FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

NEURAL RULE ENSEMBLES:

ENCODING FEATURE INTERACTIONS INTO NEURAL NETWORKS

By

GITESH DAWER

A Dissertation submitted to the
Department of Mathematics
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

2018

Gitesh Dawer defended this dissertation on June 20, 2018.
The members of the supervisory committee were:

Adrian Barbu
Professor Directing Thesis

Kyle Gallivan
Co-Professor Directing Thesis

Gordon Erlebacher
University Representative

Giray Okten
Committee Member

Mark Sussman
Committee Member

Dedicated to my Parents & Brother

ACKNOWLEDGMENTS

I would like to express my sincere appreciation and thanks to my advisor, Dr. Adrian Barbu, for his endless patience, support and encouragement. I owe my knowledge of the field to him and feel truly blessed to have experienced working under his guidance.

I am also grateful to my co-advisor, Dr. Kyle Gallivan and the committee member, Dr. Giray Okten for their gracious cooperation and support throughout the study.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Following is a short list of abbreviations used throughout this document.

- **RF**: Random Forests
- **NRE**: Neural Rule Ensembles
- **GB**: Gradient Boosted Trees
- **XGB**: Extreme Gradient Boosting
- **ANN**: Artificial Neural Networks
- **RET**: Relevant Ensemble of Trees
- **L1**: Linear Model with L_1 penalty
- **FSA**: Feature Selection with Annealing
- **RSA**: Rule Selection with Annealing
- **EL**: Linear Model with Elastic Net regularization

ABSTRACT

Artificial Neural Networks form the basis of very powerful learning methods. It has been observed that a naive application of fully connected neural networks often leads to overfitting. In an attempt to circumvent this issue, a prior knowledge pertaining to feature interactions can be encoded into these networks. This defines a task-specific structure on an underlying representation and helps in reducing the number of learnable parameters. Convolutional Neural Network is such an adaptation of artificial neural networks for image datasets which exploits the spatial relationship among the features and explicitly encodes the translational equivariance. Similarly, Recurrent Neural Networks are designed to exploit the temporal relationship inherent in sequential data. However, for tabular datasets, any prior structure on feature relationships is not apparent. In this work, we use decision trees to capture such feature interactions for this kind of datasets and define a mapping to encode extracted relationships into a neural network. This addresses the initialization related concerns of fully connected neural networks and enables learning of compact representations compared to state of the art tree-based approaches. Empirical evaluations and simulation studies show the superiority of such an approach over fully connected neural networks and tree-based approaches.

CHAPTER 1

INTRODUCTION

Machine learning deals with the ability of computer systems to learn and perform important tasks by generalizing from examples and thereby, circumventing the need for an explicit programming. One of the formal definitions, as given in Mitchell [1997], defines it in a following way,

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E "

Most of the problems in machine learning can be classified into one of these three categories: Supervised, Unsupervised and Reinforcement learning. In this work, much of our focus would be concentrated on the supervised learning tasks wherein an explicit supervision in the form of ground-truth output is available for every training input. The goal underlying these tasks is to use those supervisory examples to learn a function that maps inputs to their desired outputs and correctly generalizes to new and unseen data.

This data driven approach to learning a function has three main components: Representation, Optimization and an Objective function. Representation provides a formal language for the model that the computer systems can handle and manipulate. Any such representation defines a space of functions that can possibly be learned, usually referred to as the hypothesis space. An objective function assesses and quantifies the performance of any representable function in the hypothesis space. An optimization strategy enables traversing the hypothesis space in the search of a better performing function.

1.1 Problem Statement

We are mainly interested in binary classification problems, where we are given training examples $\{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{R}, i = 1, ..., N\}$ and we need to find a hypothesis or prediction function $f : \mathbb{R}^p \to \mathbb{R}$ parameterized by a parameter vector $\boldsymbol{\beta}$, such that the sign of $f(\mathbf{x}_i)$ agrees with y_i as much as possible. For example, for linear models, the prediction function is $f(\mathbf{x}) = \boldsymbol{\beta}^T\mathbf{x}$ and $\boldsymbol{\beta} \in \mathbb{R}^p$.

The agreement between $f(\mathbf{x})$ and y on the training examples is measured by a loss function.

$$L(\boldsymbol{\beta}) = \sum_{i=1}^{N} \ell(f(\mathbf{x}_i), y_i) + s(\boldsymbol{\beta})$$

that should be minimized, where $s(\boldsymbol{\beta})$ is a penalty such as the shrinkage $s(\boldsymbol{\beta}) = \rho\|\boldsymbol{\beta}\|^2$ which helps with generalization.

The loss $\ell(u, y)$ depends on the problem. For regression, it could be the square loss $\ell(u, y) = (u - y)^2$. For binary classification (when $y \in \{-1, 1\}$), it could be the logistic loss $\ell(u, y) = \log(1 + \exp(-uy))$, the hinge loss $\ell(u, y) = \max(1 - uy, 0)$, or other loss functions.

1.1.1 Motivation

Let us assume that all the examples come from the same distribution, that is,

$$y_i = f(x_i) + \epsilon_i$$

where the noise ϵ_i satisfies $\mathbb{E}(\epsilon_i) = 0$ and $\mathrm{Var}(\epsilon_i) = \sigma^2$.
Let $\hat{f}$ be an estimate of f obtained upon minimizing the loss function on the training set.

$$\hat{f} = \operatorname{argmin} L(f, y)$$

Since the estimate depends on the choice of a training set, $\hat{f}$ is a random variable function. We can quantify the mean squared error (MSE) on the test set by computing the following expectation with respect to possible training sets.

$$\begin{aligned}
\text{Test MSE} &= \mathbb{E}\Big((y - \hat{f}(x))^2\Big) \\
&= \mathbb{E}\Big((\epsilon + f(x) - \hat{f}(x))^2\Big) \\
&= \mathbb{E}(\epsilon^2) + \mathbb{E}\Big((f(x) - \hat{f}(x))^2\Big) \\
&= \sigma^2 + \Big(\mathbb{E}\Big(f(x) - \hat{f}(x)\Big)\Big)^2 + \mathrm{Var}\Big(f(x) - \hat{f}(x)\Big) \\
&= \sigma^2 + \Big(\mathrm{Bias}\Big(\hat{f}(x)\Big)\Big)^2 + \mathrm{Var}\Big(\hat{f}(x)\Big)
\end{aligned}$$

The first term in the error σ^2 is related to the noise inherent in the data which is beyond our control. The bias term explains underfitting, meaning that on an average, $\hat{f}$ does not have enough capacity to capture the link f between the features $\mathbf{x}$ and the target y. The last term is closely related to overfitting where the prediction $\hat{f}$ performs very well on examples in the training set but doesn't generalize very well to other examples and varies a lot with the choice of training set.

In the context of neural networks, we have a Universal Approximation Theorem [Hornik, 1991]. It states that a single hidden layer neural network with a linear output unit can, in principle, approximate any continuous function arbitrarily well, given enough hidden units. This implies that the error due to bias is unconcerning as long as there are sufficient number of hidden units. On the other hand, it has been observed that, these shallow networks are highly susceptible to overfitting and thus, exhibits high variance.

We attempt to tackle this recurring theme of overfitting by encoding prior knowledge into the networks which would eliminate the need to discover that same knowledge during the training. We believe that, in addition to having the regularizing effect, this would translate into the need for comparatively less number of learnable parameters. Since we focus primarily on tabular datasets, we will use decision trees to capture that prior information in the form of feature interactions.

1.2 Outline

The dissertation is organized in a following fashion.

In chapter 2, we present a method for obtaining compact tree ensembles based on loss minimization that involves generating a large pool of trees followed by selecting a desired number of trees while simultaneously updating their leaf weights using the FSA algorithm [Barbu et al., 2017]. We provide a mathematical derivation for the group criterion employed in conjunction with FSA for selecting relevant trees. In order to speed up the model selection process, we discuss a modification in FSA algorithm that obtains a whole range of models corresponding to different sparsity levels in just a single run.

In chapter 3, we motivate the need to work with rule based ensembles since selecting a tree as a whole could still bring in redundant and irrelevant branches. We propose a natural generalization of the FSA algorithm to tree-induced rules called Rule Selection with Annealing (RSA). We show that the selection metric used in RSA, at any iteration, is proportional to the square of cosine similarity of a rule with the gradient boosted direction up to that iteration and is a measure of the square of total weighted margin of that rule.

In chapter 4, we overcome the inherent difficulty of tree-induced rules in approximating oblique decision boundaries by mapping them into a new form of conjunctive rules called Neural Rules. This neural transformation enables training of the activated region of a rule which, as a result, allows for tighter representations of the prediction function. We study several characteristics and the initialization related concerns underlying this proposed definition of a neural rule.

In chapter 5, we present a novel method for encoding feature interactions captured by a single decision tree into neural networks called Neural Rule Ensembles (NRE). We discuss some training aspects of the algorithm and perform empirical evaluations on 19 binary classification datasets from Penn Machine Learning Benchmark (PMLB) [Olson et al., 2017]. To access the statistical significance of the results, we use two statistical tests: Wilcoxon signed-ranks test and the sign test and individually compare NRE with Random Forests (RF), Gradient Boosted Trees (GB) and Artificial Neural Networks (ANN).

CHAPTER 2

RELEVANT ENSEMBLE OF TREES

Tree ensembles are flexible predictive models that can capture relevant variables and to some extent their interactions in a compact and interpretable manner. Most algorithms for obtaining tree ensembles are based on the versions of boosting or Random Forest. Previous work showed that boosting algorithms exhibit a cyclic behavior of selecting the same tree again and again due to the way the loss is optimized. At the same time, Random Forest is not based on loss optimization and obtains a more complex and less interpretable model. In this chapter we present a novel method for obtaining compact tree ensembles by growing a large pool of trees in parallel with many independent boosting threads and then selecting a small subset and updating their leaf weights by loss optimization. We allow for the trees in the initial pool to have different depths which further helps with generalization. Experiments on real datasets show that the obtained model has usually a smaller loss than boosting, which is also reflected in a lower misclassification error on the test set.

2.1 Introduction

In this work we are interested in finding parsimonious tree ensembles that minimize a loss function constrained to contain a small number of trees (tree sparsity) of any depth in a prescribed range. As the space of decision trees is combinatorially complex, we are limited to suboptimal methods for loss minimization with tree sparsity constraints.

One such method is provided by boosting, which adds one tree at each iteration to minimize a loss function in a greedy fashion. Different versions of boosting are aimed at minimizing different loss functions, but Gradient Boost [Friedman, 2001] is a generic boosting algorithm that can be used for any differentiable loss function.

However, boosting algorithms have shown some difficulties in loss minimization when many boosting iterations are used. In such cases, a cyclic behavior has been observed [Rudin et al., 2004] where the same or a very similar weak learner is selected again and again at later boosting iterations.

One possible explanation for this behavior when the learners are decision trees comes from the correlation between the newly introduced tree and the already existing ones, which is reflected in the need to change their leaf weights for loss minimization. Since the leaf weights for the already existing trees are not updated, the only way to change them is to add another similar tree that makes up for the weight changes.

In this work, we propose to find the trees and update their leaf weights simultaneously. Instead of using a forward-selection type of approach that is taken by boosting, where the "best" tree is added conditional on the already selected trees, we will use a stepwise approach that obtains a large pool of trees by boosting, then simultaneously selects a small number of them and updates their weights using the recently introduced Feature Selection with Annealing (FSA) [Barbu et al., 2017] algorithm.

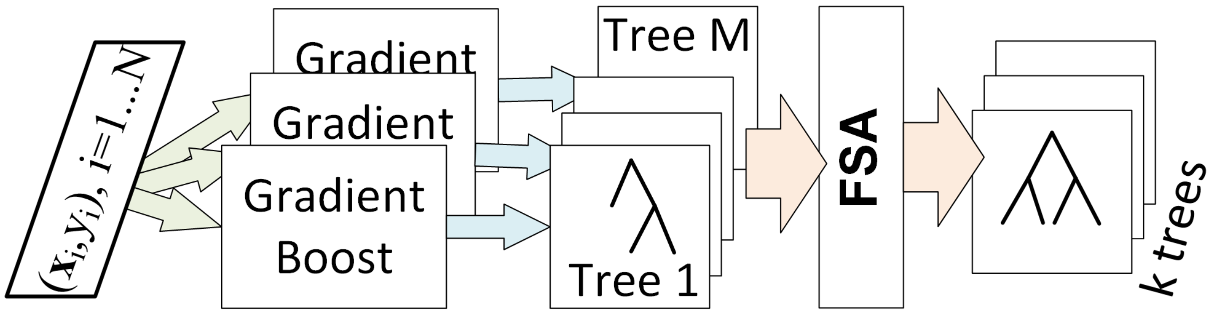


Figure 2.1: Diagram of the Relevant Ensemble of Trees algorithm.

The contributions of the work are the following.

1. It presents a novel way to obtain compact tree ensembles by generating a larger pool by GradientBoost and selecting a small number of trees by FSA, which also updates their leaf weights by loss minimization. A L_2 regularization term is added to the loss function for improved generalization. This is in contrast to boosting where L_2 regularization is hard to use and is often expressed in terms of a learning rate.

2. It introduces a method to obtain more diverse trees in the initial pool by initializing many GradientBoost threads with random vectors instead of a constant bias. Even though these trees cannot be used as they are for prediction, they can be used after their leaf weights have been updated by loss minimization. In addition, we prescribe different tree depths for the different threads, since our proposed formulation can naturally handle trees of different depths.

3. It also presents a modification of FSA that obtains a large number of models with different sparsity levels in the same run, with minimal extra computation.

2.1.1 Related Work

Trees based ensembles are regarded as one of the best off-the-shelf procedures for both classification and regression tasks. Applied naively, such ensembles often require a large of number of trees for modern big data sets. In addition, the complexity of each tree further increases with the size of the dataset. This jeopardizes their usability in large scale practical problems where memory is limited.

There has been a large amount of work on different versions of boosting, which can be used for generating tree ensembles. Different versions of boosting minimize different loss functions, starting from Adaboost [Freund and Schapire, 1997] for the exponential loss, Logitboost [Friedman et al., 2000] for the logistic loss, and Gradientboost [Friedman, 2001] for any differentiable loss. Other examples include Floatboost [Li and Zhang, 2004] and Robust Logitboost [Li, 2010].

To facilitate enhanced interpretability and overcome memory based limitations, the problem of obtaining compact tree ensembles has received much attention in the recent years. An improved interpretability was aimed in Friedman and Popescu [2008] by selecting optimal rule subsets from tree-ensembles. The classical cost-complexity pruning of individual trees was extended in Geurts [2000] to combined pruning of ensembles. The tree-ensemble based model was reformulated in Joly et al. [2012] as a linear model in terms of node indicator functions and a L1-norm regularization based approach (LASSO) was used to select a minimal subset of these indicator functions. All of these works focused on the simultaneous pruning of individual trees and large ensembles.

Another line of work focuses on lossless compression of tree ensembles. In Painsky and Rosset [2016] a probabilistic model was used for the tree ensemble and was combined with a clustering algorithm to find a minimal set of models that provides a perfect reconstruction of the original ensemble. Methods in Geurts [2000], Joly et al. [2012], Painsky and Rosset [2016] were developed for ensembles based on bagging or Random Forests only and exploit the fact that each of the individual trees is independent and identically distributed random entity for a given training data set. However, our method uses several threads of randomly initialized boosted ensembles. And it is well known that the trees generated with boosting are more diverse and much less complex compared to trees from bagging or Random Forests based models.

Boosting was also used in Reyzin [2011] to first train an entire tree ensemble. For prediction on a new example, the original tree ensemble was subsampled using a distribution induced by the tree weights. Since sampling of hypotheses needs to be done for every test example, besides being slow, one needs to store the entire original ensemble. The methods in Painsky and Rosset [2016] and Reyzin [2011] aim only to provide an accurate description of the original trees using fewer models while our goal is to even improve upon the prediction accuracy of the original ensemble.

Recently, there has been some work on updating the leaf weights of already trained trees. The leaf weights of an already trained Random Forest [Breiman, 2001] have been updated using an Artificial Prediction Market in Barbu and Lay [2012], which performs maximum likelihood learning in some cases [Barbu and Lay, 2012]. However, the Artificial Prediction Market did not obtain a compact tree ensemble by selecting a smaller set of trees, and was limited in the types of loss functions that can be optimized.

In Ren et al. [2015] the leaf weights of an already trained Random Forest were updated to minimize a loss similar to equation (2.1). The trees leaves are also pruned by merging adjacent leaves according to a L_2 significance criterion. This achieved a similar goal with our work in obtaining diverse trees of different depths, however no effort was done to obtain a compact set of trees that is smaller than the original set.

2.2 Relevant Ensemble of Trees (RET)

We will work on regression and binary classification problems, where we are given training examples $\{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{R}, i = 1, ..., N\}$ and we need to find a prediction function $f_{\boldsymbol{\beta}} : \mathbb{R}^p \to \mathbb{R}$ parameterized by a parameter vector $\boldsymbol{\beta}$, such that $f_{\boldsymbol{\beta}}(\mathbf{x}_i)$ agrees with y_i as much as possible. For example, for linear models, the prediction function is $f_{\boldsymbol{\beta}}(\mathbf{x}) = \boldsymbol{\beta}^T\mathbf{x}$ and $\boldsymbol{\beta} \in \mathbb{R}^p$.
The agreement between $f_{\boldsymbol{\beta}}(\mathbf{x})$ and y on the training examples is measured by a loss function.

$$L(\boldsymbol{\beta}) = \sum_{i=1}^{N} \ell(f_{\boldsymbol{\beta}}(\mathbf{x}_i), y_i) + s(\boldsymbol{\beta}) \tag{2.1}$$

that should be minimized, where $s(\boldsymbol{\beta})$ is a penalty such as the shrinkage $s(\boldsymbol{\beta}) = \rho\|\boldsymbol{\beta}\|^2$ which helps with generalization.

The loss $\ell(u,y)$ depends on the problem. For regression, it could be the square loss $\ell(u,y) = (u-y)^2$. For binary classification (when $y \in \{-1,1\}$), it could be the logistic loss $\ell(u,y) = \log(1+\exp(-uy))$, the hinge loss $\ell(u,y) = \max(1-uy,0)$, or other loss functions.

2.2.1 Representation as Tree Ensemble

One needs a formal language to represent the prediction function $f_{\boldsymbol{\beta}}(\mathbf{x})$ in a computer. The set of all functions that could be obtained by the chosen prediction function representation is called the *hypothesis space*. Our hypothesis space is the space of all tree ensembles, described below.

Decision tree representation. The basic building block of our representation is a decision tree. A decision tree is a function $T : \mathbb{R}^p \to \mathbb{R}$ associated with a tree representation that partitions the input domain $\mathbb{R}^p$ into a number of disjoint regions. Each region corresponds to a tree leaf and is characterized by the unique sequence of decisions that were made at the tree nodes to reach that leaf starting from the root. We will assume that each decision is based on one variable $x_j, j \in \{1,...,p\}$ of the input vector $\mathbf{x} \in \mathbb{R}^p$ and a threshold τ_j, but this assumption is only related to the algorithm we use for building the trees and can be easily relaxed to other algorithms.

Following Ren et al. [2015], we decompose the decision tree into a leaf weight vector $\boldsymbol{\beta} = (\beta_1,...,\beta_l)^T, \beta_i \in \mathbb{R}$ and an index function $\mathbf{i}_T(\mathbf{x}) : \mathbb{R}^p \to \{0,1\}^l$, which is a column vector of all zeros except a single 1 at the index of the leaf reached by the observation $\mathbf{x} \in \mathbb{R}^p$. Here $l = 2^d$ is the maximum number of leaf nodes of a tree of maximum depth d. Thus, we can write the decision tree as a dot product $T(\mathbf{x}) = \boldsymbol{\beta}^T \mathbf{i}_T(\mathbf{x})$. We will overload the notation by calling a decision tree as both the function $T(\mathbf{x}) : \mathbb{R}^p \to \mathbb{R}$ and its decomposition $T = (\boldsymbol{\beta}, \mathbf{i}_T(\mathbf{x}))$.

Tree ensembles. We will work with prediction functions that are the sum of a number of decision trees k, which are represented by their leaf weight vectors $\boldsymbol{\beta}_j$ and index functions $\mathbf{i}_j(\mathbf{x})$. Then, for a given feature vector $\mathbf{x} \in \mathbb{R}^p$, the prediction function has the form

$$f(\mathbf{x}) = \sum_{j=1}^{k} \boldsymbol{\beta}_j^T \mathbf{i}_j(\mathbf{x}).$$

Such models are usually called *tree ensembles*.

2.2.2 Training

We are interested in parsimonious tree ensembles that minimize the loss function (2.1) constrained to contain at most k trees (tree sparsity) of any depth in a prescribed set S. The parameters k and S are problem specific and can be obtained by cross-validation or using an information criterion such as AIC/BIC.

To select better trees and concurrently update their leaf weights to work best together, we will generate a large number of trees M by boosting and then use the Feature Selection with Annealing (FSA) algorithm [Barbu et al., 2017] to minimize the loss (2.1) constrained to be based on k trees, as illustrated in Figure 2.1. This way in the end, we will have k trees selected and their leaf weights optimized to minimize the loss function (2.1).

The steps of generating trees by boosting and then selecting them by FSA will be described in the next three subsections.

2.2.3 Generating a pool of trees by GradientBoost

The initial pool of M trees will be constructed by GradientBoost. In our applications, M will be usually on the order of $M = 3000$. We will explore three approaches to generating these M trees:

- **Single Chain Single Depth (SCSD)** generation where the M trees of the same depth are obtained in a single GradientBoost run with M boosting iterations.
- **Multi Chain Single Depth (MCSD)** generation where the M trees of the same depth are obtained (in parallel) by m separate GradientBoost runs with different random initializations, each run with M/m boosting iterations.
- **Multi Chain Multi Depth (MCMD)** generation where the M trees of $|S|$ different possible maximum depths are obtained (in parallel) by m separate GradientBoost runs with different initializations, each run with M/m boosting iterations.

Conventionally, GradientBoost is initialized with a constant prediction function, obtained by minimizing the loss function in terms of this constant value, which is referred to as the *bias*.

Random initialization. Since we are primarily interested in the structure of decision trees, in the multi chain approach we initialize the GradientBoost procedure with a random prediction vector. This allows us to invoke several versions of GradientBoost with different random initializations, resulting into a richer and more diverse collection of trees related to the underlying problem.

Observe that the trees obtained from random initializations cannot be directly used for prediction without modifying their leaf weights, since they were started from random values (predictions) on the training examples, which make no sense on the test set. Through this random initialization we are only interested in obtaining the structure or mathematically speaking, index function $i(\mathbf{x})$ for each tree since our algorithm will update the leaf weights by loss minimization.

Random prediction vector. It refers to a vector containing the initial score or bias for every training instance. This base score corresponding to every training instance is independently and identically sampled from $\mathcal{N}(0,1)$. This randomization is performed only once and from there on, conventional GradientBoost is employed to obtain an ensemble of trees. Since bias is different for different training examples, one does not know what value of base score to use on a new, unseen test instance. As such, there is a need to establish a constant global bias for every example, which further necessitates the modification of leaf weights for all the trees in an ensemble. Our proposed algorithm "FSA on leaves" remedies all these issues while reducing tree redundancy and greediness to some extent. We consider it to be a novel extension to FSA algorithm, which was primarily developed for linear models.

Why use a randomized base score? Let us talk about the first boosting iteration only. For a given loss function, the base score of a training instance completely determines the gradient and Hessian entries corresponding to that instance. For example, using binomial deviance loss, positive instances (true label of 1) with base scores $\ln(1/9)$ (< 0) and $\ln(9)$ (> 0) have negative gradients as 0.9 and 0.1 respectively. For the first boosting iteration, in the case of a constant base score for all the examples, all the positive instances have one and the same value for the gradient and such is the case for all the negative instances. However, for a randomized base score, all the instances, positive or negative, have different gradient and Hessian values. Tree construction at every iteration depends only on the gradient and Hessian statistics. Instances with larger negative gradients have higher contribution in selecting best splitting variable and the best split point for a node. Roughly speaking, thus randomized base score establishes a ranking among the examples in the order of their relevance towards tree building, unlike in constant base score where all the positive instances are treated alike and so are all the negative instances. Different randomizations correspond to different rankings of the training examples, which results in a different tree at the first iteration. Since successive trees build on the previous trees, we expect the effects of randomization to percolate further deep down the iteration process. The motivation behind all this is to diversify the pool of trees as much as possible and have **FSA on leaves** pick up the best ones for the task. Observe that this is different from the compressed Random Forest Joly et al. [2012] where all the trees are independently generated. In contrast, our trees in each chain are dependent on each other because they are obtained by boosting.

Besides being faster, we will see empirically in Section 2.3.3 that the multi chain tree generation results in lower loss values and a more robust algorithm compared to the single chain tree generation.

Multiple depths. In the case of Multi Chain Multi Depth (MCMD), we generate trees of different maximum depths given by the set S, obtaining a tree ensemble where the trees have a large range of depths. This is important because different features have different levels of interactions, which can be best captured with the correct tree depth. A smaller tree depth might not be sufficient for fitting the interaction properly, while a larger depth might be overfitting. Empirical evidences in Section 2.3.3 validates the superiority of MCMD over the other tree generation approaches.

2.2.4 Overview of the FSA algorithm

We will use the Feature Selection with Annealing (FSA) algorithm [Barbu et al., 2017] to select the most relevant trees and update their weights.

FSA is an algorithm for simultaneous feature selection and model learning on the selected features, aimed at minimizing a differentiable loss function $L(\boldsymbol{\beta})$ with constraints on the number k of non-zero coefficients:

$$\boldsymbol{\beta} = \underset{|\boldsymbol{\beta}|_0 \leq k}{\operatorname{argmin}} \, L(\boldsymbol{\beta}) \tag{2.2}$$

The FSA algorithm proceeds in a backward elimination manner, starting with $\boldsymbol{\beta} = 0$ and alternating one stochastic gradient update step with a step that removes some variables according to a *deterministic* schedule that specifies the number M_e of variables that should be left after iteration e. The FSA method is summarized in Algorithm 1.

The schedule $M_e, e = 1, ..., N^{iter}$ is quickly decreasing as

$$M_e = k + (p-k)\max(0, \frac{N^{iter} - 2e}{2e\mu + N^{iter}}) \tag{2.3}$$

specified by a parameter μ, where p is the dimension of the feature vectors $\mathbf{x}_i \in \mathbb{R}^p$.

Because most of the variables are removed in the first few iterations, the algorithm becomes increasingly fast after each iteration and can be hundreds of times faster than boosting when

Algorithm 1 Feature Selection with Annealing (FSA)

```
**Input:** Normalized training set $\{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{R}\}_{i=1}^N$
**Output:** Trained model $f_{\boldsymbol{\beta}}(\mathbf{x}) = \boldsymbol{\beta}^T\mathbf{x}$ with parameter vector $\boldsymbol{\beta}$.
1: Initialize $\boldsymbol{\beta} = 0$.
2: for e = 1 to $N^{iter}$ do
3:     Update $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \eta\frac{\partial L(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}$
4:     Keep only the $M_e$ variables corresponding to the highest $|\beta_j|$.
5: end for
```

selecting thousands of variables. It is also thousands of times faster than the L1, SCAD and MCP penalized methods [Donoho et al., 2006, Fan and Li, 2001, Zhang, 2010].

Besides being fast, another exciting fact about FSA is that it enjoys *theoretical guarantees of consistency and convergence* [Barbu et al., 2017]. If the learning rate is sufficiently small and the variable removal schedule is sufficiently slow, the FSA algorithm will find all the k^* true variables ($k^* \leq k$) with high probability.

2.2.5 Tree selection and leaf weight update by FSA

In this section, we describe how to select a small number of trees k from the pool of M trees generated as described in Section 2.2.3. Following the notation of Section 2.2.1, let these M trees be $(\boldsymbol{\beta}_j, \mathbf{i}_j), j = \overline{1, M}$, where $\boldsymbol{\beta}_j$ is the leaf weight vector and $\mathbf{i}_j(\mathbf{x})$ is the index function of tree j.

We present two ways to select k trees from the M generated trees:

In **FSA on trees**, the M tree responses $T_j(\mathbf{x}) = \boldsymbol{\beta}_j^T\mathbf{i}_j(\mathbf{x})$ are used by FSA as an M-dimensional feature vector to select k features, thus the k corresponding trees. If the sparse vector obtained by FSA training is $\mathbf{w} = (w_1, ..., w_M) \in \mathbb{R}^M$, then the obtained prediction function is

$$f_{\mathbf{w}}(\mathbf{x}) = \sum_{j=1}^{M} w_j \boldsymbol{\beta}_j^T \mathbf{i}_j(\mathbf{x})$$

which has only at most k non-zero coefficients w_j, thus depends on at most k trees.

Observe that FSA on trees can only be used for single chain tree generation since it doesn't update the leaf weights and the multi chain tree generation obtains randomly initialized trees that are not useful for prediction unless their leaf weights are updated.

In **FSA on leaves**, we modify the FSA algorithm to update the tree leaf weights and select trees using a group criterion. For that, all the tree leaf weights are collected in a $l \times M$ matrix

$B = (\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, ..., \boldsymbol{\beta}_M)$ of parameters for the prediction function

$$f_B(\mathbf{x}) = \sum_{j=1}^{M} \boldsymbol{\beta}_j^T \mathbf{i}_j(\mathbf{x}),$$

where l is the maximum number of leaf nodes of the trees from our pool.

In order to select relevant trees using FSA on leaves, we would need a group criterion to quantify the performance of a tree. Let us now investigate one such possible metric. We use subscript i to denote the i^{th} training example, subscript k to refer to the k^{th} leaf of a tree and the subscript j for the j^{th} tree. Let $L(\mathbf{y}, \mathbf{f})$ denote any differentiable loss function with $y_i \in \{-1, 1\}\ \forall i$, we define

$$g_i = \frac{\partial L}{\partial f_i}\bigg|_{f_i=0} \qquad h_i = \frac{\partial^2 L}{\partial f_i^2}\bigg|_{f_i=0}$$

For any decision tree $T(\mathbf{x}) : \mathbb{R}^p \to \mathbb{R}$ with a decomposition $T = (\boldsymbol{\beta}, \mathbf{i}(\mathbf{x}))$, the second order approximation of the loss function is given as

$$\begin{aligned} L(\mathbf{y}, \mathbf{T}) &\approx L(\mathbf{y}, \mathbf{0}) + \frac{1}{N}\sum_{i=1}^{N} g_i T(\mathbf{x}_i) + \frac{1}{2N}\sum_{i=1}^{N} h_i T^2(\mathbf{x}_i) \\ &= L(\mathbf{y}, \mathbf{0}) + \frac{1}{N}\sum_{i=1}^{N} g_i \beta_{\mathbf{i}(\mathbf{x}_i)} + \frac{1}{2N}\sum_{i=1}^{N} h_i \beta^2_{\mathbf{i}(\mathbf{x}_i)} \end{aligned} \tag{2.4}$$

where the leaf weight vector $\boldsymbol{\beta} = (\beta_1, ..., \beta_l)^T, \beta_i \in \mathbb{R}$. Since each training example falls into exactly one leaf, we can simplify (2.4) to

$$L(\mathbf{y}, \mathbf{T}) \approx L(\mathbf{y}, \mathbf{0}) + \frac{1}{N}\sum_{k=1}^{l}\left(\beta_k \sum_{\{i:\ \mathbf{i}(\mathbf{x}_i)=k\}} g_i + \frac{1}{2}\beta_k^2 \sum_{\{i:\ \mathbf{i}(\mathbf{x}_i)=k\}} h_i\right) \tag{2.5}$$

Setting partial derivative $\dfrac{\partial L}{\partial \beta_k} = 0\ \ \forall k$ to get the optimal leaf weight vector $\boldsymbol{\beta}$,

$$\beta_k = -\frac{\sum_{\{i:\ \mathbf{i}(\mathbf{x}_i)=k\}} g_i}{\sum_{\{i:\ \mathbf{i}(\mathbf{x}_i)=k\}} h_i}$$

Substituting the optimal $\boldsymbol{\beta}$ into (2.5),

$$L(\mathbf{y}, \mathbf{T}) \approx L(\mathbf{y}, \mathbf{0}) - \frac{1}{2N}\sum_{k=1}^{l}\left(\beta_k^2 \sum_{\{i:\ \mathbf{i}(\mathbf{x}_i)=k\}} h_i\right) \tag{2.6}$$

Since for binary classification, the loss function $L(\mathbf{y}, \mathbf{f})$ can be expressed as a function of margin, that is, $\hat{L}(\mathbf{y}\mathbf{f})$, we have

$$h_i = \frac{\partial^2 L}{\partial f_i^2}\bigg|_{f_i=0}$$

$$= y_i^2 \frac{\partial^2 \hat{L}}{\partial (yf)_i^2}\bigg|_{y_i f_i = 0} = \lambda$$

where λ is independent of the class membership of a training example. Let n_k denote the number of training examples indexing into k^{th} leaf of a tree, this simplifies equation (2.6) to

$$\begin{aligned} L(\mathbf{y}, \mathbf{T}) &\approx L(\mathbf{y}, \mathbf{0}) - \frac{1}{2N}\sum_{k=1}^{l} \beta_k^2 n_k \lambda \\ &= L(\mathbf{y}, \mathbf{0}) - \frac{\lambda}{2}\sum_{k=1}^{l} \frac{n_k}{N}\beta_k^2 \\ &= L(\mathbf{y}, \mathbf{0}) - \frac{\lambda}{2}\,\mathrm{E}(\beta^2) \end{aligned}$$

where E(.) is taken over the leaves of a tree. Since $L(\mathbf{y}, \mathbf{0})$ and λ are same for all the trees, one can employ $\mathrm{E}(\beta^2)$ to evaluate the relevance of a tree.

A variant of the FSA algorithm is run to minimize the loss function $L(B)$ from (2.1) with the matrix B taking the place of $\boldsymbol{\beta}$. For that, the criterion for selecting the variables in step 4 of FSA is changed to a group criterion $\mathrm{E}(\beta^2)$ which ensures that only at most k column vectors $\boldsymbol{\beta}_j$ will be non-zero in the end. In case of a uniform distribution over leaves of a tree, the group criterion for tree j reduces to $\frac{\|\boldsymbol{\beta}_j\|_2}{l_j}$, where l_j is the number of leaves of tree j which seems intuitive since it corrects for the bias towards selecting trees with more leaves.

2.2.6 FSA with Multiple Sparsity Levels

For computational efficiency, we modify the FSA algorithm to obtain parameters for multiple sparsity levels by memorizing the obtained parameters at each iteration and performing a deep copy of those parameters to optimize them in a separate routine when the sparsity level for any respective model is reached. The details are given in Algorithm 2 below.

The annealing parameter μ in the interval $[10, 20]$ works well in practice and FSA parameters are chosen in accordance with it. The annealing schedule M_e utilizes the smallest sparsity level k_q in its definition given in (2.3).

In this work we will use a modified version of this algorithm that has a matrix of leaf weights for $\boldsymbol{\beta}$ and a group criterion for selection, as described in Section 2.2.5.

Algorithm 2 FSA with multiple sparsity levels

Input: Normalized training examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, learning rate η, sparsity levels $\{k_1, ..., k_q\}, k_1 > k_2 > ... > k_q$, number of iterations N^{iter}, annealing schedule $M_e, e = 1, ..., N^{iter}$.

Output: Trained classifier parameter vectors $\boldsymbol{\beta}_1, ..., \boldsymbol{\beta}_q$ with $\|\boldsymbol{\beta}_i\|_0 = k_i, i = 1, ..., q$.

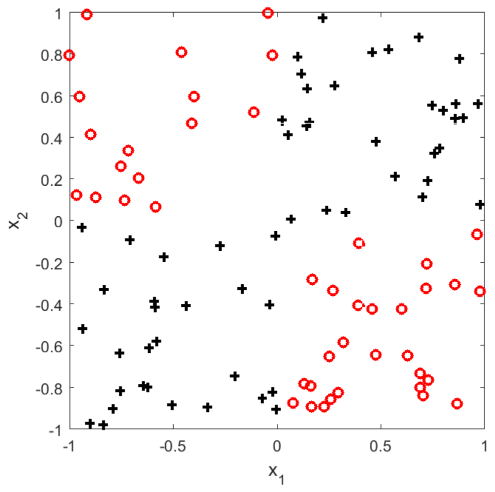
```
1:  Compute $E = \{e_1, ...e_q\}$ with $e_i = \max(\{e : M_e \geq k_i\})$.
2:  Initialize $\boldsymbol{\beta} = 0$.
3:  for $e = 1$ to $N^{iter}$ do
4:      Update $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \eta \frac{\partial L(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}$
5:      Keep only the $M_e$ variables with highest $|\beta_j|$ and renumber them $1, ..., M_e$.
6:      if $e \in E$ then
7:          for $i \in \{1, ..., q\}$ such that $e_i = e$ do
8:              Set $\boldsymbol{\beta}_i = \boldsymbol{\beta}$
9:              Keep only the $k_i$ variables $j$ with highest $|\beta_{ij}|$ and renumber them $1, ..., k_i$.
10:             Update $\boldsymbol{\beta}_i \leftarrow \boldsymbol{\beta}_i - \eta \frac{\partial L(\boldsymbol{\beta}_i)}{\partial \boldsymbol{\beta}}$ for $N^{iter}$ times
11:         end for
12:     end if
13: end for
```

2.3 Experiments

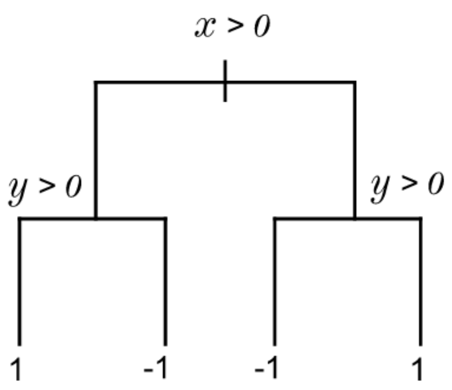
We will perform three types of experiments. Our first experiment would ascertain the effectiveness of FSA in selecting relevant trees from a larger pool. The second set of experiments will compare the loss minimization obtained by GradientBoost with the ones obtained using RET with the three tree generation approaches, for the same model complexity. A smaller loss means a smaller training error which for the same model complexity would in general reflect in a lower test error unless the model overfits. The third set of experiments will compare the test error of the proposed method with GradientBoost (GB), XGBoost (XGB) and some linear methods such as L_1 penalized logistic regression and Elastic Net on six real datasets.

2.3.1 Simulation

In this section, we use a classical example, the XOR data, to support our claim on the effectiveness of RET in obtaining compact tree ensembles. Since XOR, a non-linearly separable dataset, can perfectly be represented using a single decision tree of depth 2 as illustrated in Figure 2.2, right, we restrict the maximum tree depth to 2 for this experiment. Both the training and the test set consist of 100 randomly sampled data points.



(a)



(b)

Figure 2.2: An instance of the XOR dataset (a) and the decision tree that was used to generate the XOR data (b).

The setup of the experiment is as follows:

- **RET :** We use Single Chain Single Depth (SCSD) pool generation approach as described in 2.2.3 to obtain an initial pool of $M = 400$ trees of depth 2. We then invoke *FSA on leaves* as detailed in 2.2.5 to select just one tree.
- **GradientBoost :** We run GradientBoost procedure for the least number of iterations required to achieve about the same Test AUC as one given by RET.

Table 2.1: Simulated Experiment on XOR dataset, averaged over 100 runs.

XOR dataset, $N = 100, p = 2, d = 2$			
Method	# trees k	Train AUC	Test AUC
GradientBoost	26	**0.991**	0.967
RET	1	0.985	**0.968**

In Table 2.1 are shown the area under an ROC curve for both training and the test sets, averaged over 100 independent runs. It takes GradientBoost, on an average, about 26 trees to match the generalization performance of a single tree given by RET. From Table 2.1, it is also apparent that the tree picked up by RET closely resembles the actual tree representation that was used for generating the XOR data. This provides an empirical justification in support of the proposed Relevant Ensemble of Trees method using FSA for obtaining parsimonious tree ensembles.

Table 2.2: Dataset summary.

Dataset	obs train/test	features	classes
gisette [Guyon et al., 2004]	6,000/1,000	5,000	2
miniboone	130,065	50	2
madelon [Guyon et al., 2004]	2,000/600	500	2
wilt	4,889	6	2
abalone	4,177	8	regression
online news	39,797	61	regression

Table 2.3: RET/FSA parameter settings used in the experiments.

μ	η	N^{iter}	k_{max}	S(depth set)
10	10^{-3}	300	600	$\{2,..,7\}$
10	10^{-3}	300	600	$\{2,..,7\}$
10	10^{-3}	150	100	$\{9,..,14\}$
10	10^{-3}	150	100	$\{2,..,7\}$
10	10^{-5}	150	100	$\{2,..,7\}$
10	10^{-5}	150	100	$\{2,..,7\}$

2.3.2 Datasets

The experiments will be performed on six public datasets from the UCI machine learning repository [Lichman, 2013], of which 2 have been part of the 2003 Feature Selection Challenge [Guyon et al., 2004]. Since the FS challenge submission website is down, for the FS challenge datasets we have used the validation sets as test sets. The datasets are summarized in Table 2.3. From the datasets, we have removed any observations that had missing data.

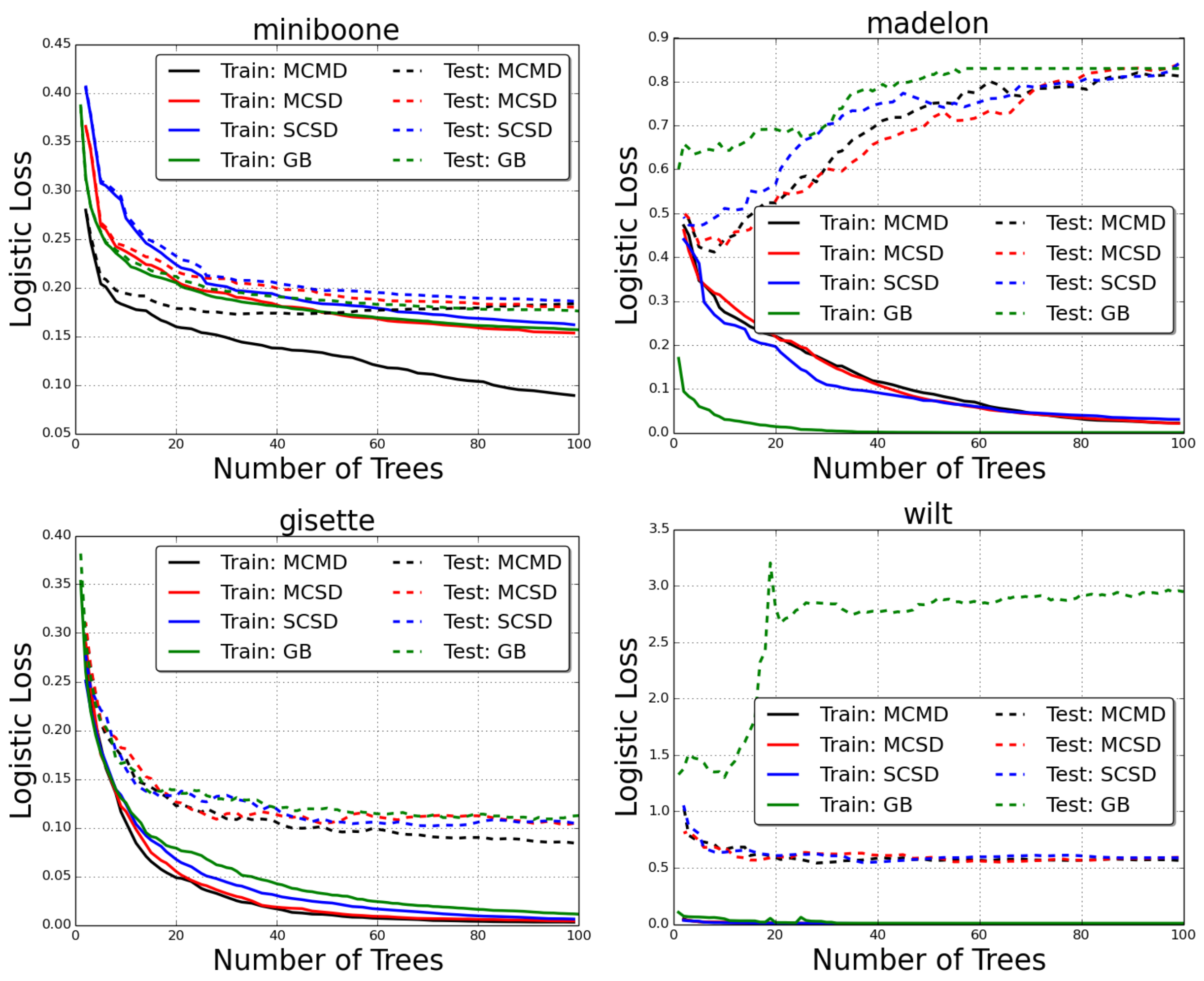


Figure 2.3: Binary Classification Datasets: Comparison of loss minimization for selecting different numbers of trees from pools generated using three different approaches and GradeintBoost.

2.3.3 Loss Minimization Evaluation

In this section, we will compare the loss minimization capabilities of the RET with GradientBoost in selecting the same number of trees. We will use pools of trees generated as follows:

- **Single Chain Single Depth (SCSD)** pool generation: We use 3000 boosting iterations to obtain M=3000 trees of one single depth, which is obtained using the parameter tuning method described in detail in Section 2.3.4.
- **Multi Chain Single Depth (MCSD)** pool generation: We use 100 boosting iterations for each of the 30 chains to obtain M=3000 trees of the same depth as SCSD.

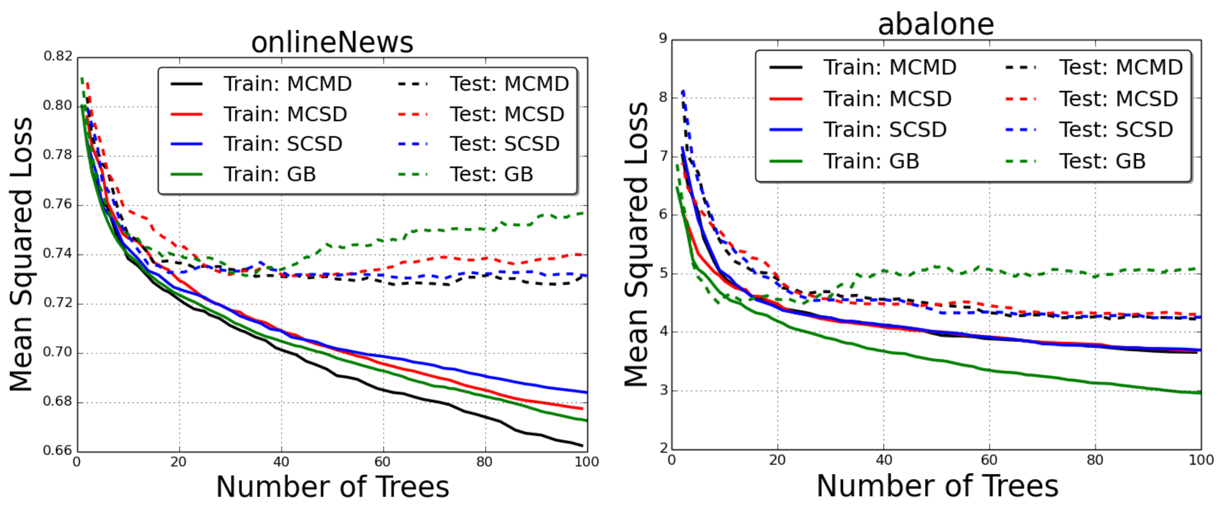


Figure 2.4: Regression Datasets: Comparison of loss minimization for selecting different numbers of trees from pools generated using three different approaches and GradeintBoost.

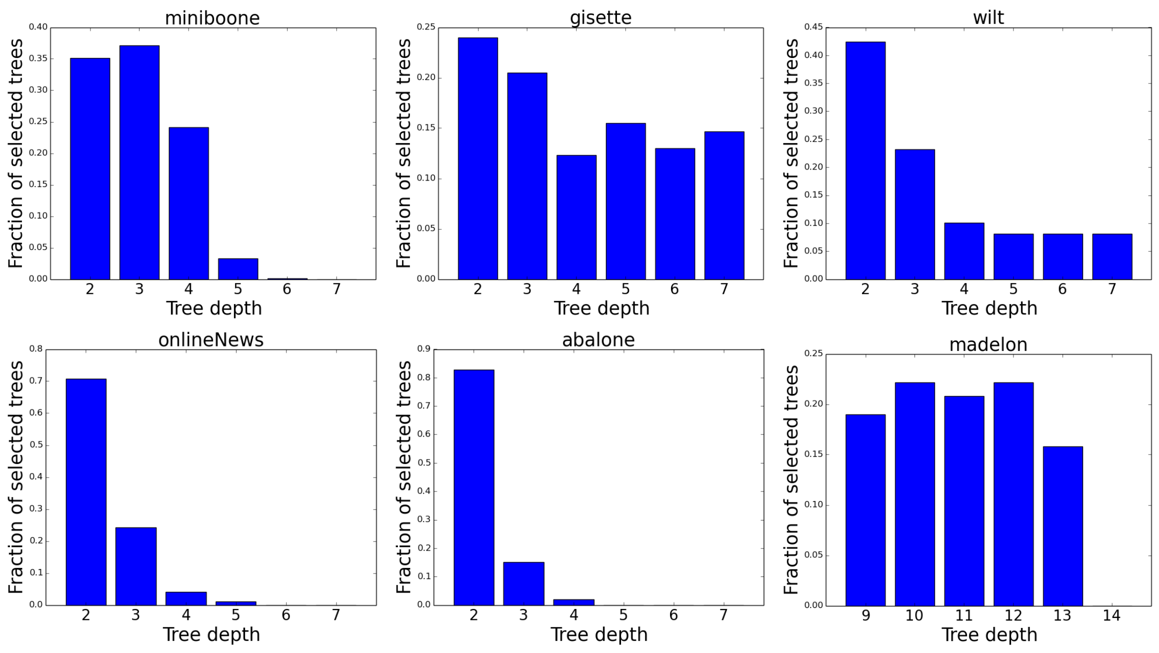


Figure 2.5: Distribution of the fraction of trees selected by FSA with tree depth.

- **Multi Chain Multi Depth (MCMD)** pool generation: There are 30 chains, 5 chains for each depth in the depth set S, and 100 boosting iterations in each chain, for a total of M=3000 trees.

Other relevant parameters are given in Table 2.3. For a fair comparison, the L_2 regularization parameter for RET was set to $\rho = 0$.

The loss functions on the six datasets for the Gradient Boost and RET with the three pool generation methods (SCSD, MCSD, MCMD) are shown in Figure 2.4. We see that the RET with the three pool generation methods generally overfit less than GradientBoost. Indeed, for all the six datasets, given a fixed number of trees, the RET methods usually have a lower test loss compared to GradientBoost resulting in a more compact or relevant ensemble of trees. Among different pool generation approaches, MCMD dominates on all but one (madelon) datasets as evidenced by test loss curves in Figure 2.4, making it a natural choice for the desired task.

Table 2.4: Real data results, averaged over 20 runs. Standard deviations are shown in parentheses.

Classification datasets, test errors in %.

Dataset	FSA	L1	EL	GB	RF	XGB	RET
gisette	2.00 (0.14)	3.80 (0.00)	3.77 (0.05)	4.78 (0.56)	3.11 (0.23)	3.88 (0.39)	**1.95 (0.19)**
madelon	50.00 (0.00)	49.58 (0.43)	49.37 (1.70)	23.3 (0.84)	28.5 (2.15)	21.9 (1.13)	**17.6 (1.42)**
miniboone	12.22 (1.23)	9.84 (0.16)	9.83 (0.19)	6.70 (0.13)	6.29 (0.08)	6.37 (0.06)	**5.64 (0.25)**
wilt	37.4 (0.00)	37.4 (0.00)	37.4 (0.00)	26.09 (5.60)	22.06 (0.32)	18.79 (1.60)	**17.14 (0.95)**
Regression datasets, test R^2 in %.							
abalone	49.79 (2.98)	50.26 (3.10)	50.27 (3.12)	50.14 (5.28)	57.29 (4.28)	51.98 (4.83)	**57.73(4.49)**
news	11.94 (0.68)	10.61 (3.80)	9.49 (5.17)	15.24 (0.24)	15.26 (1.44)	14.43 (0.84)	**16.94 (0.80)**

Histograms. For Multi Chain Multi Depth (MCMD) tree generation, the initial pool contains equal number of trees from each chain and equal number of trees for each depth. FSA is adept at automatically selecting trees corresponding to different tree depths using the group criterion specified in Section 2.2.5. Figure 2.5 shows the proportion of trees selected by FSA from every depth type for six real datasets.

We see that most of the tree depths are used in most datasets and that the distribution of tree depths differs from dataset to dataset.

2.3.4 Test Error Evaluation

In this section, we compare RET with GradientBoost (GB), XGBoost (XGB), Random Forest (RF), some generalized linear models with the L_1 [Bunea et al., 2007, Donoho et al., 2006, Knight and Fu, 2000, Zhao and Yu, 2007] and Elastic Net [Zou and Hastie, 2005] penalties, as well as linear

FSA to see if we really need tree-based models. For a more accurate comparison, we will use the logistic loss $\ell(u, y) = \log(1 + \exp(-uy))$ for linear FSA and RET.

Parameter tuning. The parameters for the methods being evaluated have been obtained by five-fold cross-validation on the training set as follows:

1. For each parameter combination, the cross-validated loss was computed as the average of the validation loss over the five cross-validation folds of the training set.
2. The parameter combination corresponding to the smallest cross-validated loss was selected and the final model was obtained for that parameter combination by training on the entire training set. This model was then used to obtain the predictions on the test set.

The parameters involved in parameter tuning are the following:

- The number of selected trees $k \in [1, k_{max}]$ for RET, Random Forest (RF) and for GradientBoost/XGBoost (as k boosting iterations). We used 50 sparsity levels k on an exponential grid between 1 and k_{max}.
- The tree depth d for GradientBoost. It has the range $d \in \{2, ..., 7\}$ for most datasets and $d \in \{9, ..., 14\}$ for madelon.
- The shrinkage parameter ρ for FSA, Elastic Net and RET, $\rho \in \{10^{-1}, 10^{-2}, ..., 10^{-4}, 10^{-5}\}$

Other parameters for FSA and RET were fixed to values given in Table 2.3. The test errors for the datasets without a test set are obtained using a random $80 - 20$ train-test split. All results are shown as averages over 20 independent runs.

In Table 2.4 are shown the test errors using the parameter tuning described above. We see that RET obtains the lowest test errors on all of the four binary classification tasks and the highest test R^2 on both regression datasets.

One downside of RET is that it has higher training times as shown in Table 2.5, which can partly be improved upon exploiting the parallelization of MCMD pool generation process. The FSA model building for the different parameter combinations on the different cross-validation folds can also be easily parallelized.

2.4 Conclusion

This chapter presents a novel method for obtaining a compact tree ensemble based on loss minimization that involves generating a large pool of trees followed by selecting a desired number

Table 2.5: Average training times in minutes.

Dataset	FSA	L1	EL	GB	XGB	RET
gisette	16.7	42	257	112	16.9	535
madelon	6.4	1.7	8.3	3.7	0.6	225
miniboone	4.6	101	439	248	107	857
wilt	0.02	0.29	1.4	0.67	0.69	6.1
abalone	0.07	0.09	0.45	0.13	0.09	7.2
online news	16.2	46	208	23	0.42	85

of trees while updating their leaf weights using the FSA algorithm. The initial pool of trees is obtained by Boosting, thus it is more compact and relevant than the trees obtained by Random Forest, leading to a more parsimonious tree ensemble. Experiments on six UCI datasets indicate that the proposed approach usually obtains a smaller test loss for the same model complexity than GradientBoost and smaller test errors on large datasets.

One of the novel ideas of this work is to generate the initial pool of trees using many parallel GradientBoost (GB) threads having random initializations and different tree depths. This approach obtains more diverse trees than a single GB thread or even compared to many GB threads trained on bootstrap samples. The leaf weights of these trees cannot be used for prediction and had to be retrained by loss minimization using the FSA algorithm. The usage of L_2 regularization in RET makes it more robust to overfitting compared to GradientBoost.

Another contribution is the modification in FSA algorithm that obtains a whole range of models corresponding to different sparsity levels in just a single run. The obtained sets of weights have the property that each contains the next one, thus forming a chain relative to the inclusion relation. This is an FSA equivalent to using warm restarts in the Elastic Net.

CHAPTER 3

RULE SELECTION WITH ANNEALING

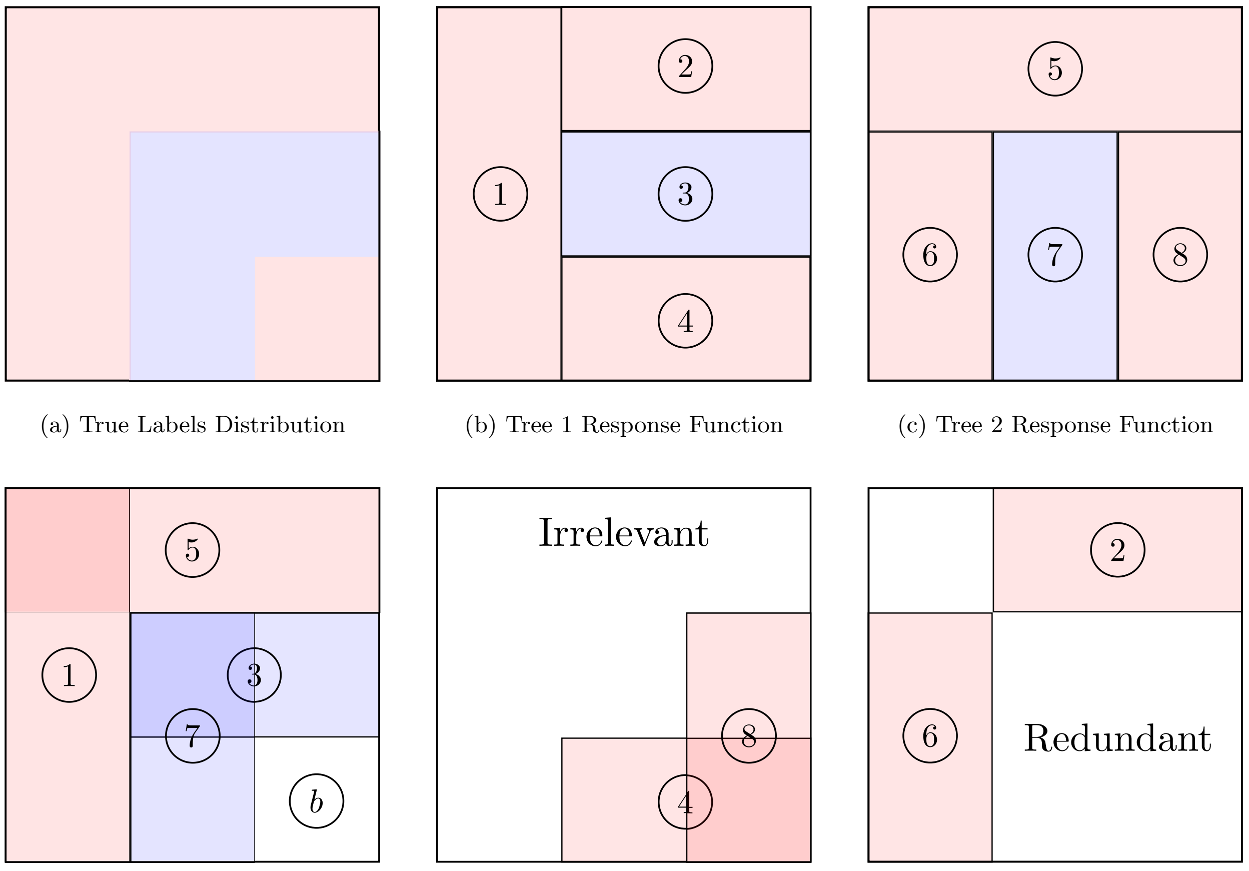
3.1 Motivation

In the last chapter, we proposed a new method for creating an ensemble of trees. One of the primary goals in the design of that approach was to reduce model redundancy and achieve model compactness by selecting only a subset of relevant trees from a larger pool. The relevance metric or the group criterion used to rank trees in the order of their relevance, involved all the branches of a tree. However, we realize that some of the branches of a tree could be completely redundant or irrelevant. A branch with a strong predictive power is defined as a *redundant* one if it can be safely removed without harming the generalization performance of the ensemble in any way. Similarly, an *irrelevant branch* is defined as one with either nil or poor predictive performance and whose presence could, if only, deteriorate the overall performance of an ensemble. We present such an observation through the following illustration.

Figure 3.1 (a) shows the distribution of positive (red) and negative (blue) labels in the feature space for a synthetic dataset and Figure 3.1 (b) and (c) represent response functions of two decision trees used to model this classification task. The branches labeled as 4 and 8 in Figure 3.1 (e) have absolutely zero discriminative power and are therefore irrelevant. On the other hand, despite having strong predictive ability and being pure, the branches with label 2 and 6 can be safely removed without hurting the generalization performance at all and are therefore redundant. We note that out of a combined total of eight branches, four are sufficient enough to correctly classify all the examples as is depicted in figure 3.1 (d) and one can safely get rid of the remaining four. This motivates us to take a plunge into the rule-based ensembles to achieve even more compact representations.

3.2 Relevant Ensemble of Rules

A conjunctive rule is a logical statement of the following form:
IF *conditions* THEN *value* ELSE 0



(a) True Labels Distribution (b) Tree 1 Response Function (c) Tree 2 Response Function

(d) Relevant Branches (e) Irrelevant Branches (f) Redundant Branches

Figure 3.1: An illustration to motivate the discussion for rule ensembles

where the *value* is either positive or negative depending on the class label and the *conditions* are a logical conjunction of many simple boolean tests. In other words, all such simple boolean expressions must evaluate to true in order to activate a rule. For an input $\mathbf{x}$ with real-valued continuous attributes $(x_1, x_2, ..., x_n)$, one can express the rule in the following mathematical form:

$$r(\mathbf{x}) = c \prod_{j=1}^{n} I(x_j \in s_j) \tag{3.1}$$

where $I(\cdot)$ is an indicator function, c is a non-zero activation value and s_j is a subset of all possible values for an attribute x_j.

3.2.1 Rule Generation

A decision tree can be regarded as a collection of conjunctive rules where each path from the root to a terminal node defines one such rule. Specifically, a regression tree with m terminal nodes can be represented as

$$T(\mathbf{x}) = \sum_{k=1}^{m} r_k(\mathbf{x}) \tag{3.2}$$

Non-zero values of $r_k(\mathbf{x})$ correspond to a hyper-rectangle in the input feature space. Each of these m hyper-rectangles are non-overlapping and collectively, define a m partitioning of the feature space.

In cases where all the features involved in a decision tree-induced rule are continuous variables, the rule in (3.1) can be reduced to a much simpler form. Let $\mathbf{x}_R$ be a set of features involved in a rule $r(\mathbf{x})$, one can now rewrite the expression for a conjunctive rule in (3.1) as follows.

$$r(\mathbf{x}) = c \prod_{x_j \in \mathbf{x}_R} \mathrm{H}(w_j x_j + a_j) \tag{3.3}$$

where $\mathrm{H}(\cdot)$ is a Heaviside step function with $\mathrm{H}(0) = 0$, c is the value at an associated terminal node, w_j is either $+1$ or -1 and a_j is the split threshold for feature x_j if $w_j = -1$, and is the negative of split threshold otherwise.

In order to create an initial pool consisting of large number of rules, we first invoke existing fast algorithms such as Random Forests and Gradient Boosted Trees to generate tree ensembles . In the subsequent step, each of the tree thus produced is decomposed into a set of conjunctive rules as prescribed above.

3.2.2 Rule Scaling

Let m denote the total number of rules present in the initial pool. The predictive model is then given by

$$f(\mathbf{x}) = w_0 + \sum_{k=1}^{m} w_k r_k(\mathbf{x})$$

The goal here is to train weights w_k's using loss optimization which is reminiscent of a linear classifier using $r_k(\mathbf{x})$'s as fixed input features. For an efficient training, LeCun et al. [1998] recommends transforming the input features in order to bring all of them on the same scale. Additionally, observe that, if we replace $r_k(\mathbf{x}) \leftrightarrow \alpha\, r_k(\mathbf{x})$ and $w_k \leftrightarrow \frac{1}{\alpha} w_k$, the product $w_k r_k(\mathbf{x})$ remains unchanged. In order to fix the scale of $r_k(\mathbf{x})$ and thereby w_k, we replace $r_k(\mathbf{x}) \leftrightarrow \frac{r_k(\mathbf{x})}{\|\mathbf{r_k}\|}$. There is still an ambiguity in the sign of $r_k(\mathbf{x})$ which however is not concerning since the sign of w_k adjusts itself accordingly. Since input features are binary variables, we do not perform mean centering so as to preserve the inherent sparseness structure. The updated prediction model now looks as follows.

$$f(\mathbf{x}) = w_0 + \sum_{k=1}^{m} w_k \, \frac{r_k(\mathbf{x})}{\|\mathbf{r_k}\|} \tag{3.4}$$

3.2.3 Rule Selection with FSA

We will use the Feature Selection with Annealing (FSA) algorithm proposed in Barbu et al. [2017] to simultaneously select the most relevant rules $r_k(\mathbf{x})$ and update the corresponding weights w_k of (3.4). The algorithm aims at minimizing a differentiable loss function $L(\mathbf{w})$ with constraints on the number p of non-zero coefficients:

$$\mathbf{w} = \operatorname*{argmin}_{|\mathbf{w}|_0 \le p} \sum_{i=1}^{N} L\left(y_i, w_0 + \sum_{k=1}^{m} w_k \, \frac{r_k(\mathbf{x}_i)}{\|\mathbf{r_k}\|} \right) \tag{3.5}$$

It proceeds in a backward elimination manner, starting with $\mathbf{w} = \mathbf{0}$ and alternates one epoch of stochastic gradient update with a step that removes some rules according to a *deterministic* schedule which specifies the number M_t of rules that should be left after iteration t. The method is summarized in Algorithm 3.

Algorithm 3 Rule Selection with FSA

```
Input: Normalized training set $\{(\{\frac{r_k(\mathbf{x}_i)}{\|\mathbf{r_k}\|}\}_{k=1}^m, y_i) \in \mathbb{R}^m \times \mathbb{R}\}_{i=1}^N$
Output: Trained model $f(\mathbf{x})$ with parameter vector $\mathbf{w}$.
1: Initialize $\mathbf{w} = \mathbf{0}$.
2: for t = 1 to $N^{iter}$ do
3:     Update $\mathbf{w} \leftarrow \mathbf{w} - \eta \, \nabla L(\mathbf{w})$
4:     Keep only the $M_t$ rules corresponding to the highest $|w_k|$.
5: end for
```

The schedule $M_t, t = 1, ..., N^{iter}$ is quickly decreasing as

$$M_t = p + (m - p) \max(0, \frac{N^{iter} - 2t}{2t\mu + N^{iter}}),$$

specified by a parameter μ, where m is the dimension of the feature vectors $\{\frac{r_k(\mathbf{x})}{\|\mathbf{r_k}\|}\}_{k=1}^m \in \mathbb{R}^m$.

3.3 Margin Maximizing Rules

In order to perform either an implicit or explicit selection of rules, we would need a metric to quantify their importance which can then be used to rank them in the order of their relevance. One such metric that can be employed is the hinge loss, also referred to as the maximum margin loss, where rules with the lower values of the loss function will have higher relevance. Mathematically, the expression for this loss function is given as

$$I\left(\mathbf{y}, \frac{\mathbf{r}_{\boldsymbol{k}}}{\|\mathbf{r}_{\boldsymbol{k}}\|}\right) = \sum_{i=1}^{N} \max\left(0, 1 - y_i \frac{r_k(\mathbf{x}_i)}{\|\mathbf{r}_{\boldsymbol{k}}\|}\right) \tag{3.6}$$

where $y_i \in \{-1, +1\} \, \forall \, i$. The quantity $y\frac{r_k(\mathbf{x})}{\|\mathbf{r}_{\boldsymbol{k}}\|}$, known as the *margin* or confidence in the prediction, is positive for the correct prediction and is negative in case of the wrong one. Notice that, in the absence of rule scaling $\|\mathbf{r}_{\boldsymbol{k}}\|$ as motivated in Section 3.2.2, the scale of $r_k(\mathbf{x})$ could artificially be chosen to make the confidence $y \, r_k(\mathbf{x})$ arbitrarily large which in that case would render the definition of *margin* useless.

Let n_k be the number of training examples that activate the rule r_k where n_k^+ of them belongs to the positive class and the remaining n_k^- comes from the negative class. Using (3.3), the euclidean norm of the rule can thus be computed as

$$\|\mathbf{r}_{\boldsymbol{k}}\|_2 = |c|\sqrt{n_k}$$

For any training input $\mathbf{x}$, the activation value of a rule $r_k(\mathbf{x})$ is either 0 or c. Consequently, the normalized value is either 0 or given by

$$\frac{r_k(\mathbf{x})}{\|\mathbf{r}_{\boldsymbol{k}}\|} = \frac{c}{|c|}\frac{1}{\sqrt{n_k}}$$

which implies that the *margin* is unconditionally bounded from above by 1 or more precisely,

$$-1 \leq y\frac{r_k(\mathbf{x})}{\|\mathbf{r}_{\boldsymbol{k}}\|} \leq 1$$

$$0 \leq 1 - y\frac{r_k(\mathbf{x})}{\|\mathbf{r}_{\boldsymbol{k}}\|} \leq 2$$

This allows us to simplify the expression for the hinge loss given in (3.6) to

$$\begin{aligned}
I\left(\mathbf{y}, \frac{\mathbf{r}_{\boldsymbol{k}}}{\|\mathbf{r}_{\boldsymbol{k}}\|}\right) &= \sum_{i=1}^{N}\left(1 - y_i \frac{r_k(\mathbf{x}_i)}{\|\mathbf{r}_{\boldsymbol{k}}\|}\right) \\
&= N - \sum_{i=1}^{N} y_i \frac{r_k(\mathbf{x}_i)}{\|\mathbf{r}_{\boldsymbol{k}}\|} \\
&= N - \sum_{\{i : r_k(\mathbf{x}_i) \neq 0\}} y_i \frac{r_k(\mathbf{x}_i)}{\|\mathbf{r}_{\boldsymbol{k}}\|}
\end{aligned}$$

Since both $r_k(\mathbf{x})$ and $-r_k(\mathbf{x})$ are potentially valid rules, the quantifying metric becomes

$$\underset{k}{\operatorname{argmin}}\, I\left(\mathbf{y}, \pm\frac{\mathbf{r}_{\boldsymbol{k}}}{\|\mathbf{r}_{\boldsymbol{k}}\|}\right) = \underset{k}{\operatorname{argmax}}\ m_k^2 \tag{3.7}$$

where

$$m_k = \sum_{\{i : r_k(\mathbf{x}_i) \neq 0\}} y_i \frac{r_k(\mathbf{x}_i)}{\|\mathbf{r}_{\boldsymbol{k}}\|} \tag{3.8}$$

This means that the rules can simply be sorted in the decreasing order of scores m_k^2 to obtain a ranking in the order of their relevance. In other words, higher value of m_k^2 corresponds to higher relevance. Let us further simply the expression given in (3.8) to

$$\begin{aligned}
m_k &= \frac{c}{|c|}\frac{(n_k^+ - n_k^-)}{\sqrt{n_k}} \\
m_k^2 &= \frac{(n_k^+ - n_k^-)^2}{n_k}
\end{aligned} \tag{3.9}$$

We now consider two special cases to facilitate a quick and simple discussion of the metric, m_k^2.

- **Case I :** $n_k^+ = n_k^-$
  A rule r_k whose support set contains an equal number of positive and negative training examples has zero predictive power by itself. And, the above definition of the measure m_k^2 is successfully able to encode this expectation by assigning the least possible value to such a rule, that is, $m_k^2 = 0$.

- **Case II :** Either $n_k^+ = 0$ or $n_k^- = 0$
  When all the training examples that activates a rule belong to either the positive class or the negative class, we categorize such a rule as being the pure one. For pure rules, the metric m_k^2 is equal to n_k and is therefore class agnostic. In other words, if a pure negative rule and a pure positive rule each has the equal number of activated training examples then they will be assigned the same measure of the metric m_k^2, which is in accordance with the expectation.

Additionally, $m_k^2 = n_k$ implies that the pure rules with more number of activated training examples will have higher value of m_k^2 and therefore more relevance as one would expect owing to their higher discriminative power.

The proposed definition of the metric m_k^2 (3.9) however has two main limitations. Foremost, it quantifies the performance of each rule when used independently. Some of the rules which might appear to have little predictive power by themselves, could however be quite relevant when employed in conjunction with the other rules. And, the second one is that the measure involves equal contribution of each training example. However, some of the examples could be much harder to train compared to the other ones. These issues motivate us to modify the metric by incorporating the weighted contribution of training examples in the definition and using larger weights for harder examples.

We use subscript i to denote the i^{th} training example, subscript k to refer to the k^{th} rule and the superscript t for the t^{th} iteration.

Lemma 3.3.1. *The metric w_k^2 used in Rule Selection with FSA Algorithm 3 measures the square of total weighted margin of the rule r_k, that is,*

$$w_k = \sum_{\{i: r_k(\mathbf{x}_i) \neq 0\}} \beta_i y_i \frac{r_k(\mathbf{x}_i)}{\|\mathbf{r}_{\boldsymbol{k}}\|} \tag{3.10}$$

where β_i is the weight of an i^{th} training example.

Proof. Let $f(\mathbf{x})$ be the prediction model as defined in equation (3.4) and $L(y, f)$ be any loss function employed in the optimization procedure for determining w_k's. Let us define,

$$g_i = -\left.\frac{\partial L}{\partial f}\right|_{f=f_i} \tag{3.11}$$

We know that, for a positive training example, the loss function L decreases as f increases. Similarly, for a negative training example, the loss function L increases as f increases. That is,

$$y_i = +1 \implies \left.\frac{\partial L}{\partial f}\right|_{f=f_i} < 0 \implies g_i > 0$$

$$y_i = -1 \implies \left.\frac{\partial L}{\partial f}\right|_{f=f_i} > 0 \implies g_i < 0$$

which implies that,

$$\text{sign}(y_i) = \text{sign}(g_i) \quad \forall i \tag{3.12}$$

Additionally, for a convex loss function in f, we have

$$\frac{\partial^2 L}{\partial f^2} \geq 0$$

which implies that the magnitude of g_i decreases as the margin, yf_i or confidence in the prediction of i^{th} training example increases because of the following inequality

$$y\frac{\partial L}{\partial f} \leq 0$$

This suggests that the magnitude of g_i could be used to quantify the hardness of i^{th} training example. Since g_i is changing at every iteration, one can use the moving average to define β_i^t after t iterations as follows:

$$\begin{aligned}\beta_i^t =& \frac{1}{t}\sum_{j=1}^{t} |g_i^j| \\ =& \frac{1}{t}\Big|\sum_{j=1}^{t} g_i^j\Big|\end{aligned} \tag{3.13}$$

Combining equations (3.12) and (3.13), we get

$$\beta_i^t y_i = \frac{1}{t}\sum_{j=1}^{t} g_i^j \tag{3.14}$$

Let us now investigate the update rule for the parameters of prediction model $f(\mathbf{x})$.

$$\begin{aligned} w_k^0 &= 0 \\ w_k^t &= w_k^{t-1} - \eta\frac{\partial L}{\partial w_k} \\ &= w_k^{t-1} - \eta\sum_{i=1}^{N}\frac{\partial f}{\partial w_k}\frac{\partial L}{\partial f}\bigg|_{f=f_i^{t-1}} \end{aligned}$$

Using equations (3.4), (3.11) and (3.14) to simplify,

$$\begin{aligned} w_k^t &= w_k^{t-1} + \eta\sum_{\{i:r_k(\mathbf{x}_i)\neq 0\}}\frac{r_k(\mathbf{x}_i)}{\|\mathbf{r}_{\boldsymbol{k}}\|}g_i^t \\ &= \eta\sum_{\{i:r_k(\mathbf{x}_i)\neq 0\}}\frac{r_k(\mathbf{x}_i)}{\|\mathbf{r}_{\boldsymbol{k}}\|}\sum_{j=1}^{t} g_i^j \\ &= \eta\, t\sum_{\{i:r_k(\mathbf{x}_i)\neq 0\}}\beta_i^t y_i\frac{r_k(\mathbf{x}_i)}{\|\mathbf{r}_{\boldsymbol{k}}\|} \end{aligned}$$

Since $\eta\, t$ is independent of i, we can absorb it into the definition of β_i to write

$$w_k^t = \sum_{\{i:r_k(\mathbf{x}_i)\neq 0\}} \beta_i^t y_i \frac{r_k(\mathbf{x}_i)}{\|\mathbf{r}_{\boldsymbol{k}}\|} \tag{3.15}$$

which is equal to total weighted margin of the rule r_k at t^{th} iteration. $\square$

At each iteration, Rule Selection with FSA procedure keeps only M_t rules corresponding to the highest squared value of total weighted margin at that iteration and drops the remaining ones. Since both r_k and $-r_k$ are valid contenders given that there is an ambiguity in the sign of r_k as mentioned in Section 3.2.2, we use w_k^2 instead of w_k to rank rules. Upon dropping some of the rules, it is probable that some of the training examples might become harder. The algorithm accounts for this change in the complexity by updating the weights β's of training examples after every iteration and thereby, making w_k^2, a dynamic and an adaptive metric for evaluating rules r_k's.

3.4 Maximizing Cosine Similarity

In this section, we present a new interpretation of the metric w_k^2 in the context of gradient boosting machine. We use subscript i to denote the i^{th} training example, subscript k to refer to the k^{th} rule and the superscript t for the t^{th} iteration.

Lemma 3.4.1. *The metric w_k^2, at any iteration t, is proportional to the square of cosine similarity of a rule r_k with the gradient boosted direction G^t up to that iteration, that is,*

$$w_k^t = \alpha^t \frac{< \mathbf{G}^t, \mathbf{r}_{\boldsymbol{k}} >}{\|\mathbf{G}^t\| \|\mathbf{r}_{\boldsymbol{k}}\|} \tag{3.16}$$

where α^t is a proportionality constant independent of k.

Proof. Let $f(\mathbf{x})$ be the prediction model as defined in equation (3.4) and $L(y, f)$ be any loss function employed in the optimization procedure. In gradient boosting, the goal is to minimize $L(y, f)$ with respect to f, where the parameters involved are the values of prediction function $f(\mathbf{x}_i)$ at each of the N training examples $\mathbf{x}_i$, collectively denoted by $\mathbf{f}$, that is,

$$\mathbf{f} = \{f(\mathbf{x}_1), f(\mathbf{x}_2), f(\mathbf{x}_3), \dots , f(\mathbf{x}_N)\}^T$$

and the optimization problem is given by

$$\mathbf{G} = \underset{\mathbf{f}}{\operatorname{argmin}}\, L(\mathbf{f}) \tag{3.17}$$

The gradient boosting solves (3.17) iteratively by taking a step in the steepest descent direction $\mathbf{g} \in \mathbb{R}^N$ defined in the space of functions, starting with an initial guess $\mathbf{G}^0 = \mathbf{0}$,

$$\mathbf{G}^t = \mathbf{G}^{t-1} + \eta\, \mathbf{g}^t \tag{3.18}$$

where the i^{th} component of the vector $\mathbf{g}^t$ is given as

$$g_i^t = -\frac{\partial L}{\partial f}\bigg|_{f=f_i^{t-1}}$$

Simplifying equation (3.18),

$$\mathbf{G}^t = \mathbf{G}^0 + \eta \sum_{j=1}^{t} \mathbf{g}^j = \eta \sum_{j=1}^{t} \mathbf{g}^j$$

Taking dot product with $\frac{\mathbf{r}_{\boldsymbol{k}}}{\|\mathbf{r}_{\boldsymbol{k}}\|}$ on both sides,

$$\begin{aligned}
< \mathbf{G}^t, \frac{\mathbf{r}_{\boldsymbol{k}}}{\|\mathbf{r}_{\boldsymbol{k}}\|} > &= \eta \sum_{j=1}^{t} < \mathbf{g}^j, \frac{\mathbf{r}_{\boldsymbol{k}}}{\|\mathbf{r}_{\boldsymbol{k}}\|} > \\
&= \eta \sum_{j=1}^{t} \sum_{\{i: r_k(\mathbf{x}_i) \neq 0\}} g_i^j \frac{r_k(\mathbf{x}_i)}{\|\mathbf{r}_{\boldsymbol{k}}\|} \\
&= \eta \sum_{\{i: r_k(\mathbf{x}_i) \neq 0\}} \frac{r_k(\mathbf{x}_i)}{\|\mathbf{r}_{\boldsymbol{k}}\|} \sum_{j=1}^{t} g_i^j
\end{aligned}$$

Next, using equations (3.14) and (3.15),

$$\begin{aligned}
< \mathbf{G}^t, \frac{\mathbf{r}_{\boldsymbol{k}}}{\|\mathbf{r}_{\boldsymbol{k}}\|} > &= \eta\, t \sum_{\{i: r_k(\mathbf{x}_i) \neq 0\}} \beta_i^t y_i \frac{r_k(\mathbf{x}_i)}{\|\mathbf{r}_{\boldsymbol{k}}\|} \\
&= \eta\, t\, w_k^t
\end{aligned}$$

Dividing both sides by $\left\|\mathbf{G}^t\right\|$, we obtain

$$\begin{aligned}
w_k^t &= \frac{\left\|\mathbf{G}^t\right\|}{\eta\, t} \, \frac{< \mathbf{G}^t, \mathbf{r}_{\boldsymbol{k}} >}{\left\|\mathbf{G}^t\right\| \|\mathbf{r}_{\boldsymbol{k}}\|} \\
&= \alpha^t \, \frac{< \mathbf{G}^t, \mathbf{r}_{\boldsymbol{k}} >}{\left\|\mathbf{G}^t\right\| \|\mathbf{r}_{\boldsymbol{k}}\|}
\end{aligned}$$

where the proportionality constant $\alpha^t = \frac{\left\|\mathbf{G}^t\right\|}{\eta\, t}$ is independent of k. □

Under this interpretation, $\mathbf{G}^t$ represents a gradient boosted direction in the functional space after t iterations. At each iteration, rules are evaluated and ranked based on the magnitude of cosine of their angle with that optimal direction.

3.5 Limitation

Conventional decision tree uses only one feature per node for recursive binary partitioning which corresponds to the fact that all the slices or partitions in the feature space are now either perpendicular or (inclusive) parallel to the feature axes. Since Relevant Ensemble of Rules procedure does not modify the decision nodes of the rules generated by these trees, it inherits their limitations as well. In other words, such models will have difficulties in approximating oblique decision boundaries in the feature space.

Let us consider a linearly separable dataset as shown in figure 3.2 (a) where the decision boundary is inclined at an angle of 45° with either of the feature axis. Owing to the aforementioned rigidity, it can be seen from figures 3.2 (b) and (c) that the only way to improve performance is to keep adding more rules to the ensemble. The inability to evolve shapes of the rules seems highly restrictive and challenges the goal of achieving compact representations. Hence, we would not want to restrict the activated region of a rule $r(\mathbf{x})$ to just a hyper-rectangle. This motivates the definition of a neural rule with a trainable support.

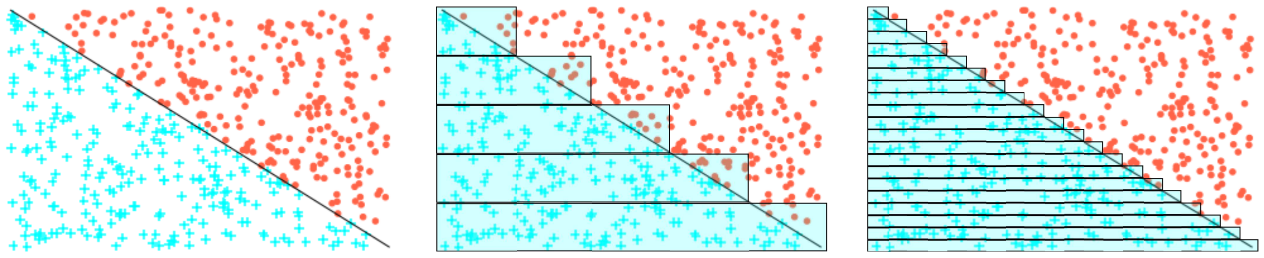

(a) Linearly Separable Dataset (b) Ensemble containing 5 Rules (c) Ensemble containing 20 Rules

Figure 3.2: Limitations of Relevant Ensemble of Rules in approximating linearly separable datasets

CHAPTER 4

NEURAL ENCODING OF A TREE-INDUCED RULE

In this chapter, we introduce a new form of the conjunctive rules called Neural Rules. In order to see how a decision tree based rule inspires the design of a neural rule, let us revisit the expression of a conjunctive rule specifically for a decision tree based rule given in (3.3).

$$r(\mathbf{x}) = c \prod_{x_j \in \mathbf{x}_R} \mathrm{H}(w_j x_j + a_j)$$

where $\mathbf{x}_R$ is a set of features involved in a rule $r(\mathbf{x})$, $\mathrm{H}(\cdot)$ is a Heaviside step function with $\mathrm{H}(0) = 0$, c is the value at an associated terminal node, w_j is either $+1$ or -1 and a_j is the split threshold for feature x_j if $w_j = -1$, and is the negative of split threshold otherwise.

Rules extracted from a decision tree involve only one feature for every node. In a neural rule, we modify (3.3) to now connect each node of a given rule with all the features used in the corresponding decision tree. Let $\mathbf{x}_T$ be a vector of all features used in the decision tree $T(\mathbf{x})$ without repetitions. With this modification, we can now have oblique decision boundaries in the feature subspace spanned by $\mathbf{x}_T$. The updated expression of the rule looks as follows.

$$r(\mathbf{x}) = c \prod_{\{j: x_j \in \mathbf{x}_R\}} \mathrm{H}(\mathbf{w}_j^T \mathbf{x}_T + a_j) \tag{4.1}$$

Next, we observe that a Heaviside step function with $\mathrm{H}(0) = 0$ is invariant to the ReLU transformation of an input. We also note that the product of several Heaviside step functions can be represented using a single Heaviside step function and the minimum pooling operation. With ReLU denoted using $\sigma(x) = \max(0, x)$, the above identities can be expressed mathematically as shown below.

$$\mathrm{H}(x) = \mathrm{H}(\sigma(x)) \tag{4.2}$$

$$\prod_k \mathrm{H}(x_k) = \mathrm{H}(\min_k x_k) \tag{4.3}$$

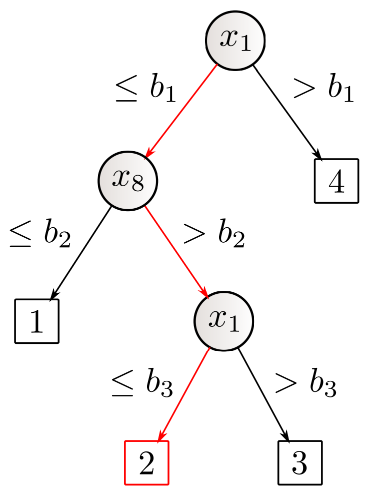
Using these identities, the rule in (4.1) becomes,

$$r(\mathbf{x}) = c \prod_{\{j:x_j \in \mathbf{x}_R\}} \mathrm{H}(\sigma(\mathbf{w}_j^T \mathbf{x}_T + a_j)) = c \cdot \mathrm{H}(\min_{\{j:x_j \in \mathbf{x}_R\}} \sigma(\mathbf{w}_j^T \mathbf{x}_T + a_j)) \tag{4.4}$$

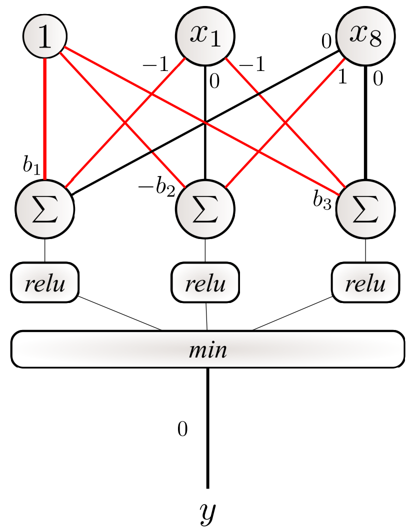
Since the derivative of a step function is zero, the gradients of all the learnable parameters will stay zero unless some modification is made. In order to be able to jointly train all the weights and splitting thresholds of all the nodes in a rule, we switch the Heaviside step function in equation (4.4) with an identity function. This gives us the neural rule in its final form as

$$r(\mathbf{x}) = c \cdot \min_{\{j:x_j \in \mathbf{x}_R\}} \sigma(\mathbf{w}_j^T \mathbf{x}_T + a_j) \tag{4.5}$$

4.1 Initialization



(a) A Decision Tree



(b) A Neural Rule

Figure 4.1: Mapping a Tree-induced Rule, with terminal label 2, into a Neural Rule

We now discuss an initialization of a neural rule corresponding to any given rule obtained from a decision tree. First, we make a list of all the features involved in that tree. All those features along with a bias unit form the input layer of a neural rule. The number of hidden units in the first hidden layer of a neural rule equals the number of decision nodes of a tree-induced rule, with one-to-one correspondence between them. The connection weight between the input feature and the hidden unit is 0 if the corresponding decision node of that hidden unit does not involve the feature under consideration. It is -1 if the corresponding decision node utilizes that feature and traverses its left child along the rule path and is $+1$ otherwise.

The magnitude of bias for every hidden unit is given by the absolute value of splitting threshold utilized in the corresponding decision node. The sign of bias for a hidden unit is positive if the corresponding decision node traverses its left child along the rule path. Otherwise it is negative.

We use figure 4.1 to illustrate one such mapping. Figure 4.1 (a) shows an arbitrary decision tree with four rules. Figure 4.1 (b) shows a neural rule corresponding to the rule with terminal label 2 (red colored branch) of the decision tree. All the bold lines in a neural rule represents trainable parameters with their initial values displayed alongside in figure 4.1 (b). Red bold lines in a neural rule carry non-zero initial weights and have their counterparts in the decision tree whereas black bold lines represents new connections with zero initialization.

4.2 Characteristics

4.2.1 Trainable Support

Each $\mathbf{w}^T\mathbf{x}_T + a = 0$ in the equation (4.5) represents a hyperplane in the feature subspace with the corresponding upper half space given by $\sigma(\mathbf{w}^T\mathbf{x}_T + a) > 0$. Application of min operation evaluates the intersection of these upper half spaces and thus, defines an activated region of a rule. For an input $\mathbf{x}$ that lies on the upper half space of the plane given by $\mathbf{w}^T\mathbf{x}_T + a = 0$, $\sigma(\mathbf{w}^T\mathbf{x}_T + a)$ is proportional to the shortest euclidean distance of the input to that hyperplane. This quantifies the margin or level of confidence in the prediction and farther the input lies from the hyperplane in its upper half space, the more confident it becomes in its prediction of that input. During training using backpropagation, the hyperplane is rotated, shifted and scaled in order to maximize the expected margin of the inputs. Because of the min pooling operation, each input contributes in updating the parameters of only that hyperplane which predicts the least margin for it at that training step among all the hyperplanes involved in a rule . The rationale here is to maximize the margin of an input only from the least confident hyperplane.

4.2.2 Restricted Gradients

For an input $\mathbf{x}$ that does not belong to the activated region of a rule, $\min_j \mathbf{w}_j^T \mathbf{x}_T + a_j$ would be less than or equal to zero which implies zero gradients of all the trainable parameters as the derivative of ReLU activation for negative inputs is zero. This suggests that only the training examples lying inside the activated region are responsible for modifying the shape of this region. In order to maximize their margins, activated examples try to push or pull the rule boundaries depending on the sign of their class membership and the sign of weight, c. If both of these signs agree then the corresponding training examples push the rule boundary outwards which expand the region and as a result, brings in more training examples. Otherwise, if the signs do not match then those contradictory examples pull in the rule boundary to get themselves out of it and thereby, shrink the region.

4.2.3 Compact Convex Support

Let C denote the support of a neural rule given by equation (4.5) and defined as $\{\mathbf{x} : r(\mathbf{x}) \neq 0\}$. We show that C is a compactly supported convex set.

Proposition 1. *For any $\boldsymbol{x}, \boldsymbol{z} \in C$, the convex combination of $\boldsymbol{x}$ and $\boldsymbol{z}$*

$$\theta \boldsymbol{x} + (1 - \theta)\boldsymbol{z} \in C$$

where $\theta \in \mathbb{R}$ with $0 \leq \theta \leq 1$

Proof. Given any $\mathbf{x} \in C$, we have from equation (4.5),

$$r(\mathbf{x}) \neq 0 \iff \mathbf{w}_j^T \mathbf{x} + a_j > 0 \;\; \forall j$$

On multiplying both sides by $\theta > 0$, we get

$$\mathbf{w}_j^T \theta \mathbf{x} + \theta a_j > 0 \;\; \forall j \tag{4.6}$$

Similarly, we have for any $\mathbf{z} \in C$,

$$r(\mathbf{z}) \neq 0 \iff \mathbf{w}_j^T \mathbf{z} + a_j > 0 \;\; \forall j$$

Multiplying both sides with $(1 - \theta) > 0$,

$$\mathbf{w}_j^T (1 - \theta)\mathbf{z} + (1 - \theta) a_j > 0 \;\; \forall j \tag{4.7}$$

Adding equations (4.6) and (4.7), we obtain

$$\mathbf{w}_j^T(\theta\mathbf{x} + (1-\theta)\mathbf{z}) + a_j > 0 \;\; \forall j$$

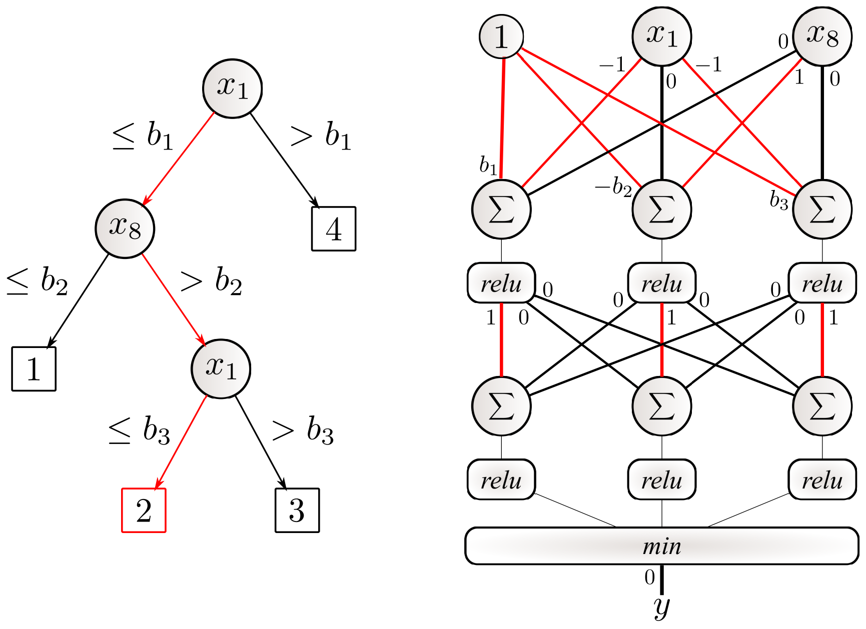
which implies

$$r(\theta\mathbf{x} + (1-\theta)\mathbf{z}) \neq 0 \iff \theta\mathbf{x} + (1-\theta)\mathbf{z} \in C$$

$\square$

4.3 Deep Neural Rule

In Section 4.2.3, we observed that the support of a proposed neural rule is a convex set. In order to allow for the rules to assume complicated non-convex shapes in the feature space, we extend the definition of a neural rule by stacking a new hidden layer with the same number of hidden units as the previous one. We refer to such a modification of neural rule as deep neural rule. Since we need to preserve the support of a tree-induced rule while mapping it into a corresponding deep neural



(a) A Decision Tree (b) A Deep Neural Rule

Figure 4.2: Mapping a Tree-induced Rule, with terminal label 2, into a Deep Neural Rule

rule, we use an identity transformation for initializing the parameters of this new hidden layer as depicted in Figure 4.2.

4.4 Simulation

In this section, we perform a simulation to illustrate the ability of a neural rule to evolve and expand its activated region. We consider a rotated XOR dataset for this experiment which is a non-linearly separable dataset since there does not exist any single hyperplane that can separate the positive training examples (shown in blue) from the negative ones (shown in red). Additionally, since we have rotated the XOR dataset by 45°, tree-based approaches such as Gradient boosted trees and Rule Selection with FSA would have hard time approximating the oblique decision boundaries as noted in §3.5 and would require a large number of trees and/or rules.

Figure 4.3 (a) shows a single neural rule just after its initialization from a corresponding tree-induced rule. Figure 4.3 (b) shows an intermediary state after training for 150 iterations. It can be seen that the rule evolves its activated region to include more examples of the same type into its support. After training for a long time, the neural rule settles into an equilibrium state consisting of only positive examples as shown in Figure 4.3 (c).

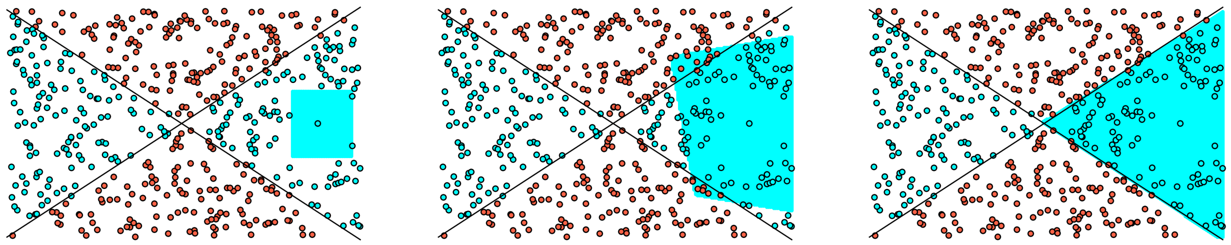

(a) Initialization (b) After 150 Iterations (c) After 3000 Iterations

Figure 4.3: Neural Rule: Evolution of the trainable support of a single rule with time

It is evident from Figure 4.3 (c) that there are still many positive training examples (shown in blue dots) which do not belong to the support of a neural rule. In order to include them, a neural rule would have to assume a non-convex shape which is not possible as shown in Proposition 1. This limitation has motivated its extension to a deep neural rule. Figure 4.4 shows the evolution of the activated region in case of a deep neural rule which can now acquire non-convex shape and hence, encompasses all the positive training examples into its support as shown in Figure 4.4 (c).

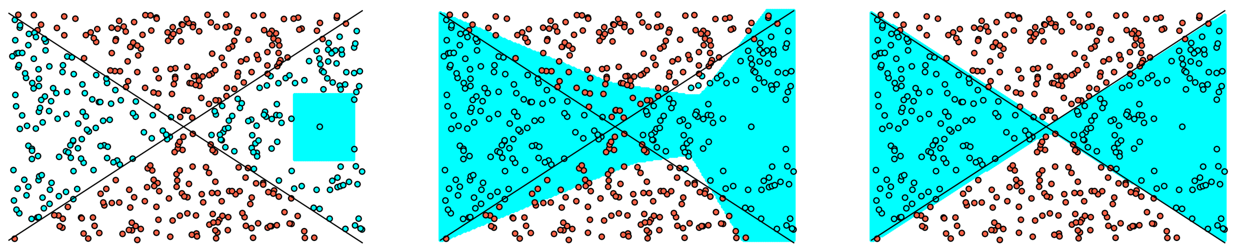

(a) Initialization (b) After 150 Iterations (c) After 3000 Iterations

Figure 4.4: Deep Neural Rule: Evolution of the trainable support of a single rule with time

CHAPTER 5

NEURAL RULE ENSEMBLES

5.1 Related Work

Tree based ensemble methods have emerged as being one of the most powerful learning methods [Friedman, 2001, Breiman, 2001] owing to the simplicity and transparency of trees, combined with an ability to explain complex data sets.

Predictive models based on rules have gained momentum in the recent years. One of the simplest rule based approach was proposed in Quinlan [1993] where a single decision tree is decomposed into a set of rules. Each such rule is pruned by removing nodes that improved its estimated accuracy. This is followed by sorting pruned rules in the ascending order of their accuracy. Prediction at any new example is obtained using a single activated rule highest in that sorted list. RuleFit [Friedman and Popescu, 2008] is another popular rule based predictive model. It involves generating a large pool of rules using existing fast tree invoking procedures. The coefficients of these rules are fit through a regularized regression. Akdemir et al. [2013] replaces the hard rules in RuleFit with soft rules using a logistic transformation. Dembczyński et al. [2008] employs gradient boosting using rule as a base classifier and rules are added iteratively to an ensemble by greedily minimizing the negative log-likelihood function. The major concern in all of these approaches is that the activated region of rules is fixed and does not allow for any training. Since the support is aligned along the feature axes, a large number of rules would be required to approximate oblique decision boundaries and therefore, would result in a loose representation of the prediction function.

Another line of work focuses on restructuring decision tree into multi-layered neural networks with sparse connections and less restrictions on the inclination of decision boundaries. One such mapping was explored in the works of Sethi [1990, 1991] which was later used by Welbl [2014] for every tree in a random forest. The mapping in Sethi [1990] replaces Heaviside unit step function with hyperbolic tangent activation which is known to suffer from vanishing gradients problem. Also, it is not apparent how to choose the contrastive hyperparameters of hyperbolic tangent activation which heavily dictate the initialization and the magnitude of gradients.

Some works along the intersection of decision trees and neural networks replace every decision node with a neural network. One such study was explored by Kontschieder et al. [2015], who learns differentiable split functions to guide inputs through a tree. The conditional networks from Ioannou et al. [2016] also use trainable routing functions to perform conditional transformations on the inputs which enables them to transfer computational efficiency benefits of decision trees into the domain of convolutional networks. This appears like an ensembling of neural networks but structured in a hierarchical fashion.

5.2 Training

Conventional procedures for generating decision tress on binary classification tasks employ either gini index or cross entropy measure. We use a new splitting criterion for invoking a decision tree based on margin maximizing rules discussed in §3.3. Let n denote the number of examples. We use subscript l to refer to the left child, r for the right child and p for the parent node. Additionally, superscripts $+$ and $-$ refer to positive and negative examples, respectively.

Assuming binary partitioning of a decision node, each split defines two simple rules $r_l(\mathbf{x})$ and $r_r(\mathbf{x})$. Using the maximum margin metric for a rule given in (3.9), the node splitting criterion can be written as follows

$$I = \frac{(n_l^+ - n_l^-)^2}{n_l} + \frac{(n_r^+ - n_r^-)^2}{n_r} - \frac{(n_p^+ - n_p^-)^2}{n_p} \tag{5.1}$$

We decompose a single decision tree into a set of conjunctive rules to obtain a pool of diverse feature interactions. Each of these rules is used to initialize their neural counterparts using mapping discussed in §4.1. Such an ensemble of neural rules, collectively referred to as Neural Rule Ensembles (NRE) is essentially a 2-layered artificial neural network with min pooling operation and thus, a universal approximator [Hornik, 1991]. However, in the proposed approach, feature interactions extracted from a decision tree are explicitly encoded into the network through its initialization. One of the characteristics of such an initialization is that the activations of any two pooled hidden units are orthogonal to each other.

After initializing the network, all the parameters are trained using Backpropagation algorithm [Rumelhart et al., 1988]. We use Adam optimization method [Kingma and Ba, 2014] with learning rate $\alpha = 0.01$ to calculate the weight updates.

Algorithm 4 Neural Rule Ensembles (NRE): Training

```
1: Preprocess the training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$: Mean Centering and Unit Standard Deviation
2: Build a decision tree using splitting criterion given in (5.1)
3: Decompose the resulting tree into a set of conjunctive rules
4: Map each tree-induced rule into a corresponding neural rule
5: Initialize each neural rule as detailed in section §4.1
6: Train an ensemble of neural rules simultaneously using Backpropagation algorithm
```

5.3 Empirical Results and Analysis

5.3.1 Datasets

In order to compare the performance of proposed algorithm with state of the art approaches, we perform empirical evaluation on simulated and multiple real datasets which ensures a wide variety of different targets in terms of their dependence on the input features.

For simulation, we use a highly non-linear and multivariate artificial dataset, MADELON, featured in NIPS 2003 feature selection challenge. It is a generalization of classic XOR dataset to five dimensions. Each vertex of a five dimensional hypercube contains a cluster of data points randomly labeled as $+1$ or -1. The five dimensions constitute 5 informative features and 15 linear combinations of those features were added to form a set of 20 redundant but informative features. Additionally, a number of distractor features with no predictive power were added and the order of the features were randomized.

For benchmarking on real datasets, we will use Penn Machine Learning Benchmark (PMLB) [Olson et al., 2017] which includes datasets from a wide range of sources such as UCI ML repository [Dheeru and Karra Taniskidou, 2017], Kaggle, KEEL [Alcalá-Fdez et al., 2011] and meta-learning benchmark [Reif, 2012]. Since we are limitiing our focus on binary classification tasks, we only consider datasets having two classes. Additionally, we remove all the datasets with fewer than 2000 training examples. This leaves us with a total of 19 datasets to base our investigation upon.

5.3.2 Statistical Tests

We use the statistical framework for hypothesis testing to investigate whether Neural Rule Ensembles (NRE) is significantly better or not compared to state of the art classifiers, namely Random Forests (RF), Gradient boosted trees (GB) and Artificial Neural Networks (ANN). A hypothesis test is a decision between two complementary hypotheses, null hypothesis H_0 and the alternate hypothesis H_1. We are trying to reject the null hypothesis that there is no difference in the classification performance of algorithms, that is, both of them perform equally well. We use the following statistical tests designed to compare two classifiers on multiple data sets.

Wilcoxon Signed-Ranks Test. In Wilcoxon signed-ranks test [Wilcoxon, 1945], we sort data sets by the magnitude of absolute difference in the performance scores of two classifiers. This is followed by assigning ranks from the lowest to the highest absolute difference. In case of ties, average ranks are assigned. Finally, a test statistic is formed based on the ranks of positive and the negative differences.

Let d_i be the difference between the performance scores of two classifiers on i^{th} data set. Let R^+ be the sum of ranks for the data sets on which NRE outperforms the other classifier, and R^- the sum of ranks on data sets where NRE gets defeated. Ranks corresponding to zero difference are split evenly between R^+ and R^-; if there is an odd number of them, one is ignored. The test statistic, T is given by

$$T = \min(R^+, R^-) \tag{5.2}$$

where

$$R^+ = \sum_{d_i>0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i) \quad R^- = \sum_{d_i<0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i) \tag{5.3}$$

For a two-tailed test with $\alpha = 0.05$ significance level, the critical value of the test statistic corresponding to $n = 19$ data sets is 46. In other words, if T is less than or equal to 46, NRE can be considered significantly better than the other classifier with $p < 0.05$ and we can reject the null-hypothesis in favor of alternate one.

Sign Test: Wins, Losses & Ties Counts. Sign test [Salzberg, 1997, Sheskin, 2007] is much weaker than the Wilcoxon signed-ranks test and will not reject the null-hypothesis unless one algorithm almost always outperforms the other. In sign test, we compare the generalization performance of classifiers by counting the number of data sets on which a classifier outperforms others.

Under the assumption that null-hypothesis is correct, that is, both classifiers perform equally well, one would expect each one of them to win on approximately $N/2$ out of N data sets. This tell us that the number of wins is distributed according to the binomial distribution.

For 19 datasets, the critical number of wins needed to reject the null-hypothesis for a two-tailed sign test at $\alpha = 0.05$ significance is 14. This implies that NRE can be considered significantly better than the other classifier with $p < 0.05$ if it is the overall winner on 14 out of 19 datasets. Since null-hypothesis is true for ties, instead of throwing them, we distribute them evenly between the two classifiers. And, we ignore one of the ties if there is an odd number of them.

5.3.3 Test Error Evaluation

In this section, we compare NRE with GradientBoost (GB), Random Forest (RF) and Artificial Neural Networks (ANN) on 19 datasets. The test errors for the datasets without a test set are obtained using five-fold cross-validation.
For each classifier, the operating settings and the tuned hyperparameters are the following:

- **Random Forests:** The number of trees used in the forest are tuned from the set $k \in \{32, 64, 128, 256, 512\}$.
- **Gradient Boosted Trees:** We use 100 boosting iterations with the maximum tree depth d selected from the range $d \in \{2, 4, 6, 8, 10\}$.
- **Artificial Neural Networks:** Fully connected networks with a single hidden layer (since NRE contains one hidden layer) and rectified linear activation. The number of hidden units $h \in \{64, 128, 256, 512, 1024\}$ selected for optimal performance.
- **Neural Rule Ensembles:** Maximum depth of the tree used for initializing the network searched over the set $d \in \{2, 4, 6, 8, 10\}$.

The hyperparameters for the methods being evaluated have been obtained by internal five-fold cross-validation on the training set. We use the scikit-learn implementation for evaluating the existing algorithms.

NRE vs Gradient Boosted Trees. From Table 5.1, it can be seen that NRE wins on 8 data sets, GB wins on 6 data sets and there are 5 ties. Ignoring one tie and splitting the remaining ones evenly, we find that NRE is better on 10 out of 19 datasets. Since the critical number of wins needed under sign test is 14, we fail to reject the null-hypothesis. Similarly, we fail to reject the null-hypothesis under Wilcoxon signed-ranks test because the test statistic $T = \min(R^+, R^-) = \min(109, 81) = 81$ is greater than 46. This implies that we don't have enough statistical evidence to establish that NRE outperforms GB. However, we realize that NRE initialized from a single tree gives a tough competition to 100 boosted trees and thus, is a more compactly represented model.

NRE vs Random Forests. We find from Table 5.2 that NRE outperforms RF on almost all the data sets except for the ring data set and the 5 tied matches. Splitting the ties evenly, NRE is better on 15 out of 19 data sets which is greater than the critical number of wins needed, that is 14, under the sign test. We can therefore reject the null hypothesis. For Wilcoxon-signed ranks test, the statistic $T = \min(R^+, R^-) = \min(176.5, 13.5) = 13.5$ is less than the critical value 46 which

	GB	NRE	difference	rank
wilt	18.60	**10.40**	8.20	19.0
madelon	14.50	**10.30**	4.20	18.0
adult	**12.91**	14.22	-1.31	17.0
phoneme	9.25	**8.14**	1.11	16.0
dis	**0.71**	1.77	-1.06	15.0
titanic	27.49	**26.89**	0.60	14.0
churn	**3.60**	4.13	-0.53	13.0
banana	9.31	**8.93**	0.38	12.0
ring	**3.15**	3.51	-0.36	11.0
spambase	**4.34**	4.63	-0.29	10.0
kr-vs-kp	0.42	**0.20**	0.22	9.0
chess	**0.21**	0.42	-0.21	7.5
coil2000	6.04	**5.83**	0.21	7.5
twonorm	2.34	**2.25**	0.09	6.0
clean2	0.00	0.00	0.00	3.0
hypothyroid	1.47	1.47	0.00	3.0
agaricus-lepiota	0.00	0.00	0.00	3.0
magic	11.67	11.67	0.00	3.0
mushroom	0.00	0.00	0.00	3.0
# wins	6	**8**		
# ties	5	5		

Table 5.1: Comparison of the test error performance of NRE with Gradient Boosted trees (GB) on binary classification tasks

allows us to reject the null hypothesis as well. This implies that NRE is significantly better than Random Forests and given that it utilizes only one tree compared to a maximum of 500 trees in RF, it is more efficient too.

NRE vs Artificial Neural Networks. It is evident from Table 5.3 that NRE outperforms ANN on 12 data sets, loses on 2 sets and there are 5 ties. NRE passes the sign test since it is better on 14 data sets (splitting the ties evenly) which matches the critical number of wins needed. Since, the test statistic for the Wilcoxon signed-rank test $T = \min(R^+, R^-) = \min(155.5, 34.5) = 34.5$ is less than the critical value 46, we reject the null-hypothesis in favor of alternate one. Both of the statistical tests agree that NRE is significantly better than the Artificial Neural Networks.

	RF	NRE	difference	rank
madelon	26.40	**10.30**	16.10	19.0
wilt	21.60	**10.40**	11.20	18.0
coil2000	7.06	**5.83**	1.23	17.0
phoneme	9.00	**8.14**	0.86	16.0
banana	9.56	**8.93**	0.63	15.0
titanic	27.49	**26.89**	0.60	14.0
spambase	4.92	**4.63**	0.29	13.0
twonorm	2.52	**2.25**	0.27	12.0
adult	14.47	**14.22**	0.25	11.0
kr-vs-kp	0.42	**0.20**	0.22	10.0
magic	11.88	**11.67**	0.21	9.0
hypothyroid	1.68	**1.47**	0.21	8.0
chess	0.62	**0.42**	0.20	7.0
ring	**3.33**	3.51	-0.18	6.0
agaricus-lepiota	0.00	0.00	0.00	3.0
mushroom	0.00	0.00	0.00	3.0
dis	1.77	1.77	0.00	3.0
clean2	0.00	0.00	0.00	3.0
churn	4.13	4.13	0.00	3.0
# wins	1	**13**		
# ties	5	5		

Table 5.2: Comparison of the test error performance of NRE with Random Forests (RF) on binary classification tasks

	ANN	NRE	difference	rank
madelon	45.50	**10.30**	35.20	19.0
phoneme	14.18	**8.14**	6.04	18.0
wilt	14.20	**10.40**	3.80	17.0
churn	6.27	**4.13**	2.14	16.0
coil2000	7.46	**5.83**	1.63	15.0
spambase	**3.47**	4.63	-1.16	14.0
ring	**2.52**	3.51	-0.99	13.0
magic	12.44	**11.67**	0.77	12.0
adult	14.79	**14.22**	0.57	11.0
hypothyroid	1.89	**1.47**	0.42	9.5
kr-vs-kp	0.62	**0.20**	0.42	9.5
banana	9.31	**8.93**	0.38	8.0
twonorm	2.43	**2.25**	0.18	7.0
dis	1.94	**1.77**	0.17	6.0
agaricus-lepiota	0.00	0.00	0.00	3.0
mushroom	0.00	0.00	0.00	3.0
clean2	0.00	0.00	0.00	3.0
chess	0.42	0.42	0.00	3.0
titanic	26.89	26.89	0.00	3.0
# wins	2	**12**		
# ties	5	5		

Table 5.3: Comparison of the test error performance of NRE with Artificial Neural Networks (ANN) on binary classification tasks

CHAPTER 6

CONCLUSION

In this work, we presented a novel method for encoding feature interactions captured by a decision tree into neural networks called Neural Rule Ensembles (NRE). This was achieved by defining a neural transformation of a tree-induced rule using ReLU units and the min pooling operation as detailed in section §4. Such a mapping addressed the initialization related concerns of fully connected neural networks and enabled learning of compact representations compared to conventional tree-based approaches.

Empirical evaluations on 19 binary classification datasets from Penn Machine Learning Benchmark (PMLB) [Olson et al., 2017] were performed to compare the generalization performance of Neural Rule Ensembles (NRE) with state of the art approaches such as Random Forests (RF), Gradient Boosted Trees (GB) and Artificial Neural Networks (ANN). We used two statistical tests, Wilcoxon signed-ranks test and the sign test, to access the statistical significance of these results. Both of these statistical tests found NRE to be significantly better than Random Forests and the Artificial Neural Networks with $p < 0.05$. When NRE was compared to Gradient Boosted Trees, we could not find enough statistical evidence to reject the null hypothesis stating that both of them perform equally well. However, NRE was more efficient since it utilized only one tree.

Another contribution of this work was to develop a method for obtaining compact tree ensembles based on loss minimization that involves generating a large pool of trees followed by selecting a desired number of trees while simultaneously updating their leaf weights using the FSA algorithm [Barbu et al., 2017]. We provided a mathematical derivation for the group criterion employed in conjunction with FSA for selecting relevant trees. In order to sped up the model selection process, we discussed a modification in FSA algorithm that obtains a whole range of models corresponding to different sparsity levels in just a single run.

We also provided a natural generalization of the FSA algorithm to tree-induced rules called Rule Selection with Annealing (RSA). We showed that the selection criterion used in RSA measures the square of total weighted margin of a rule and at any iteration, is proportional to the square of cosine similarity of that rule with the gradient boosted direction up to that iteration.

REFERENCES

Deniz Akdemir, Nicolas Heslot, and J.-L Jannink. Soft rule ensembles for supervised learning. pages 78–83, 01 2013.

Jesús Alcalá-Fdez, Alberto Fernández, Julián Luengo, Joaquín Derrac, and Salvador García. Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *Multiple-Valued Logic and Soft Computing*, 17(2-3):255–287, 2011. URL http://dblp.uni-trier.de/db/journals/mvl/mvl17.html#Alcala-FdezFLDG11.

Adrian Barbu and Nathan Lay. An introduction to artificial prediction markets for classification. *The Journal of Machine Learning Research*, 13(1):2177–2204, 2012.

Adrian Barbu, Yiyuan She, Liangjing Ding, and Gary Gramajo. Feature selection with annealing for computer vision and big data learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(2):272–286, 2017.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

F. Bunea, A. Tsybakov, and M. Wegkamp. Sparsity oracle inequalities for the lasso. *Electronic Journal of Statistics*, 1:169–194, 2007.

Krzysztof Dembczyński, Wojciech Kotlowski, and Roman Slowiński. Maximum likelihood rule ensembles. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 224–231, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390185. URL http://doi.acm.org/10.1145/1390156.1390185.

Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

David L Donoho, Michael Elad, and Vladimir N Temlyakov. Stable recovery of sparse overcomplete representations in the presence of noise. *IEEE Transactions on Information Theory*, 52(1):6–18, 2006.

J. Fan and R. Li. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association*, 96(456):1348–1360, 2001.

Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119 – 139, 1997. ISSN 0022-0000.

J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting.(with discussion and a rejoinder by the authors). *Annals of Statistics*, 28(2):337–407, 2000.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.

Jerome H Friedman and Bogdan E Popescu. Predictive learning via rule ensembles. *The Annals of Applied Statistics*, pages 916–954, 2008.

Pierre Geurts. Some enhancements of decision tree bagging. *PKDD*, pages 141–148, 2000.

Isabelle Guyon, Steve Gunn, Asa Ben-Hur, and Gideon Dror. Result analysis of the nips 2003 feature selection challenge. In *NIPS*, pages 545–552, 2004.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4 (2):251–257, 1991.

Yani Ioannou, Duncan P. Robertson, Darko Zikic, Peter Kontschieder, Jamie Shotton, Matthew Brown, and Antonio Criminisi. Decision forests, convolutional networks and the models in-between. *CoRR*, abs/1603.01250, 2016.

Arnaud Joly, François Schnitzler, Pierre Geurts, and Louis Wehenkel. L1-based compression of random forest models. In *20th European Symposium on Artificial Neural Networks*, 2012.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://dblp.uni-trier.de/db/journals/corr/corr1412.html#KingmaB14.

K. Knight and W. Fu. Asymptotics for lasso-type estimators. *Annals of Statistics*, pages 1356–1378, 2000.

P. Kontschieder, M. Fiterau, A. Criminisi, and S. R. Bulò. Deep neural decision forests. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1467–1475, Dec 2015. doi: 10.1109/ICCV.2015.172.

Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pages 9–50, London, UK, UK, 1998. Springer-Verlag. ISBN 3-540-65311-2. URL http://dl.acm.org/citation.cfm?id=645754.668382.

P. Li. Robust logitboost and adaptive base class (abc) logitboost. In *UAI*, 2010.

S.Z. Li and Z.Q. Zhang. Floatboost learning and statistical face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1112–1123, 2004.

M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. ISBN 0070428077, 9780070428072.

Randal S. Olson, William La Cava, Patryk Orzechowski, Ryan J. Urbanowicz, and Jason H. Moore. Pmlb: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, 10(1):36, Dec 2017. ISSN 1756-0381. doi: 10.1186/s13040-017-0154-4. URL https://doi.org/10.1186/s13040-017-0154-4.

A. Painsky and S. Rosset. Compressing random forests. In *ICDM*, pages 1131–1136, Dec 2016.

J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1-55860-238-0.

Matthias Reif. A comprehensive dataset for evaluating approaches of various meta-learning tasks. In Pedro Latorre Carmona, J. Salvador Sánchez, and Ana L. N. Fred, editors, *ICPRAM (1)*, pages 273–276. SciTePress, 2012. ISBN 978-989-8425-98-0. URL http://dblp.uni-trier.de/db/conf/icpram/icpram2012-1.html#Reif12.

Shaoqing Ren, Xudong Cao, Yichen Wei, and Jian Sun. Global refinement of random forest. In *CVPR*, pages 723–730, 2015.

Lev Reyzin. Boosting on a budget: Sampling for feature-efficient prediction. In *ICML*, 2011.

Cynthia Rudin, Ingrid Daubechies, and Robert E Schapire. The dynamics of adaboost: Cyclic behavior and convergence of margins. *The Journal of Machine Learning Research*, 5:1557–1595, 2004.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988. ISBN 0-262-01097-6. URL http://dl.acm.org/citation.cfm?id=65669.104451.

Steven L. Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1(3):317–328, Sep 1997. ISSN 1573-756X. doi: 10.1023/A:1009752403260. URL https://doi.org/10.1023/A:1009752403260.

I. K. Sethi. Entropy nets: from decision trees to neural networks. *Proceedings of the IEEE*, 78(10):1605–1613, Oct 1990. ISSN 0018-9219. doi: 10.1109/5.58346.

Ishwar K. Sethi. Decision tree performance enhancement using an artificial neural network implementation1 1this work was supported in part by nsf grant iri-9002087. In Ishwar K. SETHI and Anil K. JAIN, editors, *Artificial Neural Networks and Statistical Pattern Recognition*, volume 11 of *Machine Intelligence and Pattern Recognition*, pages 71 – 88. North-Holland, 1991. doi: https://doi.org/10.1016/B978-0-444-88740-5.50010-4. URL http://www.sciencedirect.com/science/article/pii/B9780444887405500104.

David J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & Hall/CRC, 4 edition, 2007. ISBN 1584888148, 9781584888147.

Johannes Welbl. Casting random forests as artificial neural networks (and profiting from it). In Xiaoyi Jiang, Joachim Hornegger, and Reinhard Koch, editors, *Pattern Recognition*, pages 765–771, Cham, 2014. Springer International Publishing. ISBN 978-3-319-11752-2.

Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945. ISSN 00994987. URL http://www.jstor.org/stable/3001968.

Cun-Hui Zhang. Nearly unbiased variable selection under minimax concave penalty. *Annals of Statistics*, pages 894–942, 2010.

P. Zhao and B. Yu. On model selection consistency of lasso. *Journal of Machine Learning Research*, 7(2):2541, 2007.

Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.

BIOGRAPHICAL SKETCH

Gitesh Dawer received his B.Tech. degree in Aerospace Engineering from the Indian Institute of Technology, Kanpur in 2013 where he was awarded the General Proficiency Medal for best academic performance among the graduating batch. Owing to his curiosity in Aeroacoustics, he went on to pursue a M.S degree in Applied and Computational Mathematics at Florida State University.

Upon the completion of his M.S. degree in 2015, he became interested in Machine Learning and started working on his PhD under the supervision of Dr. Adrian Barbu. He has worked on a wide range of projects including interpretable models, feature selection methods and object detection. During this time, he also served as a solo instructor of calculus for three semesters.

His research interests include representation learning, boosting, ensemble methods, reinforcement learning and causality.