *k*-fold Cross-Validation used for validating our generated models.

2.4 External Packages

To assist with the command-line interface, we used the `cmdliner` module. This allowed us to define data types and arguments for input, and then it handled the majority of the command-line parsing. It also automatically generated a `--help` option to inform users of what the different options were. This massively simplified developing optional new features, as we could simply add an optional flag and `cmdliner` would handle parsing it.

Additionally, we used `adtgen` to generate a JSON serializer and deserializer. This was done at the very start of the project, allowing us to work with and visualize decision tree data from the very beginning. This also meant that as our decision tree model was updated and changed, the JSON code was automatically shared and kept up-to-date between `dtl` and `classify`.

2.5 *k*-fold Cross-Validation

Finally, we split *k*-fold cross-validation into two implementations. The first, done in Python, was very simple to get working, and so we were able to start collecting data fairly early on in the process. At the same time, we developed a model selection program in OCaml, which used a similar algorithm to the one in the Python script.

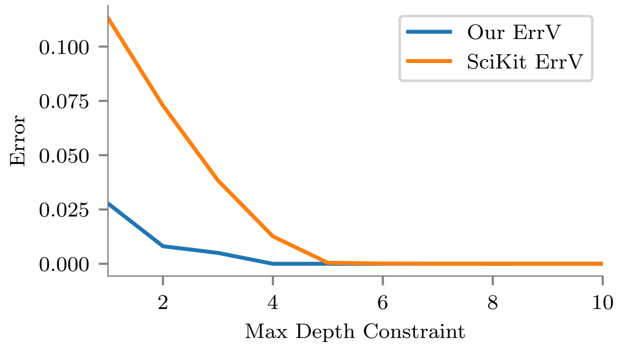

Figure 1: Error for our algorithm versus scikit-learn on the `MUSHROOMS` dataset

Using Python to generate data for *k*-fold cross-validation allowed us to collect data from both our decision tree program as well as the decision tree algorithm provided by scikit-learn [4]. This provided a simple baseline for our experimentation.

We then ran each of our datasets through the DecisionTree-Learning algorithm with maximum depths ranging from 1 to 10 and performed *k*-fold cross-validation to determine error rates, with $k = 4$. As we noticed that subsequent experiments could obtain dramatically varying error rates, we repeated each of these experiments for 100 trials, to determine both average error rates and the variance of these error rates.

3 RESULTS

The results of our experimentation varied quite a bit between datasets. For `MUSHROOMS` in particular, our algorithm was able to achieve significantly greater accuracy than scikit-learn for smaller maximum depth constraints (figure 1). However, for other datasets, such as `TICTACTOE`, our algorithm produced consistently more error-prone results at all depth values (figure 2).

The `CARS` dataset proved the most interesting. While our algorithm had worse accuracy than scikit-learn for small depth values, as the allowed depth increased we began to outperform scikit-learn's algorithm (figure 3).

4 DISCUSSION

Overall, we found that our algorithm performed much worse on datasets with a large amount of interconnected data, such as Tic-Tac-Toe, where the impact of one attribute was heavily dependent on all of the other attributes. However, for more traditional classification problems, such as evaluating cars or mushrooms, our algorithm was very accurate.

From our results, we saw that the only dataset we started overfitting on was `TICTACTOE`. This was likely due to the datasets we chose having significant complexity; although `BALANCE` had the fewest attributes, its classifications relied on combinations of all

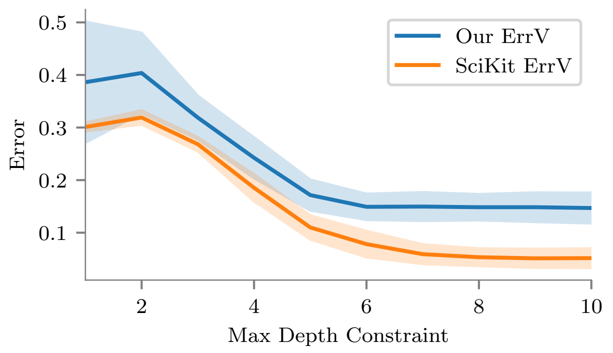

Figure 2: Error (±2×SD) for our algorithm versus scikit-learn on the `TICTACTOE` dataset

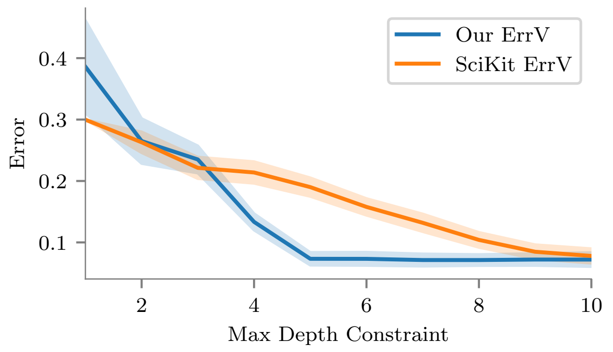

Figure 3: Error (±2×SD) for our algorithm versus scikit-learn on the `CARS` dataset

of them. Our simplest datasets were also calculated classifications, reducing the amount of noise in the data; the more complex models were the ones gathered from real-world data and so were at the most risk of overfitting - but the number of attributes was so high that a maximum depth of 10 didn't appear to reach an optimal model selection value.

We were surprised at the discrepancies between our algorithm and the one provided by scikit-learn, as we expected both decision tree algorithms to produce similar results. While an effort was made to produce similar constraints on both algorithms (scikit-learn was used with the "entropy" heuristic, as well as the same tree depth constraint), there were a few differences that we were unable to resolve. First, scikit-learn uses "an optimised version of the CART [1] algorithm."

Additionally, scikit-learn could not be used with categorical data. To get around this restriction, we used one-hot encoding for the scikit-learn algorithm. This causes a negative impact on the depth: scikit-learn's decision tree was forced to be a binary tree, and so depth constraints acted far more harshly than they did on our algorithm.

The algorithm used by scikit-learn resulted in much closer variance than ours; however, scikit tended to keep a constant variance, while our algorithm produced one which decreased as we increased the maximum depth of the tree. This indicates to us that while scikit-learn's algorithm produces consistently good results for most datasets, our algorithm can produce very good (or very bad) results for those same datasets.

5 ROLE ASSIGNMENT & CONTRIBUTIONS

5.1 Alic

Alic was responsible for implementing the decision tree learning algorithm, the JSON decision tree representation, helped improve the binary executables, and implemented unit testing for several core functions. He implemented the decision tree learning algorithm from Russell and Norvig [6], and added an optional depth constraint. He also was responsible for constructing both the internal decision tree representation and determining the JSON representation for communicating between the `dtl` and `classify` binaries. Additionally, he improved the binary executable command-line experience, enabling order-independent options as well as a `--help` option. Finally, he wrote unit tests for core functions such as `Plurality-Value`, `Entropy`, and `DecisionTreeLearning` itself. He was the project checker.

5.2 Willem

Willem worked primarily on decision tree creation, data file read in, and model selection in ocaml. He created a data reader for CSV files, which was used by all of the OCaml binaries (`dtl`, `classifier`, and the OCaml-based model selection program). He helped program the `dtl` executable, which reads in a data file for the training data, then creates a decision tree. He also wrote up the large amount of hard coded information about the different sets of data, including information about the attributes, their possible values, the possible classification, etc. He also created the OCaml version of model selection using cross validation, which takes in various optional and required command line arguments and prints out the selected tree, error rate in the training set, error rate in the validation set, and the chosen max depth. He also elected to be the project recorder.

5.3 Diego

Diego's primary focus was on writing and testing the classifier specified in the project spec. In developing the `classify` binary executable, Diego had to use Ocaml's I/O functionality as well as methods generated from Yojson to read in the decision tree to be used for classification. Additionally he made use of the reader methods developed by Willem in order to get a list of examples to classify. As he developed the classifier he suggested that indexes referring to attributes be added as a field to the decision tree structure as it was necessary for pullling the correct attribute from an example. Diego also wrote tests using Ocaml's testing framework, OUnit. For testing the classifier, he mocked up sample trees and examples for some of the selected datasets plus some made up scenario sets and verified that the classifier was indeed classifying examples correctly. Lastly, as the designated coordinator he setup daily meetings in the engineering building, and checked to make sure everyone knew their tasks and everyone else's.

ACKNOWLEDGMENTS

This work was supported in part by the University of Iowa.

REFERENCES

[1] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees.* Wadsworth.

[2] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[3] Minos Garofalakis, Dongjoon Hyun, Rajeev Rastogi, and Kyuseok Shim. 2000. Efficient Algorithms for Constructing Decision Trees with Constraints. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '00).* ACM, New York, NY, USA, 335–339. https://doi.org/10.1145/347090.347163

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[5] J. R. Quinlan. 1986. Induction of decision trees. *Machine Learning* 1, 1 (01 Mar 1986), 81–106. https://doi.org/10.1007/BF00116251

[6] Stuart J. Russell and Peter Norvig. 2010. *Artificial intelligence: a modern approach* (3rd ed.). Prentice Hall.

A SUPPLEMENTAL GRAPHS & DATA

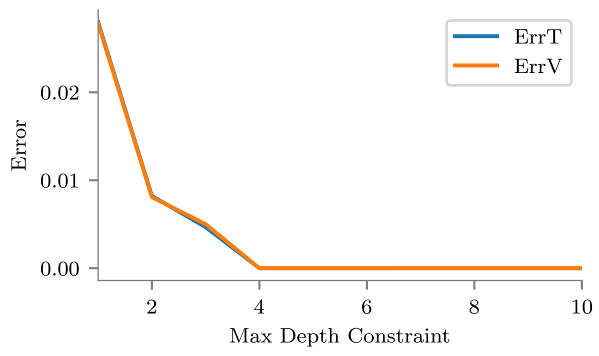

Figure 4: The error on training and validation data for the `MUSHROOMS` data set

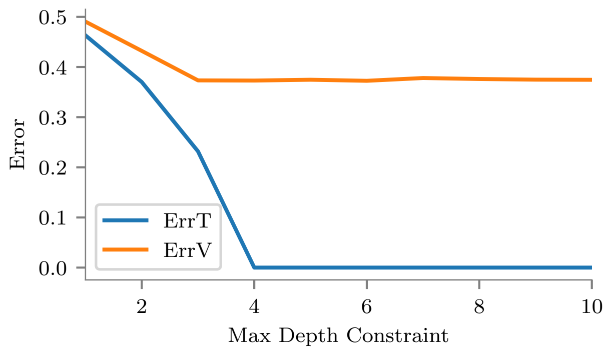

Figure 5: The error on training and validation data for the `BALANCE` data set

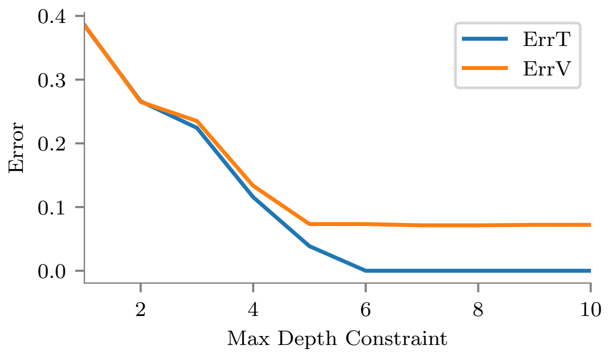

Figure 6: The error on training and validation data for the `CARS` data set

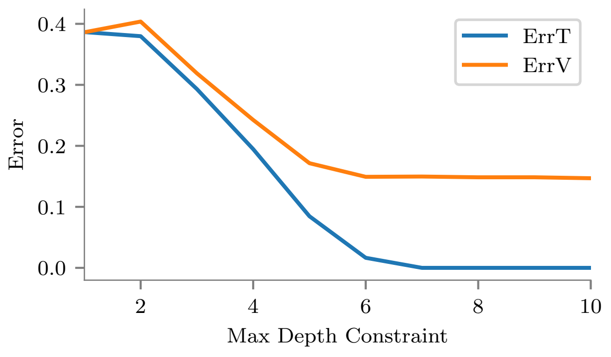

Figure 7: The error on training and validation data for the `TICTACTOE` data set

	BALANCE		CARS		MUSHROOMS		TICTACTOE	
MaxDepth	Avg Error	SD	Avg Error	SD	Avg Error	SD	Avg Error	SD
1	0.404191	0.012534	0.299769	6.409876e-17	0.113245	6.927670e-17	0.301035	0.004068
2	0.359704	0.019186	0.262998	8.538287e-03	0.072871	6.937492e-17	0.319115	0.006669
3	0.299806	0.013906	0.221366	8.831223e-03	0.038405	3.836886e-17	0.268329	0.006638
4	0.253065	0.012958	0.213860	8.886552e-03	0.012629	1.224688e-04	0.185432	0.012746
5	0.227718	0.011899	0.190035	7.656538e-03	0.000446	1.144321e-04	0.110016	0.011334
6	0.235505	0.011984	0.157795	6.878887e-03	0.000137	1.420214e-04	0.078250	0.012445
7	0.234140	0.011871	0.131944	7.268610e-03	0.000027	9.986264e-05	0.059186	0.009237
8	0.227516	0.011412	0.104103	6.179355e-03	0.000025	9.896975e-05	0.053361	0.008296
9	0.224897	0.010037	0.084815	5.794818e-03	0.000022	8.814005e-05	0.051273	0.008884
10	0.222565	0.009456	0.077986	5.976894e-03	0.000016	7.951336e-05	0.051618	0.009168

Table 1: Errors and standard deviations of decision trees for different datasets