# Learning Certifiably Optimal Rule Lists


A THESIS PRESENTED
BY
NICHOLAS L. LARUS-STONE
TO
THE DEPARTMENT OF COMPUTER SCIENCE

HARVARD UNIVERSITY
CAMBRIDGE, MASSACHUSETTS

MAY 2017

Thesis advisor: Professor Margo Seltzer Nicholas L. Larus-Stone

# Learning Certifiably Optimal Rule Lists


## Abstract

We demonstrate a new algorithm that finds an optimal rule list as well as providing proof of that optimality. Rule lists, which are lists composed of If-Then statements, are similar to decision tree classifiers and are useful because each step in the model's decision making process is understandable by humans. Our algorithm, called CORELS, finds the optimal rule list through the use of three types of bounds: bounds inherent to the rules themselves, bounds based on the current best solution, and bounds based on symmetry between rule lists. We propose novel data structures to minimize the memory usage and runtime of our algorithm on this exponentially difficult problem. One advantage of our algorithm is that it generates not only the optimal rule list and a certificate of optimality, but also the entire solution space of nearly optimal solutions. Our algorithm therefore allows for the analysis and discovery of all optimal and near-optimal solutions on problems requiring human-interpretable algorithms.

# Contents

# 0

# Introduction

As machine learning continues to grow in importance, the interpretability of predictive models remains a crucial problem. Our goal is to build models that are highly predictive but in which each step of the model's decision making process can also be understood by humans. Machine learning models such as neural nets or support vector machines are able to achieve stunning predictive accuracy, but the reasons for these predictions remain unintelligible to a human user. This lack of

interpretability is important because models that are not understood by humans can have hidden bias in their predictive decision making. A recent ProPublica article found racial bias in the use of black box machine learning models used for advising criminal sentencing[9]. Northepointe, the company which provides a black box model called COMPAS argues that their use of a black box model is necessitated by the fact that they can achieve better accuracy through the use of that model. This thesis is part of a body of work that hopes to prove that interpretability can be achieved without sacrificing accuracy. In particular, this thesis will examine the data structure optimizations used to run this algorithm efficiently.

To achieve interpretability, we use *rule lists*, also known as decision lists, which are lists comprised of *if-then* statements[20]. This structure allows for predictive models that also can be easily interpreted because the rules that are satisfied give a reason for each prediction. Given a set of rules associated with a dataset, rule lists can be constructed by placing the rules in different orders. Since most data points are captured by multiple rules, changing the order of rules leads to different predictions and therefore different accuracies. Different rule list algorithms all attempt to maximize predictive accuracy through the discovery of different rules lists.

Pitting interpretable models against black box ones raises the question of how accurate the interpretable models can be. Thus, searching for an optimal model to provide an upper bound on the potential accuracy of an interpretable model is crucial when discussing whether or not to use interpretable models. In our case, we are searching for the rule list with the highest accuracy, the optimal rule list. A brute force solution to find the the optimal rule list would be computationally prohibitive due to the combinatorially number of rule lists. Our goal is to create an algorithm to

find the optimal rule list in a reasonable amount of time.

Recent work on generating rule lists[12,22] uses probabilistic approaches to generating rule lists. These approaches achieve high accuracy while also managing to run quickly. However, despite the apparent accuracy of the rule lists generated by these algorithms, there is no way to determine if the generated rule list is optimal or how close to optimal the rule lists is. Our model, called Certifiably Optimal RulE ListS (CORELS), finds the optimal rule list and allows us to also investigate the accuracy of near optimal solutions. The benefits of this model are two-fold: firstly, we are able to generate the best rule list on a given data set and therefore will have the most accurate predictions that a rule list can give. Secondly, since CORELS generates the entire space of potential solutions, we can judge how good rule lists generated by other algorithms are. In particular, we can investigate if the rule lists from probabilistic approaches are nearly optimal or whether those approaches sacrifice too much accuracy for speed. This will allows us to bound the accuracy on important problems and determine if interpretable methods shuold be used.

CORELS achieves these results by placing a bound on the best performance that a rule list can achieve in the future. This allows us to prune that rule list if that bound is worse than the objective value of the best rule list that we have already examined. We continue to look at rule lists until we have either examined every rule list or eliminated all but one from consideration. Thus, when the algorithm terminates, we have found the rule list with the best possible accuracy. Our use of this branch and bound technique leads to massive pruning of the search space of potential rule lists and means our algorithm can find the optimal rule list on real data sets.

Due to our interest in interpretability, the amount of data each rule captures informs the value of

that rule. We want our rule lists to be understandable by humans, so shorter rule lists are more optimal. Therefore, we use an objective function that takes into account both accuracy and the length of the rule list to prevent overfitting. This means we may not always find the highest accuracy rule list–our optimality is over both accuracy and length of rule lists. This requires each rule to capture a certain amount of data correctly to make it worth the penalty of making our rule list longer. This limits the overall length of our rule lists and prevents us from investigating rule lists containing useless rules.

The exponential nature of the problem means that the efficacy of CORELS is dependent on how much our bounds allow us to prune. We list a few types of bounds that allow us to drastically prune our search space. The first type of bound is intrinsic to the rules themselves. This category includes bounds such the bound described above that ensures rules capture enough data correctly to overcome a regularization parameter. Our second type of bound compares the best future performance of a given rule list to the best solution encountered so far. We can avoid examining parts of the search space whose maximum possible accuracy is less than the accuracy of our current best solution. Finally, our last class of bounds uses a symmetry-aware map to prune all but the best permutation of any given set of rules.

To keep track of all of these bounds for each rule list, we implemented a modified trie that we call a prefix tree. Each node in the prefix tree represents an individual rule; thus, each path in the tree represents a rule list where the final node in the path contains the metrics about that rule list. This tree structure facilities the use of multiple different selection algorithms including breadth-first search, a priority queue based on a custom function that trades off exploration and exploitation, and

a stochastic selection process. In addition, we are able to limit the number of nodes in the tree and thereby achieve a way of tuning space-time tradeoffs in a robust manner. We propose that this tree structure is a useful way of organizing the generation of rule lists and allows the implementation of CORELS to be easily parallelized.

We evaluated CORELS on a number of datasets from the UCI repository. Our metric of success was prediction accuracy on a subset of the data which we calculated using 10-fold cross validation. These datasets involve hundreds of rules and hundreds or thousands of samples and CORELS is able to find the optimal rule list within 10 minutes. We show that we are able to achieve better accuracy on these datasets than the popular greedy algorithms, CART or C4.5.

In addition, we applied CORELS to the problem of predicting criminal recidivism. Larson et al examines the problem of predicting recidivism and shows that a black box model, specifically the COMPAS score from the company Northpointe, has racially biased prediction[9]. Black defendants are misclassified at a higher risk for recidivism than in actuality, while white defendants are misclassified at a lower risk. The model which produces the COMPAS scores is a black box algorithm which is not interpretable, and therefore the model does not provide a way for human input to correct for these racial biases. Our model produces similar accuracies to the predictive models and COMPAS scores from Larson et al while maintaining its interpretability.

CORELS demonstrates a novel approach towards generating interpretable models by looking for the optimal rule list. While searching for that optimal list, we are able to discover near-optimal solutions that provide insight into how effective other interpretable methods might be. We have proved numerous bounds that allow us to prune rule lists. Finally, we created a novel symmetry-

aware map that drastically reduces the search space of possible rule lists.

Chapter 2 provides an overview of related work in the fields of rule lists, interpretable models, and discrete optimization. Chapter 3 proves definitions that underlie the execution of the algorithm. Chapter 4 describes the implementation of the algorithm, emphasizing the data structures used to make this problem tractable. Chapter 5 describes experiments run to test the space/time tradeoffs detailed in chapter 4.

# 1

# Related Work

The use of classification models is popular in a number of different fields from image recognition to churn prediction. Oftentimes, however, simply receiving a prediction from software is not enough– it is important to have a predictive model that humans can investigate and understand [21,3,19,15,7]. For example, in fields such as medical diagnoses [2] and criminal sentencing [9], it is important to be able to investigate the reasons behind a model's predictions. One reason is that medical experts are unlikely

to trust the predictions of these models if they are unable to understand why the model is making certain predictions[11]. Interpretable models also allow users to examine predictions to detect systemic biases in the model. This is especially important in classification problems such as criminal recidivism prediction where there are often race-related biases[9] or credit scoring where a justification is necessary for the denial of credit[1].

Tree structured classifiers are a popular technique that combines interpretability with a high predictive accuracy. Also called decision trees, these trees are often used as either classification or regression tools. Every node in the tree classifier splits the data into two subsets; these subsets are then recursively split by nodes lower in the tree. Nodes are constructed by choosing an attribute that splits the data in such a way that it that minimizes the impurity of each subset. Trees are constructed by recursively performing splits on the child subsets until the resulting subset is entirely homogenous or small enough. Methods for constructing decision trees differ primarily based on how they define impurity and therefore what attributes they choose for each node. In Classification and Regression Trees (CART), Breiman et al lay out an algorithm to create these trees[4]. CART tries to minimize Gini impurity which is a measure of the probability that any random element taken from a node is mislabeled. Another popular algorithm, C4.5, uses the idea of information gain to make its splits instead[18]. In C4.5, nodes are chosen in such a way that each split minimizes the amount of information necessary to reconstruct the original data. Both algorithms grow the initial tree greedily and then prune later to avoid overfitting.

While most decision trees are constructed greedily, and thus sub-optimally, there has been some work on constructing optimal decision trees[16]. There has even been the use of a branch and bound

technique in an attempt to construct more optimal decision trees. Garofalakis et al introduce an algorithm to generate more interpretable decision trees by allowing constraints to be placed on the size of the decision tree[8]. They use the branch-and-bound technique to constrain the size of the search space and limit the eventual size of the decision tree. During tree construction, they bound the possible Minimum Description Length (MDL) cost of every different split at a given node. If every split at that node is more expensive than the actual cost of the current subtree, then that node can be pruned. In this way, they were able to prune the tree while constructing it instead of just constructing the tree and then pruning at the end. However, even with the added bounds, this approach did not yield globally optimal decision trees because they constrained the number of nodes in the tree.

Whereas decision trees are always grown from the top downwards, decision lists are built while looking at the entire pool of rules. Thus, while decision trees are often unable to achieve optimal performance even on simple tasks such as determining the winner of a tic-tac-toe game, decision lists can achieve globally optimal performance. Decision lists are a generalization of decision trees since any decision tree can be converted to a decision list through the creation of rules to represent the leaves of the decision tree[20]. Thus, decision list algorithms are a direct competitor to the popular interpretable methods detailed above: CART and C4.5. Indeed, decision list algorithms are being used for a number of real world applications including stroke prediction[12], suggesting medical treatments[23], and text classification[13].

Work in the field of decision lists focuses both on the generation of new theoretical bounds and the improvement of predictive accuracy of models. Recent work on improving accuracy has led to

the creation of probabilistic decision lists that generate a posterior distribution over the space of potential decision lists[12,22]. These methods achieve good accuracy while maintaining a small execution time. In addition, these methods improve on existing methods such as CART or C4.5 by optimizing over the global space of decision lists as opposed to searching for rules greedily and getting stuck at local optima. Letham et al are able to do this by pre-mining rules, which reduces the search space from every possible split of the data to a discrete number of rules. We take the same approach towards optimizing over the global search space, though we don't use probabilistic techniques. We also want to work in a regime with a discrete number of rules, thus we use the same rule mining framework from Letham et al to generate the rules for our data sets. This framework creates features from the raw binary data and then builds rules out of those features. Yang et al builds on this earlier work by placing additional bounds on the search space and creating a fast low-level framework for computation, specifically a high performance bit vector manipulation library. We use that bit vector manipulation library to help perform computations involving calculating accuracy of rules.

Our use of a branch and bound technique is inspired by the fact that it is often applied to problems that have a large number of potential solutions without a polynomial time algorithm. The branch and bound algorithm recursively splits the data into subgroups, yielding a tree-like structure. Then, by calculating a value corresponding to the end goal of the algorithm (e.g. accuracy), some branches of the tree can be proved to be worse in every case than another branch and therefore can be pruned, reducing the search space. This technique has been used to solve NP-hard problems such as the Traveling Salesman Problem[14], the Knapsack Problem[10], or the Mixed Integer Programming problems[5]. Branch and Bound is also used as an optimization technique for some machine learning

algorithms, though it has not been applied to decision lists before now[17].

Here we present some of definitions of concepts and terms that are used throughout this work.

### 1.0.1 Rules

Rules are IF-THEN statements consisting of a boolean antecedent and a classification label. We are working in the realm of binary classification, so the label is either a 0 or a 1. The boolean antecedents are generated from the rule mining mechanism and can be a conjunction of boolean clauses. These antecedents are satisfied by some data points and not for others. We say a rule *classifies* a given data point when the antecedent is satisfied for that data point. As we combine these rules into rule lists, only the first rule that classifies any given data point can make a prediction for that data point. Thus, we say a rule *captures* a given data point if it is the first rule in a rule list to classify that data point.

### 1.0.2 Rule Lists

A rule list is an ordered collection of rules. As defined above, rules have inherent accuracies based on what data they classify and how they predict the label, but that they are judged based on what data they capture. However, they can perform better or worse than their inherent accuracy depending on what rules come before them in a given rule list. Thus, any model building technique is focused on finding a rule list that maximizes predictive accuracy. A rule list also has a *default rule*, placed at the end of all of the pre-mined rules, that classifies all data points and predicts the majority label. We refer to the set of rules that compose a rule list, not including the default rule, as a *prefix*. A rule list makes prediction for all points because any point not captured by the prefix is therefore captured by the default rule.

$(age = 23 - 25) \wedge (priors = 2 - 3)$ *yes* $(age = 18 - 20)$ *yes* $(sex = male) \wedge (age = 21 - 22)$ *yes* $(priors > 3)$ *yes no*

### 1.0.3 Objective

Rule lists have a loss function based on the number of points that are misclassified by the rules in the rule list. We define our objective function to be the sum of that loss and a regularization term. We use a regularization term, which is a constant times the length of the rule list, to prevent our rule lists from growing too long and therefore losing their interpretability. This objective function is what we are going to optimize over, and it is inversely correlated to the accuracy of the rule list.

### 1.0.4 Bounds

We use the discrete-optimization technique of branch-and-bound to solve this combinatorially difficult problem. This requires tight bounds that allow us to prune as much of the search space as possible. These bounds are formalized and proved in Angelino et al.[6] and are listed in Appendix A. For clarity's sake we present summaries of the important bounds here.

#### Lower Bound

We use the term *lower bound* to mean the best possible outcome for the objective function for a given prefix. We do this by calculating the error of the prefix and assuming that any points uncaptured by the prefix will be predicted correctly. Because any future extensions of the prefix can never do better than this lower bound, we will be able to use it to prune our exploration.

0.10

## Objective Bound

The first, and most important bound, is the objective bound. This is the main bound for the branch-and-bound that says that we do not need to purse a rule list if it has a lower bound on its objective that is worse than the best objective we have already seen. This allows us to prune large parts of the search space by not pursuing rule lists that could never be better than something we've already seen.

## Permutation Bound

As defined above, every sample is captured by precisely one rule–any sample that is caught by rule A in the rule list AB cannot be caught by rule B. Now consider a permutation of the rule list AB: the rule list BA. Any samples that are captured by either rule A or B but not both will be captured identically in both rule lists. Samples that are captured by both rules will again be captured the same in both rule lists, though they may be predicted differently in the two rule lists. Thus, regardless of the order in which the rules appear, rule lists AB and BA will capture exactly the same set data. They will differ only in which rules capture which samples and therefore their accuracy may differ. We can use this knowledge to create a bound as follows. If we know that the lower bound of AB is better than the lower bound of BA, we can eliminate from consideration all rule lists beginning with BA. This is due to the fact that any corresponding rule list beginning with AB will capture exactly the same samples as the equivalent rule list beginning with BA but will have a better objective function. We can eliminate all but one permutation of a given set of rules using this principle.

### Support Bound

Due to our regularization term in calculating our objective function, adding a rule that does little to help our accuracy will actually be harmful to the overall objective score. This allows us to place a bound intrinsic to the rule we're proposing adding. We only consider adding rules that have capture enough data points correctly to overcome the regularization term. Even though all rules have at least that support in the first case, as our rule lists get longer many rules do not capture enough points that haven't already been captured.

### Equivalent Points Bound

This bound relies on the structure of our dataset. In our dataset, we may encounter two data points that have the same features but different labels. Thus, any rule that classifies one of the data points will also classify the other data point, but it is impossible to correctly predict both data points. For a given class of equivalence points, therefore, we know that we will mispredict all of the points with a minority label. We can thus update our lower bound to, by assuming that all uncaptured data will be correctly predicted only if it is not an equivalent point with a minority label. This gives us much tighter lower bounds and in practice allows us to prune much more efficiently.

## 1.0.5 Curiosity

We have a number of different ways to explore the search space (see ??). Some of them, such as BFS prioritize exploration over exploitation. Others, such as ordering by lower bound ensure that we

are purely exploiting the best prefixes that we've seen. We define a new metric, *curiosity*, that is a function of both the lower bound and the number of samples captured. This allows it to blend together both exploration and exploitation.

### 1.0.6 Remaining Search Space

Throughout this work, we will want to track how effective our bounds and other optimizations are. One metric for tracking that will be seeing how quickly we reduce the remaining search space. We start with a combinatorially large search space, but quickly prune it down. The way we calculate the currently remaining search space is by looking at all of the leaves of the queue and seeing how much each leaf could potentially be expanded. Due to our regularization term, we are able to bound the maximum length of a prefix as our best objective gets updated.

# 2

# Implementation

This chapter lays out the design and implementation, of the system used to generate optimal rule lists. We begin by describing each of the three main data structures used to run our algorithm–a prefix trie, a symmetry-aware map, and a queue. Then, we

### 2.0.7 Prefix Trie

The prefix trie is a custom C++ class which is used as a cache to keep track of rule lists we have already evaluated. Each node in the trie contains the metadata associated with that corresponding rule list. This metadata includes bookkeeping information such as what child rule lists are feasible as well as information such as the lower bound and prediction for that rule list. In addition to our BaseNode class, there are two different node types that we use in our algorithm. Firstly, we implement a CuriousNode class that has an additional field that tracks the curiosity of a given prefix. The curiosity metric is the one described in Chapter ??. The curiosity is used to determine the order that prefixes on the queue are explored. Secondly, we implement a CapturedNode class that extends on the CuriousNode by having an additional field keeping track of the captured vector for that prefix. This allows us to speed up our incremental computations at the expense of using more memory per node (otherwise we have to recompute this vector every time we have a computation).

### 2.0.8 Queue

The various node types determine what type of search policy we use to explore the search space. We use a STL C++ priority queue to hold all of the leaves of the trie that still need to be explored. We implement a number of different scheduling schemes including BFS, DFS, and a priority queue. The priority queue can be ordered by curiosity, the objective of a prefix, or the lower bound of a prefix. We also have a stochastic exploration process that bypasses the use of a queue by walking down the trie, randomly choosing a child each time, until a leaf is chosen to be explored. We find

that ordering by curiosity will often, but not always, lead to a faster runtime than using BFS.

Since we do not have access to the container underlying the queue, we cannot randomly access elements in the queue, even if we have a pointer to the leaf node in the prefix trie. Thus, we run into a problem where we may delete something in the prefix trie that is currently in the queue, but have no way to update the queue. We deal with this by lazily deleting nodes from the prefix trie. This means we mark the node as deleted in the trie, but don't actually delete the physical node until it has been popped off of the queue. As Fig 2.1 shows, this leads to a situation where our logical queue is actually smaller than the physical queue.

### 2.0.9 Symmetry-aware map

We implement the symmetry-aware map as an STL unordered_map. We have two different versions of the map that both allow identical permutations to map to the same key. In both cases, the values contain the actual ordering of the rules that represents the current best prefix for that permutation. In the first version, keys to the map are the canonical order of rule lists: i.e. the lists 3-2-1 and 2-3-1 both map to 1-2-3. The second version has keys that represent the captured data points. These are equivalent for the rule lists 3-2-1 and 2-3-1, so both will map to the same key. Most data sets have a large amount of data, so the keys representing the canonical ordering are usually more efficient. In general, for long runs of our algorithm, the permutation map dominates our memory usage.

### 2.0.10 Incremental execution

Our program executes as follows. While there are still leaves of the trie to be explored, we use our scheduling policy to select the next rule list to evaluate. Then, for every rule that is not already in this rule list, we calculate its lower bound, objective, and other metrics that would occur if the rule were added to the end of the rule list. If the lower bound is greater than the minimum objective, meaning this new rule list could never be better than one we have already seen, we don't insert the new rule list into the tree or queue. We next check if there is a better permutation of this rule list in the permutation map and discard the new rule list if it is not better than the rule list we already have in the permutation map. If our new rule list is better, then we insert it into the permutation map, the tree, and the queue and update the minimum objective if necessary.

Every time we update the minimum objective, we garbage collect on the trie. We do this by traversing from the root the all of the leaves, deleting any subtrees of nodes with a lower bound that is larger than the minimum objective. In addition, if we encounter a node with no children, we prune upwards–deleting that node and recursively traversing the tree towards the root, deleting any childless nodes. This garbage collection allows us to limit the memory usage of the trie, though garbage collection is not triggered too often.

[t!] Input: Objective function

# 3

# Experiments

# 4

# Conclusion

# A

# Some extra stuff

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Morbi commodo, ipsum sed pharetra gravida, orci magna rhoncus neque, id pulvinar odio lorem non turpis. Nullam sit amet enim. Suspendisse id velit vitae ligula volutpat condimentum. Aliquam erat volutpat. Sed quis velit. Nulla facilisi. Nulla libero. Vivamus pharetra posuere sapien. Nam consectetuer. Sed aliquam, nunc eget euismod ullamcorper, lectus nunc ullamcorper orci, fermentum bibendum enim nibh eget ip-

sum. Donec porttitor ligula eu dolor. Maecenas vitae nulla consequat libero cursus venenatis. Nam magna enim, accumsan eu, blandit sed, blandit a, eros.

Quisque facilisis erat a dui. Nam malesuada ornare dolor. Cras gravida, diam sit amet rhoncus ornare, erat elit consectetuer erat, id egestas pede nibh eget odio. Proin tincidunt, velit vel porta elementum, magna diam molestie sapien, non aliquet massa pede eu diam. Aliquam iaculis. Fusce et ipsum et nulla tristique facilisis. Donec eget sem sit amet ligula viverra gravida. Etiam vehicula urna vel turpis. Suspendisse sagittis ante a urna. Morbi a est quis orci consequat rutrum. Nullam egestas feugiat felis. Integer adipiscing semper ligula. Nunc molestie, nisl sit amet cursus convallis, sapien lectus pretium metus, vitae pretium enim wisi id lectus. Donec vestibulum. Etiam vel nibh. Nulla facilisi. Mauris pharetra. Donec augue. Fusce ultrices, neque id dignissim ultrices, tellus mauris dictum elit, vel lacinia enim metus eu nunc.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis.

Cras sed ante. Phasellus in massa. Curabitur dolor eros, gravida et, hendrerit ac, cursus non, massa. Aliquam lorem. In hac habitasse platea dictumst. Cras eu mauris. Quisque lacus. Donec ipsum. Nullam vitae sem at nunc pharetra ultricies. Vivamus elit eros, ullamcorper a, adipiscing sit amet, porttitor ut, nibh. Maecenas adipiscing mollis massa. Nunc ut dui eget nulla venenatis ali-

quet. Sed luctus posuere justo. Cras vehicula varius turpis. Vivamus eros metus, tristique sit amet, molestie dignissim, malesuada et, urna.

Cras dictum. Maecenas ut turpis. In vitae erat ac orci dignissim eleifend. Nunc quis justo. Sed vel ipsum in purus tincidunt pharetra. Sed pulvinar, felis id consectetuer malesuada, enim nisl mattis elit, a facilisis tortor nibh quis leo. Sed augue lacus, pretium vitae, molestie eget, rhoncus quis, elit. Donec in augue. Fusce orci wisi, ornare id, mollis vel, lacinia vel, massa.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Morbi commodo, ipsum sed pharetra gravida, orci magna rhoncus neque, id pulvinar odio lorem non turpis. Nullam sit amet enim. Suspendisse id velit vitae ligula volutpat condimentum. Aliquam erat volutpat. Sed quis velit. Nulla facilisi. Nulla libero. Vivamus pharetra posuere sapien. Nam consectetuer. Sed aliquam, nunc eget euismod ullamcorper, lectus nunc ullamcorper orci, fermentum bibendum enim nibh eget ipsum. Donec porttitor ligula eu dolor. Maecenas vitae nulla consequat libero cursus venenatis. Nam magna enim, accumsan eu, blandit sed, blandit a, eros.

Quisque facilisis erat a dui. Nam malesuada ornare dolor. Cras gravida, diam sit amet rhoncus ornare, erat elit consectetuer erat, id egestas pede nibh eget odio. Proin tincidunt, velit vel porta elementum, magna diam molestie sapien, non aliquet massa pede eu diam. Aliquam iaculis. Fusce et ipsum et nulla tristique facilisis. Donec eget sem sit amet ligula viverra gravida. Etiam vehicula urna vel turpis. Suspendisse sagittis ante a urna. Morbi a est quis orci consequat rutrum. Nullam egestas feugiat felis. Integer adipiscing semper ligula. Nunc molestie, nisl sit amet cursus convallis, sapien lectus pretium metus, vitae pretium enim wisi id lectus. Donec vestibulum. Etiam vel nibh. Nulla facilisi. Mauris pharetra. Donec augue. Fusce ultrices, neque id dignissim ultrices, tellus mauris dic-

tum elit, vel lacinia enim metus eu nunc.

# References


[1] Baesens, B., Mues, C., De Backer, M., Vanthienen, J., & Setiono, R. (2005). *Building Intelligent Credit Scoring Systems Using Decision Tables*, (pp. 131–137). Springer Netherlands: Dordrecht.

[2] Bellazzi, R. & Zupan, B. (2008). Predictive data mining in clinical medicine: current issues and guidelines. *international journal of medical informatics*.

[3] Bratko, I. (1997). *Machine Learning: Between Accuracy and Interpretability*, (pp. 163–177). Springer Vienna: Vienna.

[4] Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). Classification and regression trees.

[5] Clausen, J. (1999). Branch and bound algorithms - principles and examples.

[6] Elaine Angelino, Nicholas Larus-Stone, D. A. M. S. C. R. (2017). Learning certifiably optimal rule lists. *KDD*.

[7] Freitas, A. A. (2014). Comprehensible classification models. *SIGKDD Explorations*.

[8] Garofalakis, M., Hyun, D., Rastogi, R., & Shim, K. (2000). Efficient algorithms for constructing decision trees with constraints. (pp. 335–339).

[9] Jeff Larson, Surya Mattu, L. K. & Angwin, J. (2016). How we analyzed the compas recidivism algorithm. *ProPublica*.

[10] Kolesar, P. J. (1967). A branch and bound algorithm for the knapsack problem.

[11] Lavrač, N. (1999). Data mining in medicine: Selected techniques and applications.

[12] Letham, B., Rudin, C., McCormick, T. H., & Madigan, D. (2015). Interpretable classifiers using rules and Bayesian analysis: Building a better stroke prediction model. *Annals of Applied Statistics*, 9(3), 1350–1371.

[13] Li, H. & Yamanishi, K. (2002). Text classification using esc-based stochastic decision lists. *Information Processing and Management*, 38(3), 343 – 361.

[14] Little, J. D., Murty, K. G., Sweeney, D. W., & Karel, C. (1963). An algorithm for the traveling salesman problem. *Operations research*, 11(6), 972–989.

[15] Martens, D., Vanthienen, J., Verbeke, W., & Baesens, B. (2011). Performance of classification models from a user perspective. *Decision Support Systems*, 51(4), 782 – 793. Recent Advances in Data, Text, and Media Mining and Information Issues in Supply Chain and in Service System Design.

[16] Moret, B. M. E. (1982). Decision trees and diagrams. *ACM Comput. Surv.*, 14(4), 593–623.

[17] Olivier Chapelle, Vikas Sindhwani, S. S. K. (2006). Branch and bound for semi-supervised support vector machines.

[18] Quinlan, J. (1993). C4.5: Programs for machine learning.

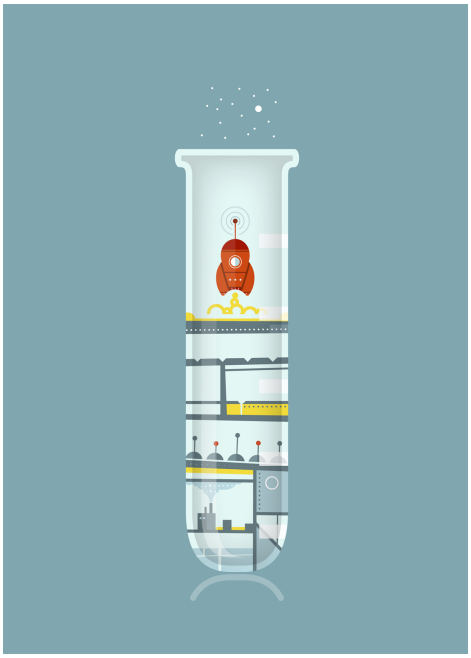
[19] Quinlan, J. R. (1999). *Some Elements of Machine Learning*, (pp. 15–18). Springer Berlin Heidelberg: Berlin, Heidelberg.

[20] Rivest, R. L. (1987). Learning decision lists. *Machine Learnning*, 2(3), 229–246.

[21] Ruping, S. (2006). Learning interpretable models.

[22] Yang, H., Rudin, C., & Seltzer, M. (2016). Scalable Bayesian rule lists. *Preprint at arXiv:1602.08610*.

[23] Zhang, Y., Laber, E. B., Tsiatis, A., & Davidian, M. (2015). Using decision lists to construct interpretable and parsimonious treatment regimes.



This thesis was typeset using LaTeX, originally developed by Leslie Lamport and based on Donald Knuth's TeX. The body text is set in 11 point Egenolff-Berner Garamond, a revival of Claude Garamont's humanist typeface. The above illustration, *Science Experiment 02*, was created by Ben Schlitter and released under CC BY-NC-ND 3.0. A template that can be used to format a PhD dissertation with this look & feel has been released under the permissive AGPL license, and can be found online at github.com/suchow/Dissertate or from its lead author, Jordan Suchow, at suchow@post.harvard.edu.