University of California

Los Angeles

Semi-Greedy Construction of Oblique-Split Decision Trees

A thesis submitted in partial satisfaction

of the requirements for the degree

Master of Applied Statistics

by

Matthew Rudolph Larriva

2019

Abstract of the Thesis

Semi-Greedy Construction of Oblique-Split Decision Trees

by

Matthew Rudolph Larriva

Master of Applied Statistics

University of California, Los Angeles, 2019

Professor Frederic Paik Schoenberg, Chair

Classification and Regression Trees (CART) are a method of structured prediction widely used in machine learning. Favored for their robustness to non-linear relationships and interpretability (James, Witten, Hastie, & Tibshirani, 2013), they have seen continued application since their broad introduction in 1984 (Breiman, Friedman, Olshen, & Stone, 1984). The shortcomings of a single-tree model, specifically overfitting and lack of competitiveness compared to other machine learning methods, have been overcome through advances in bagging, pruning, and boosting (James et al., 2013). Decision trees are fitted in a top-down greedy fashion because non-greedy fitting is computationally intractable. While this leads to efficient computation, using top-down greedy estimates can produce suboptimal models. To overcome this suboptimality I propose a semi-greedy method of decision tree construction. First, I reformulate the problem based on the work of Norouzi et. al. (2015)(Norouzi, Collins, Johnson, Fleet, & Kohli, 2015), and use an algorithm which optimizes the split functions at all levels of the tree jointly, based on a global objective. This algorithm is parameterized by a single scalar for which I propose a bound. Second, I propose a dimensionality reduction step for cases when the semi-greedy algorithm becomes intractable. In the five datasets tested (Breast Cancer Wisconsin, Banknotes Authentication, Haberman Survival Data, Blood Transfusion Data, Fertility Data), the algorithm produces higher accuracy than two existing greedy tree packages in R (rpart, tree). And, it is computationally tractable for low-width datasets (approximately 8 features or fewer). For higher order problems, I propose

a preprocessing step which reduces the higher-width data to a more tractable subset. I show that a dimensionally reduced space, processed using the semi-greedy algorithm, outperforms the greedy methods (rpart, trees) trained with both full and reduced data sets.

The thesis of Matthew Rudolph Larriva is approved.

Yingnian Wu

Guido Montufar

Frederic Paik Schoenberg, Committee Chair

University of California, Los Angeles

2019

To my parents and theirs:

the giants who let me stand on their shoulders

that I might see far

Table of Contents

List of Figures

Acknowledgments

I wish to acknowledge the staff of the Statistics department at UCLA, especially those involved with the Master of Applied Statistics Program. The paradigm of education in the United States seems to imply that formal education be completed all at once, at the beginning of our lives. The MAS program offered a route to those of us who sought a different path: those of us who wanted to continue our education at night, after work. At UCLA, we found teachers who understood: who brought dinner for class, who showed up every night with coffee to share, who taught 8 hours straight and hosted weekend office hours, all so that we could learn on our schedule.

CHAPTER 1

Background

1.1 Decision Trees for Machine Learning

Decision trees have long been a staple of decision-making. Their appeal is in their simplicity and in their parallel to human cognition. They are easy to construct, intuitive, and lend themselves to elucidating visualizations. Because of this, they have found application in fields as diverse as management (Ritzman, Krajewski, & Klassen, 2004), philosophy (Steele & Stefánsson, 2015), and of course in computer science and statistics. They were among the first methods to be used in the field of machine learning (Quinlan, 1985). This section begins with a review of the first decision trees in computer science, following their development to the early ancestor of today's decision trees. Next, an evaluation of advancements to each component of a decision tree is provided, with non-greedy methods being reserved for the following section on related work. This is followed by a discussion of achievements using multi-tree methods or *forests*. Finally, this section examines the current state of the art.

The concept of inducing a decision tree from examples emerged in 1963, shortly following the academic rise of machine learning as a distinct discipline in the 1950s (Quinlan, 1985). The earliest member of the family of so-called, Top Down Induction of Decision Trees (TDIDT) was, Hunt's Concept Learning System framework (CLS) (Hunt, Marin, & Stone, 1966). CLS focused on tree induction via minimizing two costs: the cost of identifying the true class of an element and the cost of misclassifying an element. Hunt et. al. (1966) recognized the problem of the greedy algorithm and addressed it via a look-ahead function which considered how a split decision would impact splits down the tree.

Hunt's look-ahead function indicates that the search for optimal, non-greedy decision

trees was of interest since inception. This was explored thoroughly in the late 1960s and early 1970s and was proven NP-complete in 1970 and thus intractable on the supposition that $P \neq NP$ (Laurent & Rivest, 1976). Future work moved forward without regard to correctness.

The Iterative Dichotomiser 3 (ID3) (Quinlan, 1979, 1983a) built on the work of CLS but abandoned the look-forward feature (example provided in Figure 1.1). ACLS (Paterson & Niblett, 1982) generalized ID3 and allowed the decision tree to deal with properties (here, integer values) not contained in a training set, and saw application in image recognition. ASSISTANT (Kononenko, Bratko, & Roskar, 1986) was also a generalization of ID3, but unlike ACLS, could process continuous and integer values.

ID3 inducted decision trees as follows: take a subset of training data and construct a decision tree that accurately maps the subset to its classes. Expand the subset of training data and test if the decision tree accurately maps the new data to its correct class. If so, stop, otherwise amend the original tree to correctly map the new data, then repeat.

In 1984, Breiman et. al. introduced a Classification and Regression Tree (CART) method which was fundamentally different from its predecessors. Most subsequent methods inherit from this algorithm, and Breiman et. al. (1984) are credited with the introduction of classification and regression trees into modern computer science and statistics (Breiman et al., 1984).

1.2 Classification and Regression Trees

Classification and Regression Trees (CART) is an umbrella term referring to algorithms that engage tree-based methods to create predictive models. The original implementation was formally introduced in 1984 by Breiman et. al. (Breiman et al., 1984). Therein, the authors provided an efficient, greedy algorithm that utilized binary decision trees.

They applied their binary tree structured classifiers to the problem of identifying six ship classes through their radar range profiles. Radar range profiles could range in dimensionality

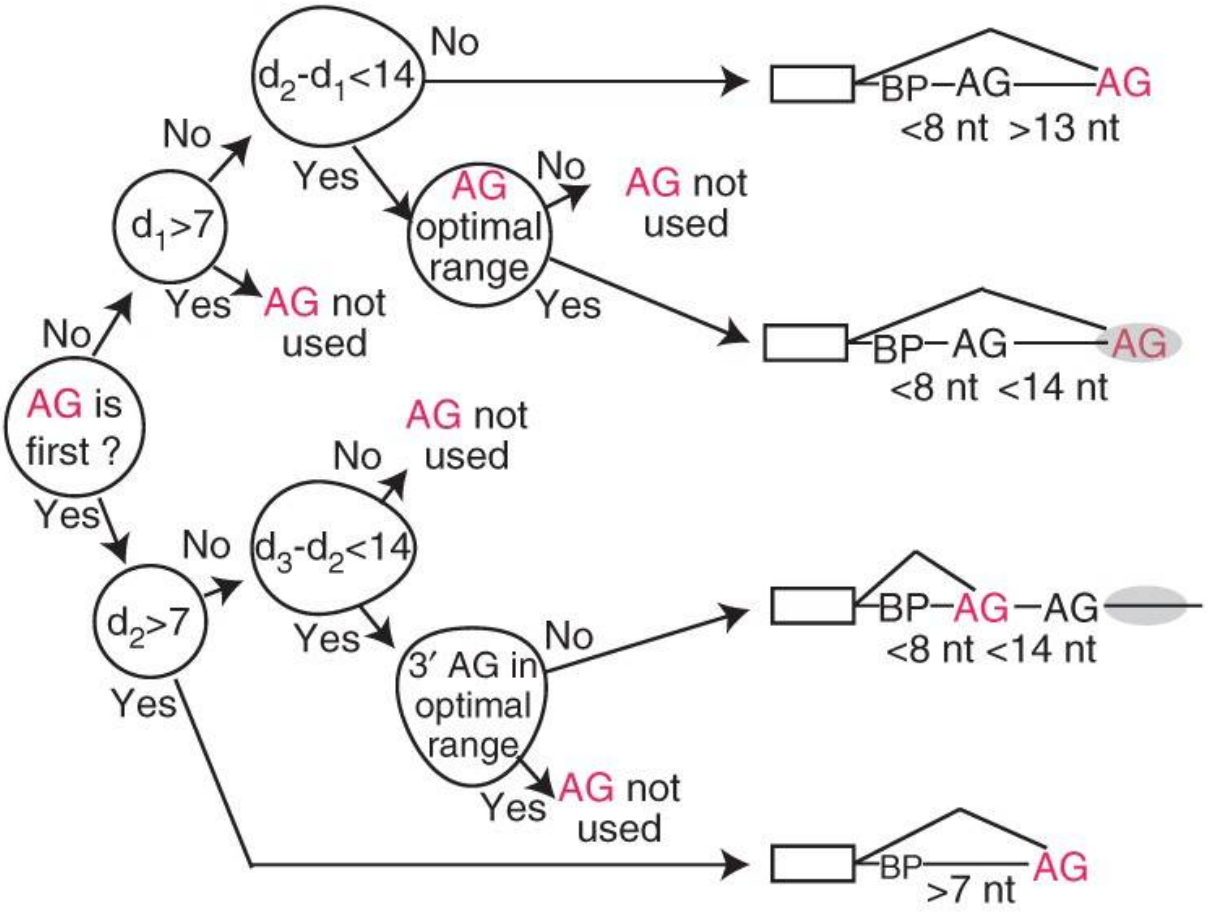


Figure 1.1: An ID3-generated decision tree with a 95% correct prediction rate in nucleotide–to-pre-mRNA pairing (Taggart, DeSimone, Shih, Filloux, & Fairbrother, 2012).

from 1 to 15, and the lack of an existing method suited to handle variable dimensionality led to the genesis of the classification tree.

Similar to prior decision trees, CART is designed to split an integrated set of data into segregated groups using partitions. But disparate from ID3, CART trains itself with all available data instead of just a window of data, seeks to optimize impurity decrease versus information gain, and applies rather than infers minimum splitting criterion.

A Classification and Regression Tree works as follows: a data set held in a matrix X with n features is first split into two groups on the basis of feature x_1. Members (rows) of X with $x_1 > n_1$ are grouped together, as are members of X with $x_1 < n_1$. These two groups are then split again, on the basis of another feature. This process continues until either a minimum splitting criterion is reached or there is only one member in each terminal node (Figure 1.2 contains an example of a Regression Tree).

As such, a decision tree has three parameters: the splits at each node, the decision to declare a node terminal or not, and the classification of the terminal node.

To measure the goodness of split of each grouping, an impurity function is needed. Originally, it was chosen as

$$i(t) = -\sum_{j \in J} p(j|t) \log p(j|t)$$

Where the impurity of the split t is the negative sum of all the number of classes of the proportion of the cases belonging to class j, in all possible classes of J, times the log proportion of classes of j in t.

At each node, every available variable is searched, and every possible split for every variable is considered. This process is repeated recursively for each subsequent node, and the split with the highest impurity decrease each single node is selected. Because subsequent nodes are not considered in the splitting decision at the current node, it is a so-called greedy algorithm. This is referred to as a non-oblique tree, as splits are made considering only one variable at a time, and are therefore perpendicular, or non-oblique, to the feature axes.

1.3 Improvements to CART

For an in-depth discussion of these improvements as they stood in 1998, see Murthys very comprehensive survey on decision trees, from where the structure of the following section is inspired and updated (Murthy, 1998).

1.3.1 Improvements to Splitting

At every non-terminal node a decision tree must select both a variable on which to split, and a value of that variable on which to segregate data. This is accomplished by evaluating a splitting function, usually for all variables across all values of the variable. The terms, feature evaluation rules, splitting functions, and splitting rules are used interchangeably, as splitting functions determine the features used, and are therefore feature-evaluators as much as split-evaluators.

Ben-Bassat segments splitting functions into rules derived from information theory, from distance measures, and from dependence measures (Ben-Bassat, 1982).

Information theory-based rules seek to optimize information gain either locally or globally, based on Shannon's entropy (Murthy, 1998).

Distance-based rules seek to minimize the difference between class probability distributions. Dependence-based measures seek to maximize statistical dependence between two random variables.

There are many more methods of splitting that do not fit into those categories including functions based on testing-cost and testing-probability, functions based on misclassification counts, and functions based on inferred probability distributions. Despite the large and varied body of splitting functions, there is little research to suggest that any one function is consistently superior to its peers (Bellacicco, 1984).

The strength of a splitting function generally depends on the characteristics of the underlying data, as different functions are capable of addressing biases found in multi-class, imbalanced, or heteroscedastic data.

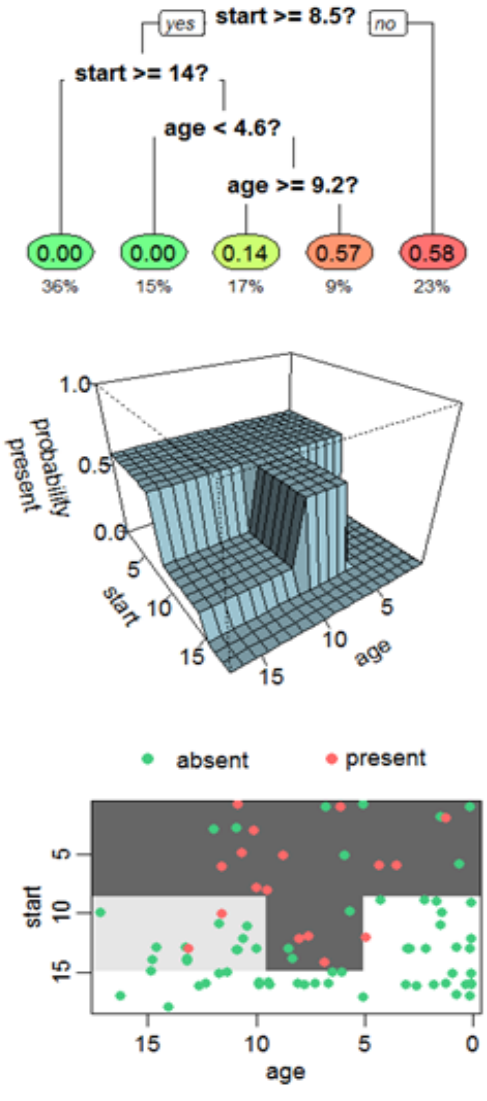


Figure 1.2: "An example tree which estimates the probability of kyphosis after surgery, given the age of the patient and the vertebra at which surgery was started. The same tree is shown in three different ways. Top: the colored leaves show the probability of kyphosis after surgery, and percentage of patients in the leaf. Middle: the tree as a perspective plot. Bottom: aerial view of the middle plot. The probability of kyphosis after surgery is higher in the darker areas. (Note: The treatment of kyphosis has advanced considerably since this rather small set of data was collected.")(Commons, 2018).

1.3.2 Improvements to Sizing

More important than splitting rules are stopping rules (Breiman et al., 1984), as running the recursion until natural termination leads to overfitting. Different stopping functions related to minimum impurity and minimum object size are popular hyperparameters despite being not particularly robust (Friedman, 1977). The most popular method of tree-sizing is called *pruning* and comes from Brieman (1984). Pruning builds trees to completion, then seeks to remove certain trees that are deemed to contribute to overfitting. The method of gauging likelihood of overfitting has many versions which are used with varying degrees of accuracy in different data sets.

1.3.3 Improvements via Multivariate Splits (Oblique Splits)

The discussion of decision trees has, up to this point, centered around the evaluation of a single variable at every node. As such, these decision trees partition their space with axis-parallel splits. But decision trees can also be evaluated obliquely, using hyperplanes to split the space, based on linear combinations of variables. Such a split can be effective when class segregation does not occur along the feature axes. The first oblique decision tree algorithm was published, by Brieman (1984), concurrently with CART and was known as CART-LC (linear combination).

Oblique splits are more computationally expensive and have proven intractable for some splitting functions. But despite the computational resource requirements, oblique decision trees often produce smaller trees with higher accuracy.

As outlined in Wickramarachchi et. al. (2015) (Wickramarachchi, Robertson, Reale, Price, & Brown, 2016), oblique decision trees have been of three forms: optimization-based decision trees seek to climb deterministic hills to find maxima as in CART-LC, Simulated Annealing Decision Trees (Heath, Kasif, & Salzberg, 1993), and OI1 (Murthy, Kasif, & Salzberg, 1994); statistical-based oblique trees use ANOVA and Linear Discriminant Analysis to find the best oblique split, as in Quick Unbiased Efficient Statistical Trees (QUEST) (Loh & Shih, 1997); heuristic oblique decision trees, like CARTopt (Robertson, Price, & Reale,

2013), start with an oblique two-class tree then from an axis-parallel tree after reflecting the training data with Householder matrices, then finally reflect the training data back onto its original space, resulting in an oblique tree.

HHCART improves upon CARTopt by allowing handling of qualitative and quantitative factors and by using the eigenvectors of the estimated covariance matrices combined with Householder matrices (Wickramarachchi et al., 2016).

1.3.4 Improvements via non-greedy splitting

Non-greedy splitting takes on two distinct forms historically: the two-stage approach and the look-ahead approach.

The look-ahead approach uses a splitting function that considers how the split at the present level will affect splits at future nodes. This can be done partially (looking ahead at least one node), or exhaustively (looking ahead at all future nodes). The IDX algorithm (Norton, 1989) used this look-ahead feature, but this approach was proven suboptimal with one-step look ahead , and intractable due to its $O(n!)$ formulation with full search.

The two-stage approach uses a greedy method to partition data, and then attempts to create an optimal decision tree using non-greedy methods. The challenge is that these methods begin with greedy trees or methods and therefore must introduce some method to escape the potential local minima. Simulated annealing has been attempted with some success (Heath et al., 1993) .

Recent improvements to non-greedy splitting have explored contract anytime algorithms that attempt an exhaustive search but can be interrupted or scheduled to run with contract time known *a priori*. Such algorithms lie in the space between greedy and exponential run times and have shown good anytime behavior and significantly better long-run-time behavior (Esmeir & Markovitch, 2006) .

1.4 Current State of the Art

Decision trees continue to be an integral part of machine learning, valued for their interpretability and predictive power. A 2017 Kaggle survey found decision trees to be the second-most used algorithm among respondents (*The State of ML and Data Science 2017*, 2017). KDNuggets found decision trees to be the fourth most-used algorithm. A number of other surveys place random forests and decision trees in the top 10 most-used algorithms (*Top 10 Machine Learning Algorithms*, n.d.). And a decision tree algorithm had two of ten spots Top 10 Algorithms in Data Mining (2009)(Wu & Kumar, 2009).

This section will discuss the current state of the art, limiting the discussion to decision trees in their traditional CART and C4.5 forms. The Related Work section will discuss the relatively new methods of combining decision trees and neural networks.

CART, as presented above, has numerous benefits and some pronounced drawbacks. Among the drawbacks is instability or non-robustness (James et al., 2013). Perturbing the learning set can often cause large changes in the predictor.

Decision trees often achieve their best accuracy when considered in ensemble fashion. While the discussion of decision trees heretofore has mostly referenced single-tree methods, much of the current research focuses on multi-tree methods.

Numerous trees are grown producing a forest which is then recombined to form a predictor.

Forests are produced commonly with one of the following methods: Random Forest, Boosting, or Bagging.

The Random Forest algorithm (Breiman, 2001) has been extremely successful in both classification and regression (Biau & Scornet, 2016). Random Forests seek to overcome the greedy suboptimality by randomly selecting features and splitting them at their midpoint.

Training many trees and then aggregating them overcomes some of the shortcomings of single-tree, recursive, greedy searching. Recent advances have improved upon the formerly accepted rates of convergence (Klusowski, 2018). Random Forests are efficient when pre-

sented with deep data sets (Gieseke & Igel, 2018), wide data sets (Biau & Scornet, 2016), and noisy data sets (Reis, Baron, & Shahaf, 2018).

Boosting is a meta-algorithm that converts weak learners to strong learners. While decision trees are not weak learners, the concepts at use in boosting can be applied to decision trees to further improve their performance. Boosting decision trees involves iteratively training many trees and linearly combining the trees with a weighting that seeks to minimize the misclassification.

Gradient boosted decision trees take the concepts of boosted decision trees but uses a differentiable loss function to combine the iteratively generated trees. This method has established dominant performance across tabular and sparse data (Feng, Yu, & Zhou, 2018).

Breiman (1996) improved on the work of Breiman, et. al. (1984) by introducing the concept of **B**ootstrap **Agg**regat**ing**, which he recoined, "bagging" (Breiman, 1996).

Bagging works as follows: bootstrap sample the training data and generate a classification or regression tree on each sample; use the mode prediction as the predicted output.

Breiman showed large gains in accuracy for non-robust predictors, and small reductions in accuracy for already robust predictors. Variations on Bagging algorithms have been used successfully to estimate a conditional class probability function (Spanakis, Weiss, & Roefs, 2016) and to enhance regression trees (Khiari, Moreira-Matias, Shaker, Ženko, & Džeroski, 2018).

1.5 Related Work

For all the advances in decision trees discussed up to this section, the models all still follow a recursive paradigm. This is inherently suboptimal as it requires some form of greedy optimization as splits are addressed one at a time. The current research on the topic of non-recursive and non-greedy decision trees is the subject of this section.

Prior research focused on brute-force via partial or exhaustive look-ahead functions with the conclusion that gains were minimal or sometimes negative (Murthy, 1998).

Current research on non-greedy decision trees focuses on finding a differentiable loss function on which gradient descent can be applied. To yield a differentiable loss function, the traditional decision tree problem must be reformulated. In current literature, the reformulation is being addressed through a Neural Net paradigm and an optimization (two-stage) paradigm.

Artificial Neural Nets (ANN) and Deep Neural Nets (DNN) have been generalized to serve the decision tree problem, leading to Deep Neural Decision Trees. Such trees have proven effective in a variety of contexts including tabular data (Yang, Morillo, & Hospedales, 2018), image recognition (Tanno, Arulkumaran, Alexander, Criminisi, & Nori, 2018), unsupervised, and semi-supervised settings (Balestriero, 2017). These Neural Decision trees use stochastic gradient descent or Bayesian model averaging to produce non-greedy splits (Siu, 2018).

The two-stage method of greedy trees was discussed briefly in the Background Section. Work persists in this field with the focus on solving an initial greedy tree and then optimizing it into a non-greedy tree (Norouzi et al., 2015) (Bennett, 1994).

Particularly promising work was done by Norouzi et.al. in 2015, which forms the basis of the approach that this paper seeks to improve upon.

Norouzi et.al. formulate a latent variable model of decision trees and propose a surrogate objective function. These yield a computationally friendly problem and allow regularization of the joint optimization of tree parameters.

The authors go on to implement stochastic gradient descent based on their surrogate objective function and achieve significant performance versus other single-tree methods. The their model requires setting, in addition to the traditional learning-rate η and batch-sizes, a regularization constant denoted by the greek nu (ν). Both eta and nu are set by hand.

CHAPTER 2

Proposed Method

This problem formulation takes the strength of the tree structure and adds to it the power of oblique, multi-variable splits, and a non-greedy initialization and optimization.

While traditional tree-based algorithms direct traversal through single-variable, single-node splitting functions, the goal of this formulation is to produce a map in the form of a vector composed of -1 and 1.

Each element of the map can be read as a direction to move at each node of a decision tree. The leaf layer of the decision tree yields a probability of membership to a base class.

The work herein uses the strength of the problem reformulation provided by Norouzi et.al. (2015) and attempts to further their work in the following areas:

- by suggesting a bounded search-space for the scalar ν of the weight matrix
- by using a greedy, gradient-descent, oblique splitting method to set the initial tree
- by reformulating the surrogate loss functions (the objective functions)
- by using single-node adjustment instead of full-tree adjustments
- by using dimensionality reduction instead of an upper bound on empirical loss to produce a tractable problem for higher dimensional data.

A binary classification application of this method begins with an input matrix X of c features by r records. When multiplied by a weight matrix W, then scaled, a vector V of length c is produced. The signs of the elements of V indicate directions to transverse a decision tree where $+1$ indicates a right move, and -1 indicates a left move. A node may

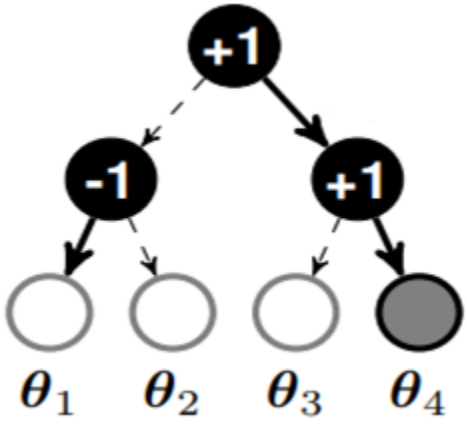


Figure 2.1: An example of a tree for a two-feature data set. Norouzi et. al. (2015)

not be along a path, in which case its value $\{-1, +1\}$ is inconsequential. Once traversed, a tree has at its leaf a matrix of probabilities P which denotes each leaf's probability of belonging to the each of the c classes. The formulation requires a symmetrical tree with a right and left node at each level, though there need not be a member of each leaf or node.

For a binary classification problem with two features, a matrix XW produces V. $V \leftarrow sign(V)$ updates V to a vector of, say, $[+1, -1, +1]$, where the first $+1$ indicates how to split upon the first dimension, and the next two bits $[+1, -1]$ deal with how to split upon the second dimension. A decoder function f walks the path indicated by V and returns the one-hot leaf vector $f(V) = [0, 0, 0, 1]$ which indicates that the sample record belongs to the fourth leaf of the tree. To establish the class probability, we multiply the $f(v)$ by the P to produce the binary vector Θ, which contains the probability of belonging to each class.

The weight matrix and the probability matrix are learned via gradient descent.

The challenge of this model is that there are two parameters to solve for: the optimal weight matrix that will direct elements to a terminal node and the probabilities of class membership once at the terminal node.

The optimal path down the tree is the one that leads to the highest probability of being in the true base class. The least-optimal path is the one which leads to the node with where the highest probability is not being in the true base class.

One approach to solving this problem would be to simply push the weight matrix toward the optimal path. Not only is this a greedy approach, as it does not consider all possible cases of weights but simply tries to move the weight matrix toward the highest probability path, but it is exhaustive, requiring iteration through all possible paths of a tree, which grows exponentially relative to input data features.

A less greedy approach would be to pull the weight matrix away from the least-optimal path. Although there is no explicit benefit to this (because the elements not on the optimal path will not be perturbed) this action has an impact of making the trees generalize better, reducing overfitting, via a non-greedy approach. But this approach also has the exhaustive-constraint, requiring iteration through all possible paths.

To make this problem tractable, I propose seeking the best and worst path down a tree by evaluating the set of trees generated by only manipulating one node within the tree. Thus for each element of training data of dimensions r,c, the search space can be represented by the identity matrix I with count of rows given by $\sum_{k=0}^{c} 2^k$

Evaluating each row as a vector path V, we perturb only one node at a time. Yet for shallow trees–trees with relatively few nodes, or data sets that have a relatively small number of dimensions–this is sufficient to solve for surprisingly strong accuracy. For deeper trees, where performance decreases, we can reformulate the problem with simple dimensionality reduction and regain a high level of accuracy.

The strength in this formulation comes from the following:

- Full-tree, oblique optimization. Most decision trees evaluate a single node at a time. Some do this while only allowing axis-parallel splits to be made. The problem formulation here allows every node of a tree to be considered at once, and allows oblique splits. This adds a high level of flexibility, as splitting decisions can be made with regard to multiple variables and multiple nodes.

- Prevention of overfitting. Overfitting is a common problem in most machine learning algorithms. It can become especially pronounced in decision tree algorithms where there is no universally dominant stopping criteria. Without providing a stopping cri-

terion, the process can recurse until there is only one element in each tree. Usually minimum-impurity splits or minimum-sample sizes are used as stopping criterion but are somewhat arbitrary. The formulation herein prevents overfitting by conducting its optimization subject to changing only one node on a tree at a time, leaving others undisturbed. Because of the bitwise formulation of a path down a tree, this formulation also has the ability to move only nodes relevant to the record at hand, as opposed to all nodes for all records.

- Semi-greedy search. Most decision tree algorithms are greedy, prioritizing the highest value split for each node without regard to down-tree implications. This reformulation forces consideration of down-tree effects of a shift. It also finds a middle ground between the underfitting of greedy methods and the intractability of non-greedy methods.

2.0.1 Specific Description of Proposed Method

The specific process follows:

Let X be the matrix containing the training data of m records and n features. Let there be a tree with p nodes such that p is related to n by $p = \sum_{k=0}^{n} 2^k$.

The weight matrix W is initialized with rows $= m$, and columns $= p$, the total number of tree nodes. The values of W are the columnwise standard deviations of the training data X.

Given training data $X_{m,n}$

$$W_{o,p} = \begin{pmatrix} \sigma_{Xn=1} & \sigma_{Xn=1} & \cdots & \sigma_{Xn=1} \\ \sigma_{Xn+1} & \sigma_{Xn+1} & \cdots & \sigma_{Xn+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{Xn} & \sigma_{Xn} & \cdots & \sigma_{Xn} \end{pmatrix}$$

$$\text{Where } o = n \text{ and } p = \sum_{k=0}^{n} 2^k$$

R Code in Appendix "Weight Matrix Code"

Probability matrix Θ is initialized randomly with dimensions rows = length of leaf layer and columns = 2 $[p(\text{in class}), p'(\text{in class})]$. A softmax is then applied such that row-wise sums are 1.

$$\Theta_{o,p} = \begin{pmatrix} \sim U(0,1) & \sim U(0,1) \\ \vdots & \vdots \\ \sim U(0,1) & \sim U(0,1) \end{pmatrix}$$

$$\text{Where } o = 1 + \sum_{k=0}^{n} 2^k \text{ and } p = 2$$

$$\Theta_{o,p} \leftarrow softmax(\Theta_{o,p})$$

R Code in Appendix "Theta Matrix Code"

An initial greedy tree is then found using the following:

1. For each row x in X, find the current path h taken down the decision tree
    - The path is $sign(W'(x-1))$
    - 1 is subtracted as an offset.
    - A path will be returned in the form of $[m_0, m_1 ... m_n]$ where n is the number of nodes in a decision tree, whose depth is equal to the number of features of the input data X.
2. Find a greedy-optimal path down the tree.
    - Let each row of the identity matrix I_n denote a path down the tree. And let h denote the starting path of the record.
    - Evaluate each path.
    - Return the path g that minimizes the misclassification according to a negative log loss function

3. Update theta by subtracting the learning rate η times the partial derivative of the loss function evaluated at g with respect to the Θ matrix

$$\Theta \leftarrow \eta \frac{\partial}{\partial \Theta} l(\Theta^T f(g), y)|_{\Theta=\Theta^(t)}$$

Where the function f takes the path given by g and returns a one-hot vector corresponding leaf of the tree, and therefore also the corresponding row of Θ.

And where the loss function l is

$$l(x) = -((y)log(p(y = 1|\Theta) + (1 - y)log(p'(y = 1|\Theta))$$

4. Update the weight matrix W by updating the feature weights which are on the optimal path but not the current path, that is elements where $g - h \neq 0$

$$W \leftarrow W + \eta((g - h)x')$$

5. To prevent gradient vanishing or explosion, scale the weight matrix row-wise, where row values become

$$min(1, \frac{\nu^{\frac{1}{2}}}{||W_{row,.}||})$$

Where $\nu = \bar{W}$ and W is the weight matrix

R code provided in Appendix "Greedy Code"

The non-greedy tree is constructed with the above tree as its input, and follows these steps:

1. Build a vector of scaling parameters for the weight matrix. The first is the mean of the original weight matrix. The second is the mean of the greedy-produced weight matrix.

$$N = [\bar{W}_{initial}, \bar{W}_{greedy}]$$

2. For each element of N, for each record in a random sample of records, perform the following:

(a) Find a non-greedy optimal path down the tree.

- A non-greedy optimal path is found by
  i. Finding the current path h
  ii. Evaluating each path in the identity matrix to establish the objective path o and the worst path v. The worst path v is the one which leads to the highest probability of not belonging to the true base class.
- Propose a new path g, found by subtracting the bits on the worst path that are also on the current tree, and adding the bits on the best tree that are not currently in the path

$$g = (o - h + v)$$

(b) using the new non-greedy optimal path g, update and scale the weight matrix and the theta matrix in the same method outlined in the Greedy section (repeating steps 3,4, and 5)

R code provided in Appendix "Non-Greedy Code"

In summary, initial parameters are set randomly for weights of each node on the tree. A greedy tree is found as a starting point, using gradient descent.

The greedy tree is then traversed and updated using a non-greedy method and stochastic gradient descent.

A normalization parameter for the weight matrix keeps the matrix of weights from exploding or vanishing. This parameter ν has a strong impact on accuracy. By limiting the search to two values (the average of the original weight matrix and the value of the revised weight matrix) the weight matrix maintains a size that is scaled down by itself. This serves as a regularization strategy. It also produces strong accuracy without cross-validation and hyperparameter testing because it serves as a regularization parameter.

The time to compute this process expands exponentially with the number of features. The algorithm is governed by the need to search all rows of an identity matrix whose rows are related to its features by $\sum_{k=0}^{c} 2^k$. Thus the algorithm is $O(2^n)$ exponential and is bounded

by some of the constraints of all non-greedy problems.

This can be curtailed by introducing dimensionality reduction. There is a contract/anytime element to the algorithm in this sense. Given time to compute, the original version as outlined above can be used. When limited in time or computing power, dimensionality reduction can be implemented first. The results for both are strong, and will be discussed further in the results section.

There are a large number of dimensionality reduction techniques in practice. I chose a random forest based method of evaluation, which reveals the features that reduce impurity the greatest. When datasets become too large to efficiently process, I reduce the number of features to a more shallow depth, selecting the highest impurity-decreasing features in order.

CHAPTER 3

Results

3.1 Datasets

All tree-based algorithms can improve their performance via boosting, bagging, or creating forests. Results here are shown for a simple single-tree implementation of the algorithm described above.

This algorithm (hereafter: *ng_tree* for "non-greedy tree") was tested on five continuous, real-valued training sets from the UCI Machine Learning repository:

1. Breast Cancer (Wisconsin) Original (*Breast Cancer Wisconsin (Diagnostic) Data Set*, 1995)
2. Banknotes (*Banknote Authentication Data Set*, 2013)
3. Blood Transfusion (*Blood Transfusion Service Center Data Set*, 2008)
4. Haberman's Survival (*Haberman's Survival Data Set*, 1999)
5. Fertility (*Fertility Data Set*, 2013)

Data sets were chosen for their diversity of underlying subject matter (cellular biology, image, behavior, health-science) and for their variety in data structures (integer, continuous, normalized, non-normalized, categorical, time).

Each test was conducted over 20 rounds, using an 80% 20% train/test split, with training and testing data resampled each round. There was no validation, hyperparameter tuning, or other human-in-the loop elements. The algorithm is fully independent.

The performance of the algorithm was compared to two popular R packages: *tree* (Ripley, 2018) and *rpart* (T. Therneau, Atkinson, & Ripley, 2018).

The *rpart* package is Recursive partitioning for classification, regression and survival trees. An implementation of most of the functionality of the 1984 book by Breiman, Friedman, Olshen and Stone (T. M. Therneau, Atkinson, et al., 1997). The *rpart* package is used in a canonical text to describe functionality of recursive partitioning tree-based algorithms. It is in the 99.99th percentile of downloads of R packages.

Tree is a slightly newer package for classification and regression trees that is gaining popularity.

In each case, the *ng_tree* produced the highest accuracy, which was always found at shallow trees. At depth, *ng_tree* underperformed *rpart* and *tree*.

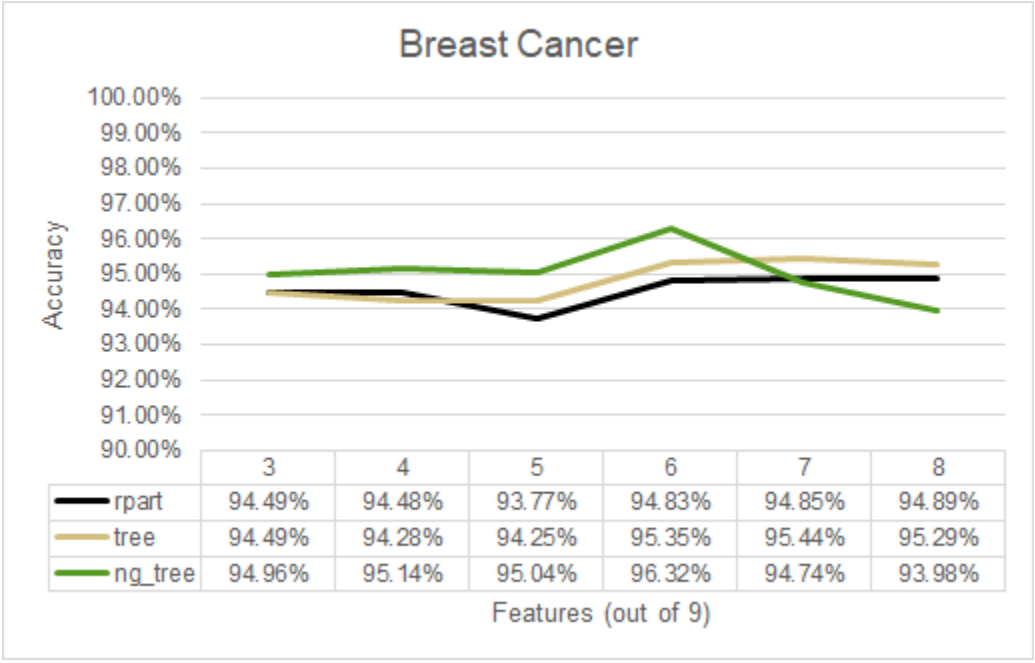
This algorithm, without preprocessing, may not be well suited to data sets with a high number of features. In such cases, preprocessing the data with improves performance once more.

Because the algorithm works by only perturbing one node on a tree at a time, it is understandable that it would underperform when given higher feature data. Increased features require increasingly complex trees to purify the classes. But it is noteworthy that even high feature-set data can be efficiently reduced and split using the algorithm, it must simply undergo dimensionality reduction first.

3.2 Breast Cancer Wisconsin

The Wisconsin Breast Cancer data is a collection of 699 records of 10 attributes describing characteristics of biopsies. The data has two classes: benign and malignant. These classes are the target values of the predictive algorithms. The cancer data, in its original from, has been processed such that each variable is bounded by [1,10].

The different algorithms were shown 3-8 features out of the full 9-feature set. The *ng_tree* produced the highest accuracy, when shown 6 features (Figure 3.1). Accuracy decreased sharply at greater depth. Calculation was intractable for the full 9-feature set.

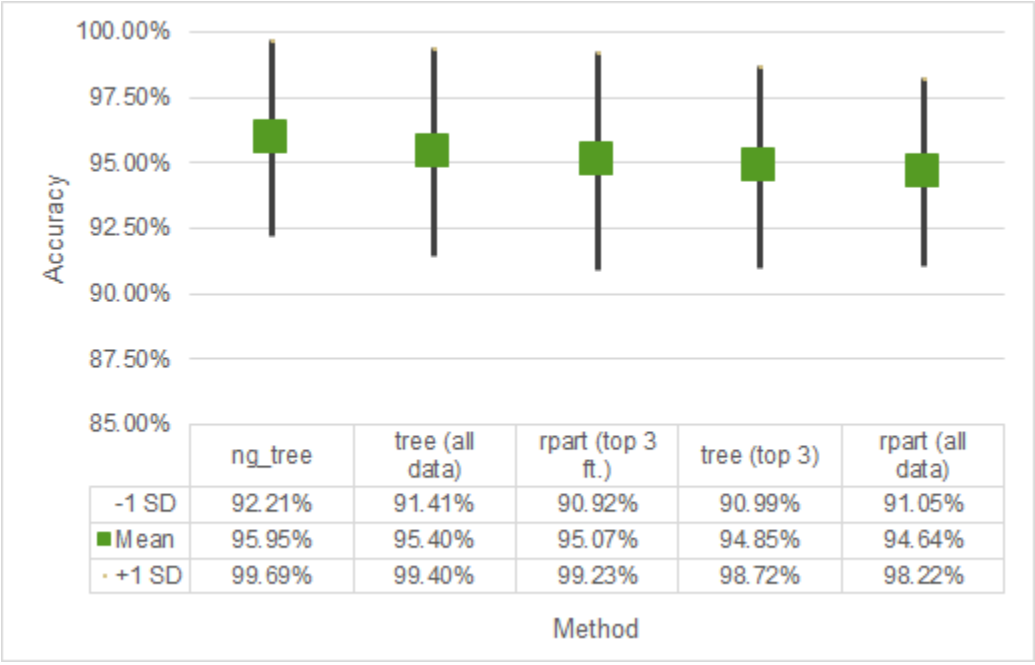


	3	4	5	6	7	8
rpart	94.49%	94.48%	93.77%	94.83%	94.85%	94.89%
tree	94.49%	94.28%	94.25%	95.35%	95.44%	95.29%
ng_tree	94.96%	95.14%	95.04%	96.32%	94.74%	93.98%

Figure 3.1: Results of different algorithms on predicting the malignancy of breast cancer. Each result at each Feature is the average of 20 rounds of trials.

To decrease computation time, the Breast Cancer data set was analyzed to find the features which lead to the highest impurity reduction. Then the reduced-dimensionality data was used as a training data set. The full data was also given to *rpart* and *tree* for comparison (Figure 3.2).

The non-greedy algorithm outperforms tree and rpart on both lower-dimension data and full data sets.



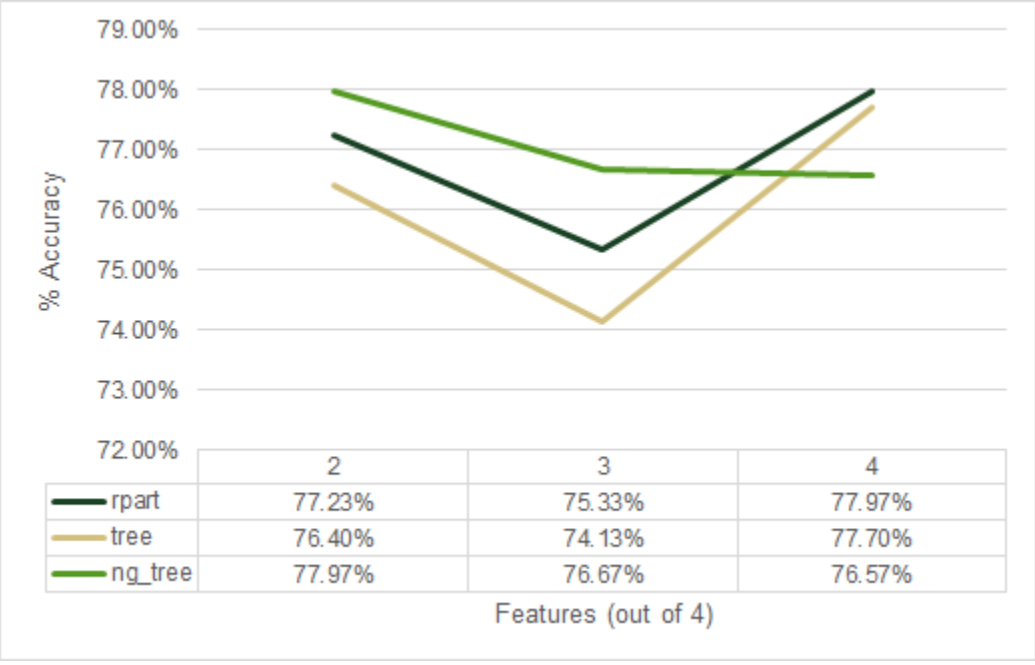
	ng_tree	tree (all data)	rpart (top 3 ft.)	tree (top 3)	rpart (all data)
-1 SD	92.21%	91.41%	90.92%	90.99%	91.05%
Mean	95.95%	95.40%	95.07%	94.85%	94.64%
+1 SD	99.69%	99.40%	99.23%	98.72%	98.22%

Figure 3.2: Results of different algorithms on predicting the malignancy of cancer data using reduced-dimensionality data. Each result is the average of 20 rounds of trials.

These results suggest the outperformance of the *ng_tree* algorithm on health-science data that has been preprocessed to have similar variance and range. This could imply strong success in other health-science data, especially data sets that lend themselves toward similar normalization.

3.3 Blood Transfusion

The blood transfusion data has four variables about past donors as well as whether the donor gave blood in March of 2007. The data is collected from the Blood Transfusion Service Center in Hsin-Chu City in Taiwan. Using four features (recency, frequency, amount of blood donated, and months since first donation), the goal is to accurately predict whether or not a person will donate in the following month.



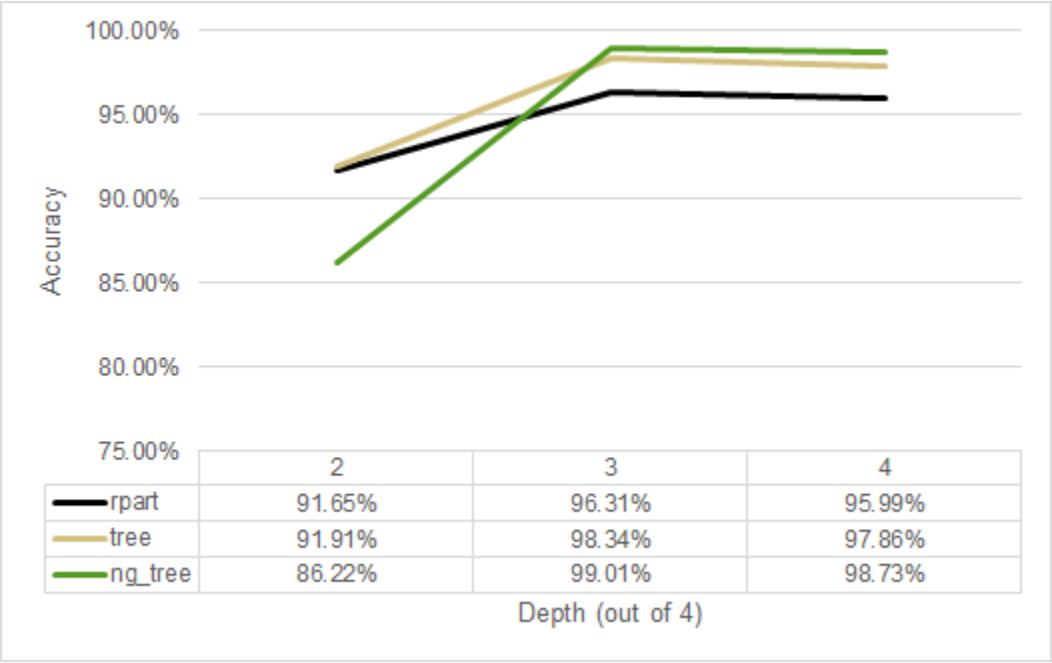
	2	3	4
rpart	77.23%	75.33%	77.97%
tree	76.40%	74.13%	77.70%
ng_tree	77.97%	76.67%	76.57%

Figure 3.3: Results of different algorithms on predicting whether a donor will give blood in the next month. Each result at each Feature is the average of 20 rounds of trials.

The *ng_tree* algorithm again finds the highest accuracy, but finds it at low depth (Figure 3.3).Unlike the Cancer data, the Blood Transfusion data is not standardized. Variables are very different in variance and range, though all variable values are positive integers. The outperformance of the *ng_tree* on this data set suggests that it may have strength in predicting behavioral data.

3.4 Banknotes Authorization

The Banknotes Authorization data contains 4 features that describe an image of a banknote. Given those features, the goal is to distinguish authentic from forged banknotes. The banknotes were modified with wavelet transformation (a tool used for compression and cleaning), and then the characteristics of the wavelet transformation were analyzed. Given those features (variance, skewness, curtosis, entropy of image) the goal is to predict the authenticity.



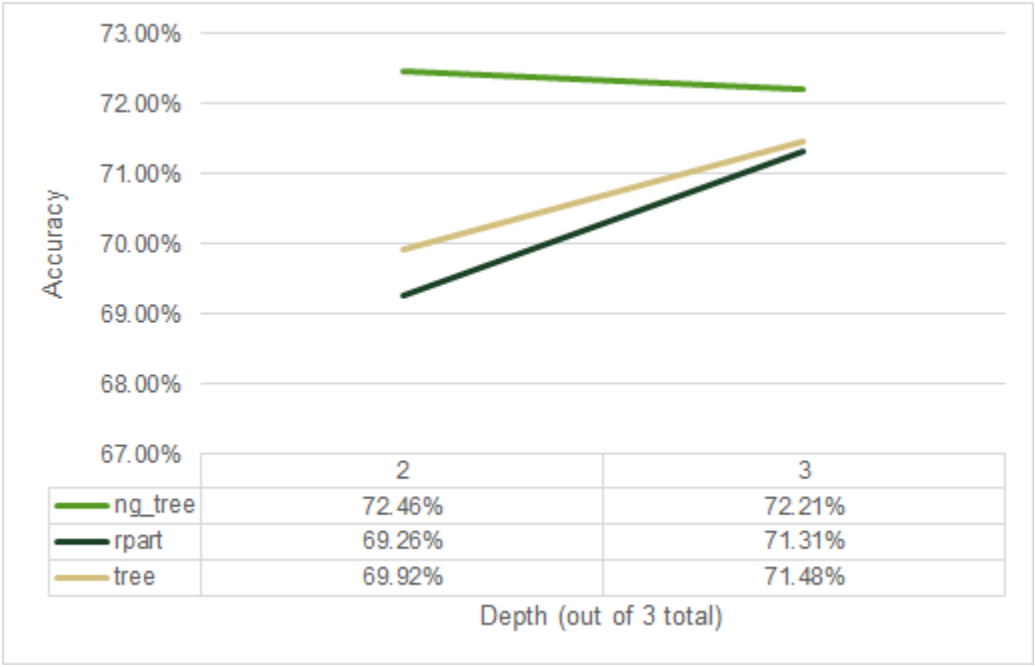
	2	3	4
rpart	91.65%	96.31%	95.99%
tree	91.91%	98.34%	97.86%
ng_tree	86.22%	99.01%	98.73%

Figure 3.4: Results of different algorithms on predicting the authenticity of a banknote. Each result at each Feature is the average of 20 rounds of trials.

The *ng_tree* algorithm found the highest accuracy prediction, again at a shallower depth than full-feature data (Figure 3.4). The banknotes data is very different from the prior data sets not only because it is image-based, but because it is meta data. Rather than examine the underlying variable itself (color of the image), the data represents the meta analysis of a transformation. The outperformance of the *ng_tree* algorithm suggests that the method used may have strong predictive power in analyzing complex meta data.

3.5 Haberman's Survival

The Haberman's survival data represents the 5-year survival rates of breast cancer patients who underwent surgery at the University of Chicago's Billings Hospital. Given three features (age of patient at time of operation, patient's year of operation, and number of positive axillary nodes detected) the goal is to predict the patients survival over the subsequent 5 years.



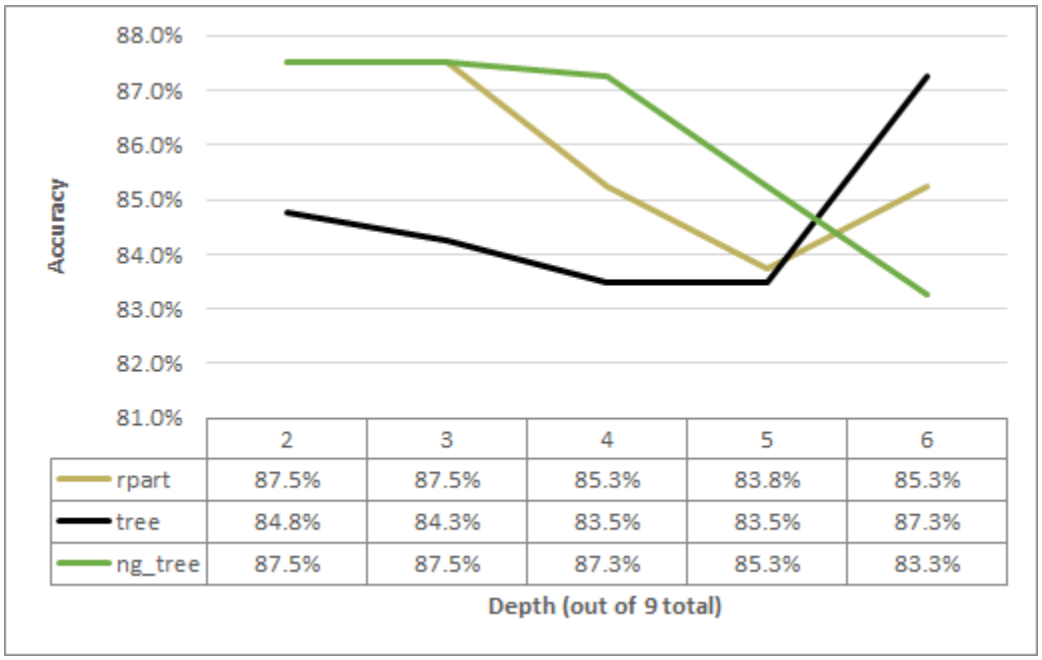
	2	3
ng_tree	72.46%	72.21%
rpart	69.26%	71.31%
tree	69.92%	71.48%

Figure 3.5: Results of different algorithms on predicting survival of post-operative breast cancer patients. Each result at each Feature is the average of 20 rounds of trials.

The *ng_tree* algorithm found the highest accuracy prediction, using only two features (Figure 3.5). This is the second data set that deals with breast cancer wherein the *ng_tree* algorithm had strong performance. There may be a characteristic of breast cancer data that lends itself well toward analysis by this non-greedy algorithm. The data may be prone to overfitting, and thus well-handled by the *ng_tree* algorithm which precludes overfitting.

3.6 Fertility

The Fertility data is drawn from a population of 100 participants who volunteered for a fertility study. Sperm samples were analyzed for normalcy of concentration. Given nine features describing a participant (season that sample was provided, age, childhood diseases, accident or trauma, surgical intervention, high fevers, frequency of alcohol consumption, smoking habit, number of hours spent sitting per day) the goal is to predict if the sample's concentration is normal or not.



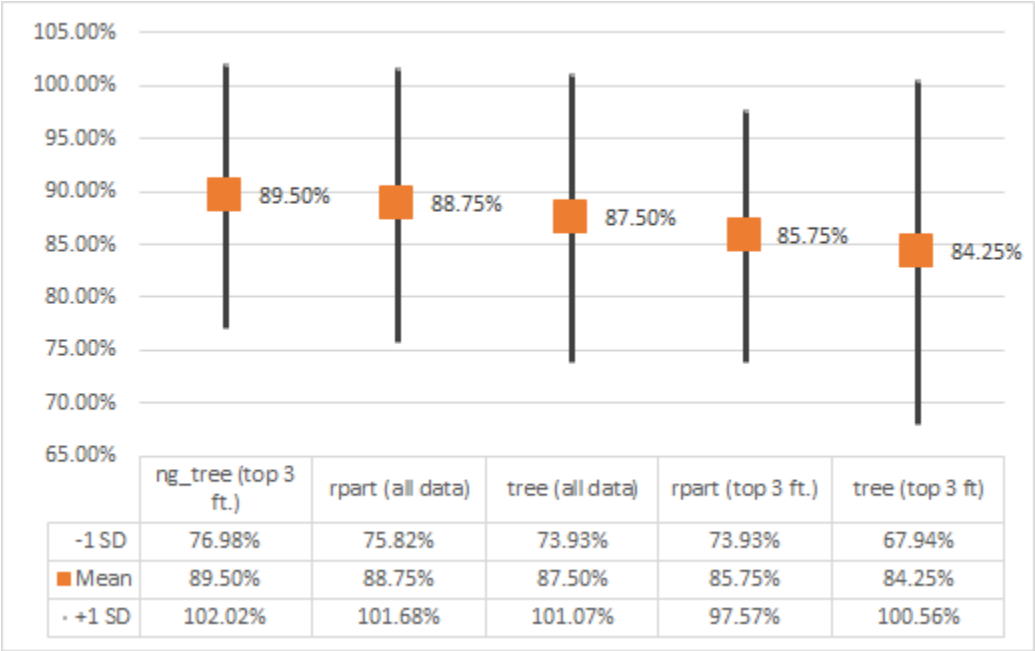
	2	3	4	5	6
rpart	87.5%	87.5%	85.3%	83.8%	85.3%
tree	84.8%	84.3%	83.5%	83.5%	87.3%
ng_tree	87.5%	87.5%	87.3%	85.3%	83.3%

Figure 3.6: Results of different algorithms on predicting the normality of sperm concentration in patients. Each result at each Feature is the average of 20 rounds of trials.

The *ng_tree* algorithm again finds the highest accuracy prediction, but it does so given only two and three features (Figure 3.6) . The accuracy falls sharply past 4 features and the program becomes intractable for larger feature sets.

As with the breast cancer data, we can reduce the dimensionality of the data and retrain the models on both reduced and full dimensinoality data sets (Figure 3.7).

As with the Breast Cancer data, the *ng_tree* algorithm outperforms *tree* and *rpart* at both full and reduced-dimension data.



	ng_tree (top 3 ft.)	rpart (all data)	tree (all data)	rpart (top 3 ft.)	tree (top 3 ft)
-1 SD	76.98%	75.82%	73.93%	73.93%	67.94%
Mean	89.50%	88.75%	87.50%	85.75%	84.25%
+1 SD	102.02%	101.68%	101.07%	97.57%	100.56%

Figure 3.7: Results of different algorithms, using reduced-dimensionality data, predicting the normality of sperm concentration in patients. Each result is the average of 20 rounds of trials.

3.7 Analysis of Results and Limitations

For a broad range of datasets covering a wide range of data types, the ng_tree algorithm consistently has the highest levels of accuracy prediction. However, the algorithm achieves this success at arbitrarily low levels of features. That is, there is no way to know *a priori* how many features one should use, when the highest accuracy will be achieved, and when accuracy will deteriorate.

The dimensionality reduction step obviates the selection problem. By simply finding which variables contain the highest impurity and training the model on those variables, one can consistently generate a strong model.

To aid with variable-count-selection (training a model using 1-n variables), I suggest reducing dimensionality by the square root of the feature count. In the cancer data and fertility data sets, models were trained on three features ($\sqrt{\text{Full 9-Feature Dataset}}$) and achieved strong performance.

Because of the semi-greedy nature of the algorithm, computation time is still a concern, and the algorithm will not be practical for large data sets (approximately 100 features or greater, depending on computing power).

CHAPTER 4

Conclusion

In this work, I have developed a method for semi-greedy optimization of multivariate oblique decision trees. The method involves multiplying the input features by a weight matrix, and solving for an optimal mapping vector which directs transversal down a symmetrical binary decision tree. True non-greedy solutions to decision trees would be $n(p)$ complete, requiring $O(n!)$ time, while this method's computation time is exponential $O(2^n)$. Still, this can be intractable for problems with high feature counts. In such cases, the method can be augmented by simple dimensionality reduction. Giving the algorithm the top n features of a data set, as determined by the highest impurity reduction, reduces the time to compute, and outperforms greedy decision trees trained on the full-feature data set. This algorithm can be made even stronger by implementation of extreme gradient boost methodologies and stochastic gradient descent.

CHAPTER 5

Appendix

The following is the R code for the GivingTree algorithm. Throughout, the training_data is a matrix X with final column equal to the target class Y.

5.1 Weight Matrix Code

```
i_weights<-apply(train_data[,1:(ncol(train_data)-1)],2,sd)
W<-matrix(rep(c(i_weights),depth),nrow=dim(train_data)[2]-1,ncol=depth,byrow=F)
```

5.2 Theta Matrix Code

```
initialize_theta<-function(data,depth){
   #  function takes:
   #     the train data
   #     an optional depth which is the number of leaves (m+1) or a specified
       depth if
   #        not using the full depth of tree. Depth must be of 2**n value
       (1,2,4,8...)

   if(missing(depth)){
      depth<-0
      i<-0
      while(i<ncol(data)-1){
         depth=depth+2**i
         i=i+1
      }
      depth<-depth+1
   }
   theta<- matrix(c(runif(2*depth**2)),nrow=depth,ncol=2,byrow=T)
   theta<-update_theta(theta)
   return(theta)
}


update_theta<-function(theta){
   # function takes as arguments:
   # an m+1-length one-hot indicator vector, which is only non-zero at the index
       of the selected leaf
   # an 1 by k length matrix of probabilities
   # function returns:
   # an m+1 by k matrix of softmax processed probabilities of p(y= l|j)
```

```
   # theta<-t(theta)
   if(!is.null(nrow(theta))){
      for (i in seq(1,nrow(theta))){
         theta_vec<-theta[i,]
         denominator<-do.call(sum,lapply(theta_vec,exp))
         new_probs<-matrix(unlist(lapply(theta_vec, function(x)
            exp(x)/denominator)))
         theta[i,]<-new_probs}
         } else {
         denominator<-do.call(sum,lapply(theta,exp))
         theta<-matrix(unlist(lapply(theta, function(x) exp(x)/denominator)))
         }
   return(theta)
}
```

5.3 Helper Functions

```
sgn<-function(W,data,row){
   # function takes:
   # W: Weight matrix
   # data: training data matrix
   # row: integer of row to be evaluated
   # function returns:
   # the sign of the bits on the path
   x = data[row,c(1:ncol(data)-1)]
   out_mat = t(W) %*% x -1
   sign  = sign(out_mat)
   return(sign)}


f <- function(h){
   # function takes as arguments:
   # an m-bit vector of potential split decisions (h)
   # function returns:
   # an m+1-length one-hot indicator vector, which is only non-zero at the index
       of the selected leaf
   h[h==-1]<-0
   l<-c(h[1])
   i=1
   while(i<=length(h)){
      n_i<- 2*i+h[i]
      i<-n_i
      l<-append(l,list(h[i]))
      }


   j<-0
   while(2**j<= i){
```

```
      j=j+1
      }
   out_pos<- i - 2**(j-1) +1

   full_vec <- rep(0, length(h)+1)
   full_vec[out_pos] <- 1

   return(full_vec)
}

accuracy<-function(theta,W,test_data){
   # function takes as arguments:
   # Weights (W)
   # dataset (test_data)
   # theta matrix (theta)
   # function returns:
   # the accuracy of the predictions
   test_theta<-update_theta(theta)
   acc<-0
   for(i in seq(1,nrow(test_data))){
      true_base = as.numeric(test_data[i,ncol(test_data)])
      predicted = grep(1,round((f(sgn(W,test_data,i))%*%(test_theta))))-1
      # cat(true_base," ",predicted,"\n")
      if(true_base==predicted){
         acc = acc+1
      }
   }
   return(acc/nrow(test_data))
}

est<-function(theta,W,data){
```

```
   # function takes as arguments:
   # Weights (W)
   # dataset (data)
   # theta matrix (theta)
   # function returns:
   # the estimated probability of each class, for each element in data

   test_theta<-update_theta(theta)
   pred<-data.frame()
   for(i in seq(1,nrow(test_data))){
      predicted = grep(1,round((f(sgn(W,test_data,i))%*%(test_theta))))-1
      pred<-rbind(pred,predicted)
   }
   return(pred)
}

loss<-function(data,samp_row,theta,g){
   # function takes as arguments:
   # the dataset and row being evaluated
   # predicted probabilities from the update_theta softmax
   # function returns:
   # log loss value
   true_base = as.numeric(data[samp_row,ncol(data)])
   probs<- t(theta) %*%f(g)
   prob<-probs[2]
   log_loss = - (sum(true_base * log(prob) + (1 - true_base) * log(1 - prob))) /
       length(true_base)
   return(log_loss)
}

loss_prime<-function(theta){
```

```
  # function takes as arguments:
  # probability matrix (theta)
  # function returns:
  # partial derivative of loss function
  a = 1/theta
  return(a)
}
```

5.4 Greedy Code

```
greedy<-function(theta,W,tau,alpha,v,train_data,exhaustive){
   # function takes
      # theta: theta matrix
      # W: weight matrix
      # tau: scalar number of rows to be sampled
      # alpha: scalar learning rate
      # v: scalar nu
      # train_data: matrix
   # function returns:
      # theta: the updated theta matrix
      # W: updated Weight
   cols<-dim(train_data)[2]-1
   for(t in seq(1,tau)){
      samp_row<-t

      h = sgn(W,train_data,samp_row)
      g = objective(W,train_data,theta,samp_row,exhaustive)
      W = W + t((alpha*(g-h)%*%t(train_data[samp_row,0:cols])))

      for(i in seq(1,nrow(W))) {
         a = min(1, v**(1/2) / (sum(W[i,]**2)**(1/2))) %*% W[i,]
         W[i,]<-a
         }

      h = sgn(W,train_data,samp_row)
      true_base = as.numeric(train_data[samp_row,ncol(train_data)])
      r<-grep(1,f(sgn(W,train_data,samp_row)))
      probs<-theta[r,]
```

```
      if (true_base==1){
         true_probs<-c(0,1)
         } else {
         true_probs<-c(1,0)
         }

      gradient = loss_prime(probs)
      theta[r,] = theta[r,] - alpha *
          (update_theta(t(gradient))*(theta[r,]-true_probs))
      theta[is.nan(theta)]=0.01
   }
   ret <-list("theta" = theta, "W"=W)
   return(ret)
}

objective<-function(W,data,theta,samp_row,exhaustive){
   # function takes as arguments:
   # Weights (W)
   # row being tested (row)
   # dataset (data)
   # theta vector
   # function returns the argument (g) that minimizes the loss
   width<-dim(W)[2]
   argmax<- 5e10
   memo<-list()
   gs<-matrix(rep(0,width**2),nrow=width,ncol=width)
   for(row in seq(0,width)){
      gs[row,row]<-1
   }

   for(row in c(1:nrow(gs))) {
```

```
      g<-gs[row,]
      x<-data[samp_row,c(1:ncol(data)-1)]
      u_p <-update_theta(theta)
      second_term = loss(data,samp_row,u_p,g)
      if(second_term<argmax){
         argmax<-second_term
         argmax_row<-row
         }
      }

   return(gs[argmax_row,])
}
```

5.5 Non-Greedy Code

```
non_greedy<-function(theta,W,tau,alpha,v,train_data){
# function takes
   # theta: theta matrix
   # W: weight matrix
   # tau: scalar number of rows to be sampled
   # alpha: scalar learning rate
   # v: scalar nu
   # train_data: matrix
# function returns:
   # theta: the updated theta matrix
   # W: updated Weight
   cols<-dim(train_data)[2]-1
   for(t in seq(1,tau)){
      samp_row <-sample(1:nrow(train_data),1)
      h = sgn(W,train_data,samp_row)

      g_v = objective_verbatum(W,train_data,theta,samp_row)
      g_o = objective(W,train_data,theta,samp_row)
      W = (W
         + t(alpha*(g_o-h)%*%t(train_data[samp_row,0:cols]))
         - t(alpha*(g_v)%*%t(train_data[samp_row,0:cols]))
         )
      h = sgn(W,train_data,samp_row)

   for(i in seq(1,nrow(W))) {
      a = min(1, v**(1/2) / (sum(W[i,]**2)**(1/2))) %*% W[i,]
      W[i,]<-a
      }
```

```
   true_base = as.numeric(train_data[samp_row,ncol(train_data)])
   r<-grep(1,f(sgn(W,train_data,samp_row)))
   probs<-theta[r,]

   if (true_base==1){
      true_probs<-c(0,1)
      } else {
      true_probs<-c(1,0)
      }

   gradient = loss_prime(probs)
   theta[r,] = theta[r,] - alpha *
       (update_theta(t(gradient))*(theta[r,]-true_probs))
   theta[is.nan(theta)]=0.01
   }
   ret <-list("theta" = theta, "W"=W)
   return(ret)
}

objective_verbatum<-function(W,data,theta,samp_row){
   # this is the surrogate objective function
   # function takes as arguments:
   # Weights (W)
   # row being tested (row)
   # dataset (data)
   # theta vector
   # function returns the argument (g) that maximizes the loss
   width<-dim(W)[2]
   memo<-list()
   gs<-matrix(rep(0,width**2),nrow=width,ncol=width)
   for(row in seq(0,width)){
```

```
      gs[row,row]<-1
   }
   second_max<-0
   first_max<-0
   argmax<- -1e10
   for(row in c(1:nrow(gs))) {
      g<-gs[row,]
         x<-data[samp_row,c(1:ncol(data)-1)]
         first_term = sum((sgn(W,data,samp_row) - g)^2)
         first_term = g%*%t(W)%*%x
         u_p <-update_theta(theta)
         second_term = loss(data,samp_row,u_p,g)
         val <- second_term
            if(
               val>argmax
               ){
               argmax_row<-row
               argmax<-val
               second_max<-second_term
               first_max<-first_term
            }
   }
   return(gs[argmax_row,])
}
```

Bibliography

Balestriero, R. (2017). Neural decision trees. *arXiv preprint arXiv:1702.07360*.

Banknote authentication data set. (2013, Apr). Retrieved from `https://archive.ics.uci.edu/ml/datasets/banknote+authentication`

Bellacicco, A. (1984). *Handbook of statistics 2: Classification, pattern recognition and reduction of dimensionality: Pr krishnaiah and ln kanal (eds.) north-holland, amsterdam, 1982, xxii+ 903 pages, dfl. 275.00.* North-Holland.

Ben-Bassat, M. (1982). 35 use of distance measures, information measures and error bounds in feature evaluation. *Handbook of statistics*, *2*, 773–791.

Bennett, K. P. (1994). Global tree optimization: A non-greedy decision tree algorithm. *Computing Science and Statistics*, 156–156.

Biau, G., & Scornet, E. (2016). A random forest guided tour. *Test*, *25*(2), 197–227.

Blood transfusion service center data set. (2008, Oct). Retrieved from `https://archive.ics.uci.edu/ml/datasets/Blood+Transfusion+Service+Center`

Breast cancer wisconsin (diagnostic) data set. (1995, Nov). Retrieved from `https://archive.ics.uci.edu/ml/datasets/BreastCancerWisconsin(Diagnostic)`

Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*(2), 123–140.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). Classification and regression trees. wadsworth int. *Group*, *37*(15), 237–251.

Commons, W. (2018). *File:cart tree kyphosis.png — wikimedia commons, the free media repository.* Retrieved from `\url{https://commons.wikimedia.org/w/index.php?title=File:Cart_tree_kyphosis.png&oldid=296397192}` ([Online; accessed 3-March-2019])

Esmeir, S., & Markovitch, S. (2006). When a decision tree learner has plenty of time. In *Proceedings of the national conference on artificial intelligence* (Vol. 21, p. 1597).

Feng, J., Yu, Y., & Zhou, Z.-H. (2018). Multi-layered gradient boosting decision trees. In *Advances in neural information processing systems* (pp. 3555–3565).

Fertility data set. (2013, Jan). Retrieved from `https://archive.ics.uci.edu/ml/`

datasets/Fertility

Friedman, J. H. (1977). A recursive partitioning decision rule for nonparametric classification. *IEEE Transactions on Computers*(4), 404–408.

Gieseke, F., & Igel, C. (2018). Training big random forests with little resources. In *Proceedings of the 24th acm sigkdd international conference on knowledge discovery & data mining* (pp. 1445–1454).

Haberman's survival data set. (1999, Mar). Retrieved from `https://archive.ics.uci.edu/ml/datasets/Haberman's+Survival`

Heath, D., Kasif, S., & Salzberg, S. (1993). Induction of oblique decision trees. In *Ijcai* (Vol. 1993, pp. 1002–1007).

Hunt, E. B., Marin, J., & Stone, P. J. (1966). Experiments in induction.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning* (Vol. 112). Springer.

Khiari, J., Moreira-Matias, L., Shaker, A., Ženko, B., & Džeroski, S. (2018). Metabags: Bagged meta-decision trees for regression. In *Joint european conference on machine learning and knowledge discovery in databases* (pp. 637–652).

Klusowski, J. M. (2018). Complete analysis of a random forest model. *arXiv preprint arXiv:1805.02587*.

Kononenko, I., Bratko, I., & Roskar, E. (1986). Assistant: A system for inductive learning. *Informatica*, *10*.

Laurent, H., & Rivest, R. L. (1976). Constructing optimal binary decision trees is np-complete. *Information processing letters*, *5*(1), 15–17.

Loh, W.-Y., & Shih, Y.-S. (1997). Split selection methods for classification trees. *Statistica sinica*, 815–840.

Murthy, S. K. (1998). Automatic construction of decision trees from data: A multi-disciplinary survey. *Data mining and knowledge discovery*, *2*(4), 345–389.

Murthy, S. K., Kasif, S., & Salzberg, S. (1994). A system for induction of oblique decision trees. *Journal of artificial intelligence research*, *2*, 1–32.

Norouzi, M., Collins, M., Johnson, M. A., Fleet, D. J., & Kohli, P. (2015). Efficient

non-greedy optimization of decision trees. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems 28* (pp. 1729–1737). Curran Associates, Inc. Retrieved from `http://papers.nips.cc/paper/5886-efficient-non-greedy-optimization-of-decision-trees.pdf`

Norton, S. W. (1989). Generating better decision trees. In *Ijcai* (Vol. 89, pp. 800–805).

Paterson, A., & Niblett, T. (1982). Acls manual, version 1. *Rapport technique*.

Quinlan, J. (1985). Induction of decision trees,[broadway, nsw, australia]: New south wales institute of technology. *School of Computing Sciences*.

Reis, I., Baron, D., & Shahaf, S. (2018). Probabilistic random forest: A machine learning algorithm for noisy data sets. *The Astronomical Journal*, *157*(1), 16.

Ripley, B. (2018, Mar). *Classification and regression trees.* Retrieved from `https://cran.r-project.org/web/packages/tree/tree.pdf`

Ritzman, L. P., Krajewski, L. J., & Klassen, R. D. (2004). *Foundations of operations management*. Toronto.: Pearson Prentice Hall.

Robertson, B., Price, C., & Reale, M. (2013). Cartopt: a random search method for non-smooth unconstrained optimization. *Computational Optimization and Applications*, *56*(2), 291–315.

Siu, C. (2018). Automatic induction of neural network decision tree algorithms. *arXiv preprint arXiv:1811.10735*.

Spanakis, G., Weiss, G., & Roefs, A. (2016). Bagged boosted trees for classification of ecological momentary assessment data. In *Proceedings of the twenty-second european conference on artificial intelligence* (pp. 1612–1613).

The state of ml and data science 2017. (2017). Retrieved from `https://www.kaggle.com/surveys/2017`

Steele, K., & Stefánsson, H. O. (2015). Decision theory.

Taggart, A. J., DeSimone, A. M., Shih, J. S., Filloux, M. E., & Fairbrother, W. G. (2012). Large-scale mapping of branchpoints in human pre-mrna transcripts in vivo. *Nature structural & molecular biology*, *19*(7), 719.

Tanno, R., Arulkumaran, K., Alexander, D. C., Criminisi, A., & Nori, A. (2018). Adaptive

neural trees. *arXiv preprint arXiv:1807.06699*.

Therneau, T., Atkinson, B., & Ripley, B. (2018, Feb). *Recursive partitioning and regression trees.* Retrieved from `https://cran.r-project.org/web/packages/rpart/rpart.pdf`

Therneau, T. M., Atkinson, E. J., et al. (1997). *An introduction to recursive partitioning using the rpart routines.* Technical Report 61. URL http://www. mayo. edu/hsr/techrpt/61. pdf.

Top 10 machine learning algorithms. (n.d.). Retrieved from `https://www.dezyre.com/article/top-10-machine-learning-algorithms/202`

Wickramarachchi, D., Robertson, B., Reale, M., Price, C., & Brown, J. (2016). Hhcart: An oblique decision tree. *Computational Statistics & Data Analysis*, *96*, 12–23.

Wu, X., & Kumar, V. (2009). *The top ten algorithms in data mining.* CRC press.

Yang, Y., Morillo, I. G., & Hospedales, T. M. (2018). Deep neural decision trees. *arXiv preprint arXiv:1806.06988*.