Neural Decision Tree and Forest Classifiers: With Application to Electroymyogram (EMG) Gesture Classification

A thesis presented to the faculty of
San Francisco State University
In partial fulfilment of
The Requirements for
The Degree

AS
36
2016
MATH
.R63

Master of Arts
In
Mathematics

by

David Rodriguez

San Francisco, California

May 2016

CERTIFICATION OF APPROVAL

I certify that I have read *Neural Decision Tree and Forest Classifiers: With Application to Electroymyogram (EMG) Gesture Classification* by David Rodriguez and that in my opinion this work meets the criteria for approving a thesis submitted in partial fulfillment of the requirements for the degree: Master of Arts in Mathematics at San Francisco State University.

[illegible]

Alexandra Piryatinska
Associate Professor of Mathematics

[illegible]

Xiaorong Zhang
Assistant Professor of School of Engineering

[illegible]

Arek Goetz
Professor of Mathematics

Neural Decision Tree and Forest Classifiers: With Application to Electroymyogram (EMG) Gesture Classification

David Rodriguez
San Francisco State University
2016

As the amount of data increases from sensors, networks, and data stored in vector form so does the need to detect anomolies and classify it. My research goals in this thesis is to develop learning models that can be applied on raw signals and assist and automate the process of feature generation and selection. The goal of such models is to synthesize multiple signals sampled simultaneously for the purpose of classifying human movement.

There has been great success in using multi-resolution techniques like wavelet decomposition and hand-crafted features in classifying human movement from electromyogram (EMG) signals. However, these methods either make simplifying assumptions about the features and or introduce redundancies that require subsequent dimensionality reduction techniques before the data can be used as input to train a model. Additionally, an emphasis on tuning a single model is usually applied. My goal is to develop an ensemble of learners (i.e. models) capable of taking raw signals as input and finding the most defining features, that when aggregated perform better than any single individual. The models in this thesis, therefore, address previous shortcomings by providing effective signal processing and generalization to unseen data not used in training.

The new model in this thesis is a variant of the *Neural Decision Forests* introduced by Bulo and Kontschieder. The model is also a generalization of the more widely known random forest algorithm introduced by Breiman because it is a multivariate decision tree capable of embedding feature generation and selection. This variant of neural decision trees and forests are competitive with the CART trees and random forest counter-parts in classification problems found in the UCI machine learning repository datasets.

Chapter 2 is an introductory chapter that introduces decision trees and neural networks - these form the building blocks of neural decision trees and forests. Chapter 3 focuses on neural decision trees with varying neural network architectures compared to a baseline CART style decision tree. Here it is shown that the neural decision tree improves upon the classification performance of the CART decision tree on the iris dataset and is competitive in the image-segmentation dataset.

Chapter 4 focuses on neural decision forests. The accuracy perforance is compared among a variety neural decision forests with varying neural networks architectures, but this time by inducing randomization, so that the neural networks within a tree are not uniform. Here it is shown that the neural decision forests are competitive with the classificaiton performance of the random forest algorithm on the iris, image-segmentation, and handwritten datasets.

Chater 5 focuses on applying neural decision forests on EMG signals and classifying hand gestures recorded in the signals. The highlight here is that the neural decision forest can be trained on raw signals by incorporating a random window technique effectively embedding the feature generation and selection. It is shown that the neural decision forest trained on raw signals and signals with features extracted prior outperform the random forest algorithm with smaller forest sizes and are competitive at larger forest sizes. In addition, the neural decision forest proves robust under a variety of classification tasks where signals are sampled at high and low frequencies and under gesture recognition between and within individuals.

I certify that the Abstract is a correct representation of the content of this thesis.

[illegible] 04/21/2016

Chair, Thesis Committee Date

ACKNOWLEDGMENTS

This thesis would not have been possible without the inspiration, review, and support from my peers, advisors, and role-models. Thank you Dr. Piryatinska for your expertise in time series and your belief in me down the many paths where this research began. And Dr. Zhang, thank you for your expertise in neural machine interfaces and helping hone my ability to research topics methodically and to iterate and refine code and writing over time. I also want to thank all my professors from San Francisco State University, you have trained me to think hard and articulate questions in writing in the pursuit of research.

Of course without my family and friends I would not have had the energy or the passion to undertake the steps towards this culminating experience. I love you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

In this introductory chapter, we raise two problems this thesis wishes to solve. The first problem relates to traditional signal processsing techinques producing too many features to train a model. The second, remarks on the necessity of building a learning algorithm capable of generalizing to data not used in training. We conclude with an outline of the chapters.

1.1 Overview

As the amount of data increases from sensors, networks, and data stored in vector form so does the need to detect anomolies and classify it. My research goals in this thesis is to develop learning models that can be applied on raw signals to assist and automate the process of feature generation and selection. The goal of such models is to synthesize multiple signals sampled simultaneously for the purpose of classifying human movement.

There has been great progress in delivering technologies in neural machine interfaces (NMI) such as prosthetic limbs and hands-free devices [44]. These technologies extract complex information from real-time applications. One of the goals of the NMI is the development of general and scalable algorithms that can jointly solve user's intention and learn the necessary intermediate feature representations of the user's range of movement contained in normal motion [63]. However, standard approaches towards this goal have two common shortcomings.

1. **Too Many (and redundant) Features** In signal processing, two recipes are usually followed to extract features: i) applying a window uniformly over the frame of a signal and extracting some time domain frequency measures like the number of *zero-crosings, absolute mean signal length, etc.*, or ii) using a multi-resolution technique (i.e. fast-fourrier or wavelet decomposition) [50]. The downfall of these methods is seen when considering sensors recording muscle frequencies of a human making a gesture: e.g. pronating, supinating, etc., and training a model to classify them. The difficulty of the sliding window technique and selecting hand-crafted time domain features is the possibility of redundant features [46]. On the other hand, the difficulty with multi-resolution techniques is that they produce a large amount of features typically equal to the number of data-points sampled. Said differently, both techniques usually require further dimensionality reduction before the training of a model can commence with the supplied data.

2. **Models Unable to Generalize** While a model's performance depends crucially on the feature representations used as input [6], time can be lost tuning a model incapable of generalizing properly [7]. For example, it is known that decision trees do not generalize to new variations of data not used in training [8]. That is, a decision tree trained on raw EMG signals to label two gestures that are the same, but the recorded EMG signals appear translated, will require an exponential amount of training cases or depth in the tree to perform the required classification if it can at all. Relevant to this thesis, models face the challenge of i) distinguishing a variety of related gestures from a single individual, and ii) distinguishing a variety of gestures recorded on different individuals. Both these challenges push the limits of how well a model can generalize.

The models in this thesis culminate in a technique addressing these two shortcomings. Addressing the first problem, these models generate successive windows of various sizes capable of capturing both local and global deviances in EMG signals and controling the number of features generated. In addition, embedded neural networks are trained with l_2 regularization and a variety of architectures for further feature selection. This is in contrast to the aforementioned *window/slide* and *multi-resolution techniques* usually requiring additional dimensionality reduction techniques prior to training a model. As for the second problem, we explore an ensemble of neural decision trees which partition the induced feature space into regions that are subsequently assigned a probability of each class label appearing. By bagging neural decision trees into a forest, we deploy a method capable of generalizing to new variations of data not seen in training.

The models herein provide both competitive techniques with previously established benchmarked algorithms and of particular importance the random forest algorithm. Furthermore, they provide state of the art performance with respect to training and using hand-crafted features and performing feature generation and selection. Inspiration for these new models come from the ideas derived in the two methods: a) decision trees, and b) neural networks, and working on a unified technique incorporating the successes of each.

1.2 Contributions and Outline of This Thesis

Previous multivariate decision tree work is focused on classification using one tree. In contrast, the forest models presented in this thesis provide compelling reasons for adopting bootstrapped and aggregated methods especially when underlying heirarchical structures may be nested in class labels. This constitutes an important foundation previously neglected in the field of multivariate decision trees.

The neural decision forest algorithm was first introduced by Bulo and Kontschieder focused on neural decision trees and neural gating functions $f(x)$ with $x \in \mathcal{X}$ with 1-or-more hidden layers and using Bayesian methods to optimally find the networks defining weights [51]. In contrast, I focus on constructing a forest of neural decision trees with neural gating functions using neural networks with 1-hidden layer. In ways, I wished to maintain the spirit of the original CART algorithm introduced by Breiman [12], using entropy measures and greedy searches, but achieve similar improvements over the random forest algorithm like that achieved by Bulo and Kontschieder. Restricting embedded neural networks to one layer enabled performance gain and made feasible larger forest sizes. To my knowledge this is the first study of this kind.

In terms of signal processing, this is the first study applying neural decision forests on EMG signals for gesture classification. In addition, we believe we have paved a new approach of randomly windowing multiple EMG signals simultaneously that are recording an event.

Chapter 2 is an introductory chapter that introduces general decision trees and neural networks usually found in introductory textbooks. The next three chapters outline the main neural decision tree classifiers into: i) trees, ii) forests, iii) signal processing. A final chpater distills the findings and discusses shortcomings, advantages and potential future directions.

Summary Chapter 3: Neural Decision Trees

The first modeling choice I investigated is a single neural decision tree classifier and compared its peformance to that of the CART decision tree classifier. I first outline the implementation details then explore the performance of the classifier on two publicly accessible machine learning datasets: Iris and Image Segmentation. In addition, we staged a comparative study of the CART and neural decision tree classifiers on a synthetic data set resembling problems in image analysis. We conclude that the neural decision tree classifier is capable of outperforming the decision tree with various presets. For example, one preset we explored was to hot start the net weight training using *random projections*. The intuition was that each neuron is a logistic function as in linear models, so why not provide an approximate hyperplane seperating classes as initial parameters. The results of this and other neural decision tree presets are explored in this chapter.

Summary Chapter 4: Neural Decision Forests

The second modeling choice I investigated is an ensemble of neural decision tree classifiers and compared their performance to that of the random forest classifier. Here I wished to establish a baseline of each algorithm and so used datasets with already established features. I first outline the implementation details then explore the performance of the classifier on three publicly accessible machine learning datasets: Iris, Image Segmentation, and Handwritten Letters. In addition, we staged a comparative study of the random forest and neural decision forest classifiers on the synthetic dataset used in chapter 3. We conclude that the neural decision forest classifier has comparable performance to random forest algorithm, but based on the number of presets possible in a neural decision forest the door is open to imporovement.

Summary Chapter 5: Neural Decision Forests and Personalized EMG Gesture Classification

The last modeling choice I investigated was a variant of neural decision forest capable of training on raw electromyogram (EMG) signals using random windows. That is, we have here a method embedding feature generation. To train a neural decision forest on raw EMG signals, I first outline the new alterations to the neural decision forests discussed in the previous chapter and explore the performance against a neural decision forest and random forest trained on features extracted prior to training. The hallmark study is a personalized gesture classification experiment where we couple a leave-one-user-out technique with k-folding. We conclude that the neural decision forest trained on raw signals significantly improves all peformance measures when compared to all models using a small number of trees per forest.

The next chapter will give the necessary mathematical background of decision trees and neural networks and their training induced by loss functions. We will also see more motivation for why a unification of each method is reasonable to explore.

Chapter 2

Decision Tree / Neural Network Background

In this chapter, I will motivate the reasons one might wish to study neural decision forests. This will be approached by discussing how neural decision trees and deep learning intersect in similar approaches. Next, I will define the most basic form of a decision tree and neural network. I will end this chapter with a short introduction to loss functions; loss functions being the cornerstone of any optimization of a statistical or machine learning algorithm.

Neural decision trees rely on the foundations of decision trees and neural networks. These machine learning techniques may both accomplish the goal of discriminative learning but are founded on different approaches. Better understanding each method indvidually will instruct the expectations of a unified technique built from both. Decision trees and neural networks both have nice mathematical descriptions.

2.1 Neural Decision Trees and Deep Learning

In this section, I outline how neural decision trees are related to representation learning commonly used in deep learning techniques.

2.1.1 The Need for Feature Generation and Selection

Hand-crafting features are often simplified abstractions leaning on domain-specific knowledge. The difficulty with hand-crafted features is i) they require a fair amount of ingenuity to define, and ii) potentially capture redundant information. Facing these problems, one might just wish to permit extraneous features, but the increase of variables to training set size suffers from the curse of dimensionality when subsequently training a model [5]. To combat this, some try to reduce the number of explanatory variables in the training data with methods like principal component analysis [31] while others incorporate regularization methods in the model [65]. If a learned model could incorporate both feature generation and selection then a model may reduce it's reliance on hand-crafted features.

2.1.2 Trees of Different Features

One way to learn features is to introduce bottle-necks into neural network architectures. Some well-known methods are to use auto-encoders or sparse-encoding stacked atop one another forming a network [59, 18]. The goal is to filter the original input in stages before the final task of discriminative learning. Said differently, these methods delay the final predictions of a learner focusing foremost on techniques to untangle the underlying covariance of input features [6]. Seen in this way, neural decision trees fit this mold. In particular, the neural networks in a neural decision tree can be seen to route and filter the original input. But unlike the random forest algorithm, these networks are capable of encoding and decoding the input feature covariance relationships before a final routing is produced. In addition, these neural networks may be trained without necessarily terminating the learning process, that is these networks are not trained to minimize prediction error (at least not immediately) and serve as filters.

2.1.3 Ensembles and Feature Generation

One more alternative to one-shot embedding feature generation and selection within a model is to induce an ensemble technique. The benefit, is that if the feature generation method is generic and random enough, then repeated attempts should improve performance. This process of ensemble learning has poven powerful by inducing strategic randomization to a series of similarly constructed learners [20, 13, 2] with success in decision trees [21] and neural networks [28]. For example, the error of a model can be broken down into the three components: i) bias, ii) variance, and iii) irreducible error. Therefore, if we can control the bias and variance of a model we can control the overall error [29]. So if the individual learners have low bias but high variance, the learners considered in aggregate can decrease the overall variance and thus outperform any one individual learner [13].

2.1.4 Computational Challenges

The models discussed herein become progressively more computationally expensive with each chapter. At the most extreme is our neural decision forest trained on raw signals. With the aide of Amazon Web Services (AWS) we have successsfully parallelized forest generation and stored intermediate results not otherwise capable without 20+ CPU cores and 50+ GB RAM. While this thesis does not explicitly explore speed performance optimizations, there are a variety of approaches one may wish to investigate. For example, one may focus on finding more efficient methods of training neural networks, or focus on parallelizing the construction of each tree in a given forest.

It is in this intersection of math and software engineering that my research has developed. The next section will describe the related basics of decision trees and neural networks.

2.2 Decision Trees

In this section, I will give a basic introduction to decision trees [12]. Figure 2.1 shows the schema of a learned decision tree consisting of root (upper most node), internal

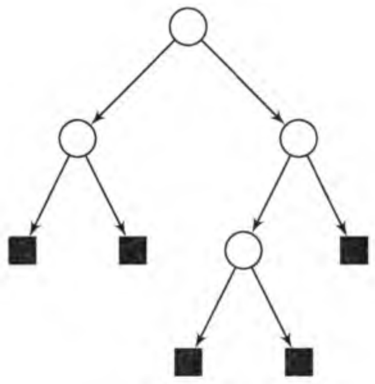


Figure 2.1: Schema of a decision tree. Internal nodes (circle) leaves (square).

(nodes with sub-nodes), and leaves (nodes without sub-nodes). To classify a new vector $x_i \in \mathcal{X}$, imagine a pinball atop the tree at the root. The pinball represents x_i. This pinball flows through the tree based on the results of each test at internal nodes. If the pinball is at a leaf, the label associated with the leaf becomes the predicted class y_i.

Input to learn a decision tree is a collection of *training cases*: $S = \{(x_i, y_i)\}_{i=1}^{m}$ with $x_i \in \mathcal{X}$[1] and $y_i \in \mathcal{Y}$ where $\mathcal{Y}$ is a set of classes. The goal is to learn from S a function,

$$t : \mathcal{X} \longrightarrow Y$$

such that $x_i \mapsto y_i$. This maps from the d *attributes* or *components* of $x_i \in \mathcal{X}$ to a predicted class $y_i \in Y$. In section 2.4 we discuss how to obtain an optimal t by reducing the error over S.

[1]for our purposes $\mathbb{R}^d$.

Decision trees are characterized by a recursive structure where a *leaf* maintains a label with a class value from Y and *internal* nodes are considered tests, with (typically) two outcomes, each linked to a subtree [49]. You may notice each subtree in a decision trees has a simple pattern: a root node and two children. This is a characteristic of divide and conquer algorithms, where recursively applying a rule on smaller subsets yields results for the original set [19].

The *divide and conquer* algorithm used to construct a tree, t, from a training set S partitions S or construct arrangments, or partitions, of the feature space $\mathcal{X}$ containing unique classes of Y [49]. Each internal node of a tree performs a similar test as other internal nodes. The division of S follows the algorithm,

- Return a leaf labeled y_j, if all Y in S are the same class (say y_j).
- Otherwise, form two chidren S_L and S_R using a funciton $f : \mathcal{X} \longrightarrow \mathbb{R}$ with $S_L = \{(x, y) : f(x) < \theta\}$ and $S_R = \{(x, y) : f(x) \geq \theta\}$ that are more homogeneous in terms of the distribution of classes.

The above algorithm produces a binary tree structure originally introduced by Breiman and used in the CART algorithm [12]. Further, the algorithm may be altered by considering multiple outcomes[2] producing multiple sets $S_1, \ldots, S_k$. This variation is explored by Quinlin [49, 32].

The divide and conquer stage relies on the test f known as a candidate test and acts as a gating function. At an internal node the test f has two flavors. One, is

[2]For example if $\theta_1 < f(x) \leq \theta_2$

a single attribute test known as a univariate test , the other a multi-attribute test, known as a multivariate test. The *univariate* test is very popular and first studied by Breiman and Quinlin [12, 49, 32], *multivariate* tests generalize these early methods [62]. The tests define regions in the feature space $\mathcal{X}$ with designated labels from Y. There is a recipe to the candidate test.

For example, let f be a test that is either univariate or multivariate. Suppose $\mathcal{X} = \mathbb{R}^n$. Then, a typical test is of the form

$$f(x_i) \geq \theta$$

where $\theta \in \mathbb{R}$. Or, alternatively,

$$f(x_i) \in A$$

where $A = \{a_1, \ldots, a_t\}$ is a set. Both cases reduce to either finding or setting θ or A, given f. The pair (f, θ) or (f, A) are found by a greedy search of the values that maximize the value of the splitting criterion with respect to S.

An example of various univariate pairs (f, θ) is as follows. Let

$$S = \{((1,2), A), ((2,1), B), ((1,1), A)$$

and consider the two projection functions $f_1(x_1, x_2) = x_1$ and $f_2(x_1, x_2) = x_2$ and let $\theta = 2$. Notice,

$$S_L = \{(x, y) : f_1(x_1, x_2) < \theta\} = \{((1, 2), A), ((1, 1), A)\}$$

while on the other hand

$$S_L = \{(x, y) : f_2(x_1, x_2) < \theta\} = \{((2, 1), B), ((1, 1), A)\}$$

and similarly for S_R with each f_1 and f_2.

In contrast to the above example, to form a multivariate test we could consider two functions $f_1(x_1, x_2) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$ and $f_2(x_1, x_2) = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2$. In other words, a multivariate test, is a possibly linear or non-linear combination of the features $\mathcal{X}$. Depending on the test f there are known algorithms to efficiently determine the split points θ [17].

Finally, we determine the optimal pair (f, θ) splitting S into S_L and S_R by measuring the decrease in disparity of class labels using an *entropy* measure. We detail how to split S is Section 2.4. Alternative measures are the gain ratio [49], twoing [12], and using posterior probabilities [22].

2.3 Neural Networks

In this section, I will give a basic introduction to feed-forward neural networks [4]. These neural networks will act as candidate tests f in a decision tree. Figure 2.2 shows the schema of a learned neural network consisting of input (circle), bias

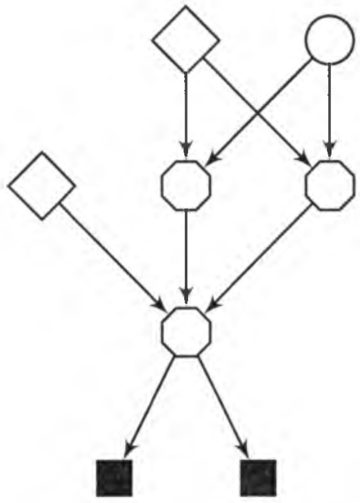


Figure 2.2: Schema of a neural network. Input (circle) bias (diamond) and activation function (octagon).

(diamond), and neuron/activation function (octagon) classes (square). To classify a new vector $x_i \in \mathcal{X}$, components of x_i are sent through the graph until reaching the final neuron/activation function where if the final value is less than a threshold the value is routed left and label assigned at the square, otherwise routed right and label assigned. The components travel by multiplication with edge weights and used as input to an neuron/activation function, resulting values then similarly travel subsequent edges.

Input to learn a neural network is a collection of *training cases*: $S = \{(x_i, y_i)\}_{i=1}^{m}$. The goal is to learn from S a function,

$$f : \mathcal{X} \longrightarrow Y$$

such that $x_i \mapsto y_i$. As in linear regression one learns the coefficients, learning a neural network f reduces to learning the edge weights of the networks. In fact, neural networks generalize linear models. Logistic regression models can be interpreted as a single layer neural network consisting of a single neuron. More generally, for example, if in Figure 2.2 the circle and octagons map the inputs through a logistic function $g(z) = [1 + e^{-z}]^{-1}$ we see this network as a complex of logistic models.

You may notice neural networks have a simple pattern: possibly one logistic function after another. This leads to a nice mathematical description. Let a_i be a neuron's activation. This can be computed as the inner product of the input with its parameters, followed by addition with its bias: $a_i = f(W_i x + b_i)$ with weights $W_i \in \mathbb{R}^n$. We can disentangle the multiplication and the nonlinearity and write this in matrix notation for m many neurons stacked horizontally as:

$$z = Wx + b \tag{2.1}$$

$$a = f(z) \tag{2.2}$$

where $w \in \mathbb{R}^{m \times n}$ and the f is applied element-wise:

$$f(z) = f([z_1, \ldots, z_m]) = [f(z_1), \ldots, f(z_m)] \tag{2.3}$$

The output of such a neural network layer can be seen as a transformation of the input capturing various interactions with the output. Adding subsequent layers gives alternative routes to disentangle inputs with output. When multiple layers are stacked on top of each other we will use a superscript to denote the various matrices that parameterize each layer's connections:

$$z_1 = W_1 x + b_1 \tag{2.4}$$

$$a_1 = f(z_1) \tag{2.5}$$

$$z_2 = W_2 a_2 + b_2 \tag{2.6}$$

$$a_2 = f(z_2) \tag{2.7}$$

Notice figure 2.2 reduces to $W_1 \in \mathbb{R}^{2\times 2}$ and $W_2 \in \mathbb{R}^{3\times 1}$. The last layer (here the 2nd layer) is then given to an output layer on which an error function is minimized. We discuss error functions and finding the optimal set of weight for a neural network in section 2.4. Standard examples are a softmax classifier and training with the cross-entropy error. But, just as in linear regression, a linear output layer may be minimized with least squares error.

2.4 Loss Functions and Optimization

Error Leaf Assignment

Let t be the root node of a tree, i.e. the predicted classes for the initial training set S. Let $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ represents the elements of S. We will determine what predicted class should be for each x_i: $t(x_i)$. For more details refer to the MIT Prediction: Machine Learning and Statistics course by Cynthia Rudin [52].

Consider the empirical error using least squares loss:

$$J(t) = \sum_i (y_i - t(x_i))^2 \tag{2.8}$$

Notice, $\min\{J(t)\} = 0$ if tree t is grown until leaves are pure. Therefore, $t(x_i) = y_i$. The following method is used to begin the process of defining a set of splitting rules associated with t. We will derive what to label a node with a distribution of classes $y \in \mathcal{Y}$.

Consider spliting the training set S into left and right subsets and producing two leaves. Then we can calculate the error,

$$J(t) = \sum_{i \in L} (y_i - t(x_i))^2 + \sum_{i \in R} (y_i - t(x_i))^2 \tag{2.9}$$

Let t_L, t_R represent the predicted values for each x_i in a leaf since it's constant. We wish to choose t_L, t_R to minimize the error in each leaf. Consider the left node.

We can minimize this error taking the derivative with respect to t_L:

$$\begin{aligned} 0 &= \frac{d}{dt_L}\sum_{i\in L}(y_i-\tilde{t})^2\Big|_{\tilde{t}=t_L} && (2.10)\\ &= -2\sum_{i\in L}(y_i-\tilde{t})\Big|_{\tilde{t}=t_L} && (2.11)\\ &= -2\left(\sum_{i\in L}y_i-|L|\tilde{t}\right)\Bigg|_{\tilde{t}=t_L} && (2.12)\end{aligned}$$

with $|L|$ the number of elements from S in the left leaf. We see that $t_L = \frac{1}{|L|}\sum_{i\in L} y_i$ concluding the sample average of labels for leaf L. This is the value to assign for each $t(x_i)$ in each L, R leaf. Here, we have a tree with two leaves, in general, we apply the same rule to a tree with any number of leaves. This example assumed we knew how to split.

Optimal Splitting Pair (f, θ)

Finally, we determine the optimal pair (f, θ) splitting S into S_L and S_R by measuring the decrease in disparity of class labels using an *entropy* measure. Alternative measures are the gain ratio [49], twoing [12], and using posterior probabilities [22].

We define the information entropy of a set S as

$$I(S) = \sum_{p_i} p_i \log(p_i)$$

where p_i is the probability of being class i in S.Therefore, we can can compute the improvement of S_L and S_R over S by computing the gain as,

$$I(S) - p_L I(S_L) - (1 - p_L) I(S_R)$$

where p_L is the probability of being in subset S_L. The optimal pair (f, θ) is found by greedily searching all possible f's and θ's.

In otherwords, to split a node we greedily find the test (f, θ) pair. That is we find the maximum gain

$$\max_{f,\theta} \{I(S) - p_L I(\{(x, y) : f(x) < \theta\}) - (1 - p_L) I(\{(x, y) : f(x) \geq \theta\})\}$$

Error Backpropogation

Let $f(x; W)$ be a neural network. Suppose $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ where $\mathcal{Y} = \{0, 1\}$. Let us deifne the cross entropy loss:

$$J(W) = \sum_i \log(f(x^{(i)}; W)^{y_i} (1 - f(x^{(i)}; W))^{1-y_i}) \tag{2.13}$$

we wish to minimize J, requiring us to take partial $\partial J(W)/\partial w_{ij}$ for each w_{ij}. Figure 2.3 is a shematic of the formulas from Equations 2.4-2.7. will guide us through the derivation of the gradient for each weight which can be used in the backpropogation algorithm to find the optimal weights of the network [53].

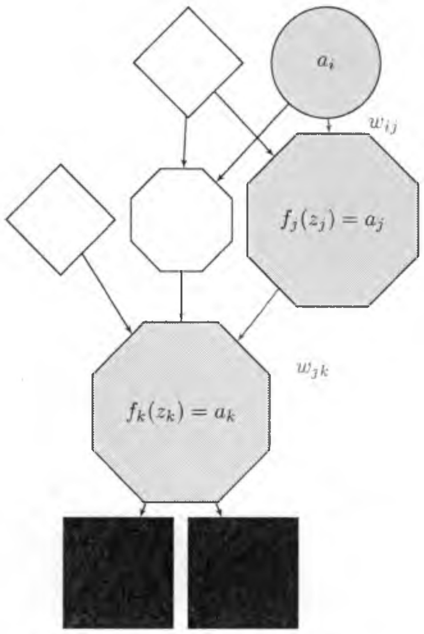


Figure 2.3: Schema of the neural network 2.2 to derive backpropogation algorithm to update the network weights.

Once we define the partial, one option is to use the regular gradient descent to update w_{ij}:

$$w_{ij} \longleftarrow w_{ij} - \alpha \frac{\partial J(W)}{\partial w_{ij}}$$

where $\alpha \in \mathbb{R}$ and is defined by the user and is known as the learning rate. From Figure 2.3 we can compute the gradient for an output layer weight as follows,

$$\begin{aligned}
\frac{\partial J(W)}{\partial w_{jk}} = \frac{\partial a_k/\partial w_{jk}}{a_k} &= \frac{1}{f_k(z_k)}\frac{\partial}{\partial w_{jk}}f(z_k) && (2.14)\\
&= \frac{1}{f_k(z_k)}f_k'(z_k)\frac{\partial}{\partial w_{jk}}z_k && (2.15)\\
&= \frac{1}{f_k(z_k)}f_k'(z_k)\frac{\partial}{\partial w_{jk}}(\sum_j f_j(z_j)w_{jk}+b_j) && (2.16)\\
&= \begin{cases} \frac{1}{f_k(z_k)}f_k'(z_k)a_j & \text{if } w_{jk} \text{ not bias}\\ \frac{1}{f_k(z_k)}f_k'(z_k) & \text{if } w_{jk} \text{ bias}\end{cases} && (2.17)
\end{aligned}$$

let $\delta_k = \frac{1}{f_k(z_k)}f_k'(z_k)$. Next we derive the gradient for an input layer weight as follows,

$$\begin{aligned}
\frac{\partial J(W)}{\partial w_{ij}} = \frac{\partial a_k/\partial w_{ij}}{a_k} &= \frac{1}{f_k(z_k)}\frac{\partial}{\partial w_{ij}}f_k(z_k) && (2.18)\\
&= \frac{1}{f_k(z_k)}f_k'(z_k)\frac{\partial}{\partial w_{ij}}z_k && (2.19)
\end{aligned}$$

Here, we use the chain rule to find $\frac{\partial}{\partial w_{ij}}z_k$,

$$\begin{aligned}
\frac{\partial}{\partial w_{ij}} z_k &= \frac{\partial z_k}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}} && (2.20)\\
&= w_{jk}\frac{\partial a_j}{\partial w_{ij}} && (2.21)\\
&= w_{jk}\frac{\partial f_j(z_j)}{\partial w_{ij}} && (2.22)\\
&= w_{jk} f_j'(z_j)\frac{\partial}{\partial w_{ij}} z_j && (2.23)\\
&= w_{jk} f_j'(z_j)\frac{\partial}{\partial w_{ij}} (\sum_i x_i w_{ij} + b_i) && (2.24)\\
&= \begin{cases} w_{jk} f_j'(z_j) x_i & \text{if } w_{ij} \text{ not bias} \\ w_{jk} f_j'(z_j) & \text{if } w_{ij} \text{ bias} \end{cases} && (2.25)
\end{aligned}$$

Putting this together we get,

$$\frac{\partial J(W)}{\partial w_{ij}} = \begin{cases} \delta_k w_{jk} f_j'(z_j) x_i & \text{if } w_{ij} \text{ not bias} \\ \delta_k w_{jk} f_j'(z_j) & \text{if } w_{ij} \text{ bias} \end{cases} \tag{2.26}$$

we can further let $\delta_j = \delta_k w_{jk} f_j'(z_k)$ for the simpler expression,

$$\frac{\partial J(W)}{\partial w_{ij}} = \begin{cases} \delta_j x_i & \text{if } w_{ij} \text{ not bias} \\ \delta_j & \text{if } w_{ij} \text{ bias} \end{cases} \tag{2.27}$$

we have therefore derived expressions for all weights of the network to define for any weight w_{ij} an update $w_{ij} \longleftarrow w_{ij} - \alpha\frac{\partial J(W)}{\partial w_{ij}}$

2.5 Error Backpropogaion Optimizations

There are two ways we can optimize the training of a neural networks. The first involves regularization methods when finding the optimal weights. The second, involves a faster method of convergence from the aforementioned newton method known as BFGS which is a quasi-newton method.

As for the first, we may alter the Equation 2.13 with an additional penalty term known as l_2 loss as follows [42],

$$J'(W) = \sum_i \log(f(x^{(i)}; W)^{y_i}(1 - f(x^{(i)}; W))^{1-y_i}) + \frac{\gamma}{2}\sum_{w \in W} w^2 \tag{2.28}$$

where γ is chosen by the user. This change is quite simple to incorporate into the partial derivates and changes our Newton update rule as follows,

$$w_{ij} \longleftarrow w_{ij} - \alpha\frac{\partial J'(W)}{\partial w_{ij}} = w_{ij} - \alpha\left(\frac{\partial J(W)}{\partial w_{ij}} - \gamma w_{ij}\right)$$

The second optimization is known as the BFGS update rule [41]. Recall, the hessian represents the matrix of second order derivatives with entries,

$$H_{ij} = \frac{\partial J(w)}{\partial w_i \partial w_j}$$

which reduces to taking the partials of the neural network f. We can then update the parameters W from f by moving in the direction d,

$$W_{i+1} \longleftarrow W_i - \alpha b = W_i - \alpha H_i^{-1} g_i$$

such that $b = H_i^{-1} g_i$ with $g_i \equiv \nabla J(W_i)$. Let $s_k = W_{k+1} - W_k$ and $y_k = g_{k+1} - g_k$. Additionally with $\rho_k = \frac{1}{y_k^T s_k}$ and letting the initial hessian H_0 be any positive semi-definite matrix (even the identity matrix). The algorithm to compute $\mathbf{H}_i^{-1} d$ for a given d can be found in Figure 2.4 [27].

From BFGS algorithm the limited memory L-BFGS algorithm is preferable if we employ larger neural networks and wish to limit memory usage [37]. The L-BFGS algorithm can be derived by limiting the indices in the for-loops and initial hessian $\mathbf{H}_0$ from 2.4. For example, the first for-loops would decrement down as in $i = n, \ldots, n-k$ with $k > 1$ and the second for-loop would increment as in $i = n-k, \ldots, n$. In addition, the hessian used to compute the center can be changed to depend on the last k s_i and y_i as in: $\mathbf{H}_0^{-1} = y_{k-1}^T s_{k-1} / y_{k-1}^T y_{k-1} \mathbf{I}$.

```
function BFGSDirection($\mathbf{H}_0^{-1}$, $\{s_k\}$, $\{y_k\}$, $d$)
    $r \leftarrow d$
    // Compute right product
    for $i = n, \ldots, 1$ do
        $\alpha_i \leftarrow \rho_i s_i^T r$
        $r \leftarrow r_i - \alpha_i y_i$
    // Compute center
    $r \leftarrow \mathbf{H}_0^{-1} r$
    // Compute left product
    for $i = 1, \ldots, n$ do
        $\beta \leftarrow \rho_i y_i^T r$
        $r \leftarrow r + (\alpha_{n-i+1} - \beta) s_i$
    return r
```

Figure 2.4: Pseudocode for BFGS update calculation.

2.6 Conclusions

In this chapter we introduced decision tree and neural networks architectures. We first saw how these methods are special types of graphs. We also discusssed how to learn the architectures for machine learning purposes. In this regard, loss functions are cornerstone.

Chapter 3

Neural Decision Trees

In this chapter, we combinine decision tree and neural network classifiers from chapter 2 into one structure known as a neural decision tree classifier. This novel structure enables the recursive application of neural network's trained with l_2 regularization effectively embedding variable selection. Additionally, the neural networks enable complex decision boundaries formed by linear combinations of the features to partition a feature space into regions designated to a unique label for the purpose of classification.

3.1 Introduction

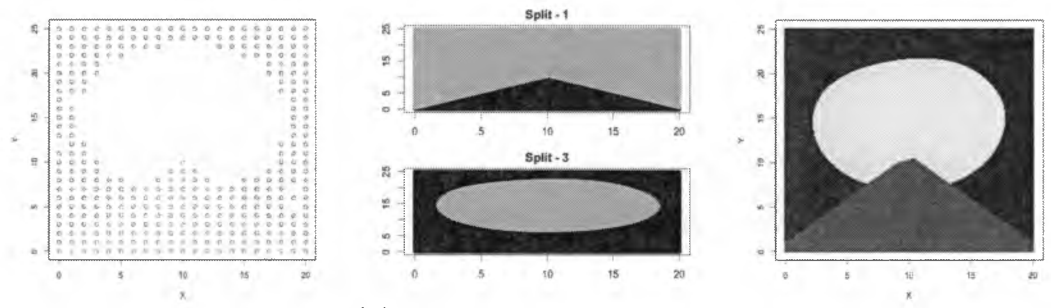
Machine learning techniques are a strong driving force in multidimensional classification problems. These problems require a balance of accuracy and interpretability. A classification method popularized by Quinlin and Breiman in the 1980's is known as deision trees [12]. One of the biggest strengths of *univariate* decision trees is the simplicity of learning orthogonal hyperplanes to partition a feature space [12]. But, this strength can also be a perceived achiles heel, requiring convoluted configurations to approximate simple patterns. *Multivariate* decision trees define simple and potentially linear patterns constructed with a linear combination of features [15]. Neural networks with one hidden layer give the ability to learn a linear combination of features and multi-layer perceptrons the ability to learn non-linear patterns [23]. Additionally, placing neural networks with one hidden layer in a tree-like fashion simulate multi-layer perceptrons [40]. Neural networks are a popular method used in image recognition and biological system modeling [36].

Neural networks popularity is in part because of their ability to disentangle covariance of input features [4] and can be trained with regularization methods to perform variable selection [47]. A desire to reduce dependence on hand-crafted features by embedding variable selection techniques and to partition a feature space achieved by (possibly) non-linear hypesurfaces, with a conceptually simple framework, has lead to neural decision tree architectures. The neural decision tree core consists of multiple neural networks connected in a tree-like fashion and have been successfully applied to language processing and comptuer vision problems [54, 26].

We introduce our neural decision tree implementation - utilizing single layer neural networks. The core of our method is dynamic methods which allow substitution of uniform nets (layed in a tree-like structure) with nets varying in hidden units informed by the complexity of data used in training. This method simultaneously achieves i) regularization and variable selection on various subsets of the training data, and ii) optiimal splitting and routing criterias for left and right children by class exchange methods. This provides an easy transition for those already familiar with decision tree tuning.

Standard neural networks fix the architecture and subsequently arrive at multi-class predictions. Neural decision trees achieve a more robust method using a divide and conquer strategy without pre-defining the depth of the tree. Heuristically, neural decision trees partition a data set starting from the easiest and ending with the hardest. An example of a neural decision tree's performance can be seen in Figure 3.1. Here we staged the performance of neural decision trees on a toy artificial data set with resemblance to problems in computer vision and image analysis. In particular, the neural decision tree is able to perform variable selection and exploit the conditional heirarchy of the pattern and with only two splits arrives at the fitted neural decision tree.

Experiments on a variety of image and pattern recognition problems show neural decision trees offer significant improvement over stand-alone decision trees. Further, neural decision trees provide a variety of presets to embed variable selection techniques by adapting the number of hidden units within a neural network and to use different regularization terms to control the edge weights within a network. These findings support that parallel tree and neural network mixtures produce a rich field for future research in representation, pattern, and discriminative learning.



(a) Artificial Training Pattern

(b) Two splits of the Neural Tree with 95% class purity.

(c) Neural Decision Tree Prediction

Figure 3.1: Results of training a neural tree on an artificial pattern.

3.1.1 Related Work

We focus on related approaches altering the split criteria in decision trees as well as previous work using one layer neural networks.

Stromberg, et al, were one of the first to describe the neural decision tree algorithm, successfully showing the algorithm is competitive with larger neural networks in speech recognition [54]. Similarly, Gelfand and Guo deomonstrated neural decision trees reduced tree complexity and maintained performance in waveform and digit recognition problems [26]. Molinari, et al. gave further evidence of neural decision trees comparable performance to neural networks on phonetic and synthetic datasets [40]. Typically, these approaches do not investigate performance using various dynamic methods to set the neural splits topology.

Neural trees can also be seen as a class of multivariate decision tree architecures. Apart from Quinlin and Breiman, multivariate decision trees have been studied since

1992. Brodley and Utgoff gave rigorous treatments to overfitting and efficient methods to learning linear combinations of features [57]. One special case in contrast to using neural networks is support vector machines (SVM). Bennett and Blue provide a rigorous approch to support vector trees using dual optimization techniques [9]. Alpaydin and Yildiz generalized upon multivariate to omnivariate methods by providing statistical measures of choosing which multivariate method to use at any split node [62].

Additionally, there are neural network architectures simulating tree structures. For example, there are perceptron trees [56].

3.2 Neural Decision Trees

A *neural decision tree* can either be seen as a decision tree with multivariate split criteria or a heirarchical neural network. This mixture of tree and neural networks introduces significant variations in methods and performance to stand-alone trees and neural networks. Positive impacts can be seen in the final classifier with negative impacts on training time incurred by repeated subroutines. The inference procedure in neural decision trees is described in the next section and coincides with finding the optimal splitting pair (f, θ) from from chapter 2. The learning phase however, is different and will be addressed in Sect 3.3.

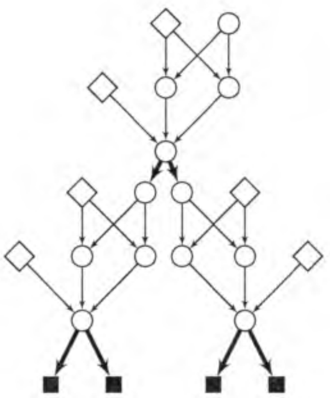


Figure 3.2: Neural decision tree schema. Neural networks with 1-2-1 architecture and one neural split, and four children nodes.

3.2.1 Inference

Inference in a neural decision tree is similar to a decision tree. A neural decision tree, is a learned tree $t : \mathcal{X} \longrightarrow \mathcal{Y}$ with $\mathcal{X}$ a feature space and $\mathcal{Y}$ a set of classes, that classifies a sample $x \in \mathcal{X}$ by routing it from the root to a leaf node. A leaf node maintains a label which is assigned to an arriving x. The collection of *leafs* partition the space $\mathcal{X}$ into regions labeled by elements of $\mathcal{Y}$. The computations used to route samples form $\mathcal{X}$ at internal nodes to arrive at the predicted classification assigned by leaves is the defining characteristic of neural decision trees.

Figure 3.2 is a schematic of a trained neural decision tree. The neural networks anchor as split points our routing gates. Recall, a tree is also a set of learned functions and cutoff pairs $(f_1, \theta_1), \ldots, (f_m, \theta_m)$. These pairs are the basis of a split function. Repeated application of a sequence of pairs (f_i, θ_i) with comparisons

$f_i(x) < \theta_i$ for an $x \in \mathcal{X}$ results in routing x through a tree. When the f_i are neural networks the internal node are said to be routing samples by *neural split* functions. These pairs split a training set $S = \{(x, y) : x \in \mathcal{X}\ y \in \mathcal{Y}\}$ into two subsets S_L and S_R. These subsets are formed with the following properties $S_L = \{(x, y) : f(x) < \theta\ (x, y) \in S\}$ and $S_R = \{(x, y) : f(x) \geq \theta\ (x, y) \in S\}$.

3.2.2 Neural Split Function

A *neural network* or *multi-layer perceptron* used in a neural decision tree is a parameterized non-linear mapping $f : \mathcal{X} \longrightarrow [0, 1]$. The computational structure is similar to bipartite graphs arranged in layers. An neural network decomposes into *input layer* connected to the sample from $\mathcal{X}$, k *hidden layers* and one *output layer.* Nodes in a layer are called *perceptrons.* Edges are directed and connect adjacent nodes in layers with real values called *weights. Weights* are learned and are the result of training.

We can construct an neural network from the familiar logistic function. Recall the logistic function in two variables $\sigma(x, y) = [1 + e^{-(b+w_1x+w_2y)}]^{-1}$. Figure 3.3 depicts the situation. Given an appropriate training set we can *fit* (b, w_1, w_2) as in logistic regression. Adding an input layer of two nodes, bias, and logistic terms as in figure 3.3b we arrive at a simple neural network with 1 hidden layer. Notice, a bias term is necessary for similar reasons an intercept term is needed in logistic regression.

$$\begin{tikzcd}
1 \arrow[d, dash] & x \arrow[d, dash] & y \arrow[d, dash] \\
b \arrow[dr, dash] & w_1 \arrow[d, dash] & w_2 \arrow[dl, dash] \\
& \sigma &
\end{tikzcd}$$

(a) Schema depicting $\sigma((1, x, y)^T(b, w_1, w_2))$

$$\begin{tikzcd}
& 1 \arrow[dr] \arrow[drrr] & & x \arrow[dl] \arrow[dr] & & y \arrow[dlll] \arrow[dl] \\
1 \arrow[drr] & & \sigma_1 \arrow[d] & & \sigma_2 \arrow[dll] & \\
& & \sigma_3 & & &
\end{tikzcd}$$

(b) Neural network 2-2-1 architecture.

Figure 3.3: Buiding up a neural network from familiar familiar linear model techniques.

Let i be the superscript of the layer of the neural network with $b^{(i)}$ the vector of intercepts and $W^{(i)}$ the matrix of weights with row entries the jth neuron input weights. Then, the neural network from figure 3.3b has the form,

$$\sigma(W^{(2)}\sigma(W^{(1)}x + b^{(1)}) + b^{(2)})$$

where σ applied to a vector distributes as in $\sigma(a, b) = (\sigma(a), \sigma(b))$. Therefore, the whole neural network can be defined as a function $f(\cdot; W) : \mathcal{X} \longrightarrow \mathbb{R}$ obtained by the composition of layer functions.

3.2.3 Training Neural Networks

Given a node in a treet Let $x^{(i)} \in \mathcal{X}$ with label $y^{(i)} \in \mathcal{Y} = \{0, 1\}$. We define the cross-entropy loss

$$J(W) = \sum_i \log(f(x^{(i)}; W)^{y_i}(1 - f(x^{(i)}; W))^{1-y_i}) + \frac{\gamma}{2} \sum_{w \in W} w^2$$

we wish to minimize this value by finding the optimal W denoting all the weights in the neural network f. One method is to use BFGS to train the neural network f by updating the weights W,

$$W_{k+1} \longleftarrow W_k - \alpha H_k^{-1} \nabla g_k$$

with $g_i \equiv \nabla J(W_i)$ and $H_{ij} = \frac{\partial J(w)}{\partial w_i \partial w_j}$ and $\alpha \in \mathbb{R}$ the learning rate chosen by the user. With this rule we can then use *backpropogation* to iteratively update weights from successive layers in a neural network. The backpropogation algorithm details can be found in Chapter 2.

We reduce multi-class problems to binary classification problems in a neural network by using class exchange training strategy. Suppose for example, that $\mathcal{Y} = \{c_1, \ldots, c_K\}$. That is, $\mathcal{Y}$ has K distinct classes. Then, the class exchange method exhausts the $\binom{K}{2}$ neural networks distinguishing 2 disjoint classes with each made of elements from K. Details of the training of a neural network using class exchange methods can be found in Gelfand, et al [26].

Essentially, one randomly splits $\mathcal{Y} = A \cup B$ such that A and B are disjoint and train the neural network to distinguish these two classes on the training set. Then, supposing there is training error re-run all the training data through the trained neural network and collect the vectors into A' and B'. Compute the weighted average of the entropy of A' and B'. Randomly switch one c_i from A to B and retrain the neural network and compute the weighted average of the entropy of the subsequent A' and B'. Continue this method until the smallest weighted average entropy is found or you run out of c_i to switch.

The neural networks are trained using a maximum number of iterations of 100 unless stated otherwise.

3.3 Learning in Neural Decision Trees

We have begun to describe traditional tree building popularized by Breiman and Quinlin [49, 14]. The method follows an intuitive induction or divide and conquer strategy.

For example, let S be a training set at node m of a tree. We then train a neural network $f : \mathbb{R}^d \longrightarrow \{0, 1\}$ and find a set of weights W, defining the neural network, to partition the training set into two classes. We use a multi-class exchange method to reduce multi-classes to binary (Sec 4.2.3). This neural network acts as a gating function to form two new subsets. That is, the neural network f partitions S: $S_L = \{(x, y); f(x; W) < .5\}$ and $S_R = \{(x, y) : f(x; W) \geq .5\}$.

We repeat the process above on subsequent children S_L and S_R resulting from each neural split. The process is guaranteed to conclude for a few reasons. The first, is when then is only one element in each S_L, S_R or when each child has only one class represented, resulting in no reason to split. Lastly, there may be a variety of classes but a trained neural network may be unable to be trained and discriminate between the two, resulting in no possible split. This later result, for the simple reasons that one trades-off one neural network training time for tree size and potentially faster tree building.

3.4 Neural Network Topology Selection

Figure 3.2 outlines a neural decision tree with a parent neural network with 2 hidden units and two children with the same number of hidden units. A neural decision tree usually has a fixed input layer equal to the number of components of the feature space. A variety of neural decision tree architectures can be derived by altering the number of hidden units between parent and children neural networks. In addition, a particularly interesting topology was the use of preseting the weights of the neural networks using a method of random projections prior to training.

For example, in contrast to fixing the number of hidden units one might choose k% of the number of input units: here we tried $k \in (50, 200)$ percent. Alternatively, we set the number of units to be m/d where m is the number of training instances d the number of features. The m/d-features assumes the larger the number of instances to features, more complex neural networks are required. Lastly, we choose the number of units to reflect the percentage of the number of original classes remaining. In particular, suppose at a neural split 50% of the original number of classes remain, then, we chose a $k + 50\%$ of the features. Here we choose some $k = 0, 100, or 200$.

Lastly, we performed a technique by fixing a network architecture and hot-starting the neural networks. This technique utilized the weak learning technique of random projections attempting to randomly fit a hyperplane with margin l between classes. Given our multi-class exchange methods reduce our multi-class to binary-classes we randomly chose some s hyperplanes equal to fill the number of hidden units and output units of our nets to seperate two classes.

3.5 Neural Decision Tree Experiments

In this section we report experimental results on two datasets and an artificial pattern. The two datasets are the Iris and Image Segmentation. With the datasets all our experiments compare to a baseline conventional decision tree implemented by the rpart package in R. For all our provided methods, we train trees on all features from the datasets. We use 5-fold vaidation to derivde all performance measures.

For the decision tree gini impurity and no pruning methods are used. For our neural decision tree we compare the effect of 4 different variants. We restrict our variants to using 1 layer perceptrons. The actual topology selection is compared for fixed and dynamic neural splits. The four neural decision trees are i) NT_5 every neural split is restricted to 5 units in the hidden layer, ii) NT_{col} every neural split has the same number of hidden units as the number of columns or number of input units, iii) $\mathrm{NT}_{col\cdot 1/2}$ ever neural split has half the number of hidden units as the number of input units, and iv) $\mathrm{NT}_{\max\{m/n,5\}}$ neural splits have either the number of rows divided by the number of columns of the training data at the split, or at minimum 5 units in the hidden layer, v) $\mathrm{NT}_{\pi(5)}$ neural splits have 5 hidden units but weights preset by random projection hyperplanes, and vi) $\mathrm{NT}_{\pi(col)}$ neural splits have a number of hidden units equal to the *col* input dimension but weights presets by random projection hyperplanes. We report three types of scores, i) **Global** the percentage of all correctly classified instances, ii) **Class-Average** known as *recall* derived from the calculation: $\frac{TP}{TP+FN}$ and, iii) **Jaccard** defined as $\frac{TP}{TP+FP+FN}$ (where TP, FP, FN stand for true positive, false positive, and false negative, respectively. Lastly, we report the average number of splits used in a tree.

3.5.1 Iris Dataset

To assess the behavior of the neural decision tree with respect to a linearly seperable dataset we use this the Iris dataset. The iris dataset has a 4-dimensional feature space and a classification space of 3 labels. The labels are equally reprented at approximately 33.3% making the dataset balanced.

On average using all features we have 2.0/0.0/-.2% change on Global/Class-Average/Jaccard scores compared to the baseline decision tree, when using the NT_5,

Methods	Iris				Image-Seg			
	Global	**Class-Avg**	**Jaccard**	**Splits**	**Global**	**Class-Avg**	**Jaccard**	**Splits**
NT_5	96.7 ± 4.0	51.2 ± 48.3	49.7 ± 47.0	2.0 ± 0.0	81.3 ± 7.2	51.6 ± 33.3	46.8 ± 27.4	7.0 ± 0.7
NT_{col}	96.7 ± 3.3	50.1 ± 49.1	49.7 ± 46.7	2.0 ± 0.0	82.9 ± 4.4	49.7 ± 35.1	46.6 ± 28.1	7.4 ± 1.5
$\text{NT}_{col*1/2}$	95.4 ± 3.8	51.7 ± 46.4	49.5 ± 44.6	2.0 ± 0.0	80.0 ± 7.6	50.4 ± 32.0	46.4 ± 25.4	6.2 ± 0.4
$\text{NT}_{\max\{m/n,5\}}$	96.0 ± 4.4	50.2 ± 48.5	48.4 ± 46.9	2.0 ± 0.0	84.3 ± 7.0	52.5 ± 33.9	49.2 ± 28.1	7.8 ± 1.3
Tree Baseline	94.0 ± 3.7	51.0 ± 45.5	49.4 ± 42.3	2.0 ± 0.0	84.8 ± 4.3	51.7 ± 34.1	49.5 ± 27.8	8.0 ± 0.7
$\text{NT}_{\pi(col)}$	96.6 ± 3.3	51.1 ± 48.6	49.6 ± 47.2	2.0 ± 0.0	84.2 ± 3.9	51.4 ± 36.1	47.3 ± 30.1	7.8 ± 1.1
$\text{NT}_{\pi(5)}$	97.3 ± 2.9	51.0 ± 49.1	49.7 ± 47.8	2.0 ± 0.0	83.8 ± 6.3	51.6 ± 35.1	48.5 ± 28.4	9.6 ± 1.3

Table 3.1: Neural tree Experiemntal results with comparisons to tree baseline. All number reported in (%).

$\text{NT}_{col \cdot 1/2}$, $\text{NT}_{\max\{m/n,5\}}$, indicating that more complex models can provide some gain but have a potential to overfit the data. One of the better performing neural decision trees is the NT_{col} with the 2.7/-.9/.3% change compared to the baseline decision tree. The various methods can be compared in Table 4.2.

When comparing our best global score $\text{NT}_{\pi(col)} \approx 97.3\%$ we find similar performance among all the proposed neural decision tree frameworks. But that the hot start did provide a potential increase in performance. This suggest the neural decision trees can have a positive performance gain on datasets similar to Iris.

3.5.2 Image Segmentation Dataset

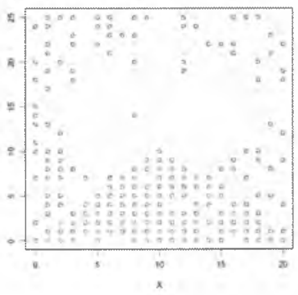
To asses the behavior of the neural decision tree with respect to a larger multi-class problem with applications to computer vision and image analys we use the UCI image segmentation data set of 1990. The image segmentation dataset has a 19-dimensional feature space and a classification space of 7 labels. The labels are equally represented at approximately 14.3% making the dataset balanced.

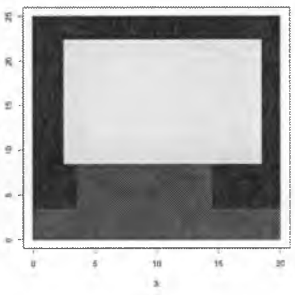
On average using all features we have -3.4/-1.1/.1% change on Global/Class-Average/Jaccard scores compared to the baseline decision tree, when using the NT_5, NT_{col}, and $\text{NT}_{col \cdot 1/2}$. indicating that more complex models may not gain, may overfit, or may need further neural tuning. One of the better performing neural decision trees is the $\text{NT}_{\max\{m/n,5\}}$ had a -.5/.8/-.3% difference in scores from the baseline decision tree. The various methods can be compared in Table 4.2.

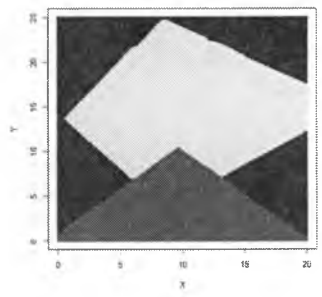
When comparing our best global score $\text{NT}_{\max\{m/n,5\}} \approx 84.3\%$ we find the $\text{NT}_{\pi(col)}$ has similar performance. The result is suggestive that more informed models or perhaps a better trained neural network can improve the overall accuracy. The conclusion of the experiment suggests neural decision trees are competitive with decision trees on datasets similar to the image segmentation and leave room for improvement.

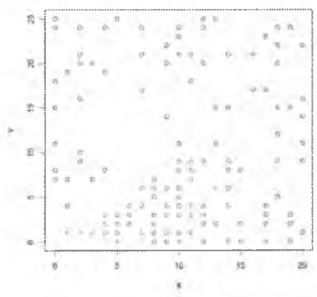
3.5.3 Artificial Pattern

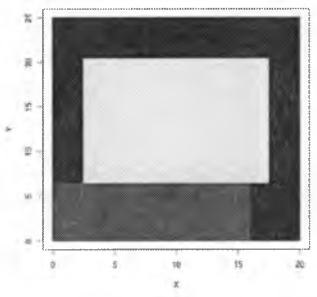
To asses the behavior of the neural decision tree with respect to pattern discrimination with overlapping class distributions we constructed an artificial pattern. The pattern has two dominant shapes: circle and triangle, overlapping, atop a background color. The idea is to train the neural decision tree and decision tree on the pattern. Then make predictions over the entire plane and compare the resulting predictions.


(a) Artificial Pattern 5% reduced sampling.


(b) The Decision Tree Predicion


(c) Neural Decision Tree Prediction


(d) Artificial Pattern 25% reduced sampling.


(e) The Decision Tree Prediction.

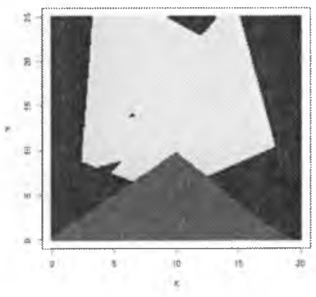
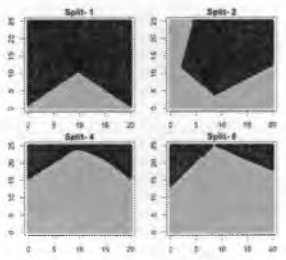

(f) Neural Decision Tree Prediction.

Figure 3.4: Comparitive performance of the neural decision tree and decision tree on an artificial pattern with 5% and 25% reduced sampling. (b) and (c) predictions from pattern (a). (e) and (f) predictions from (d).

The pattern was consructed three times. Each time uniform samples were placed in the corresponding regions outlined by the shapes to fill 100%, 95%, and 75%, respectively. To reduce the amount of neural decision tree frameworks considered we choose the NT_5 as it behaved relatively stable in tests. Figure 3.1 demonstrated


(a) Neural decision tree splits producing (e) predictions.

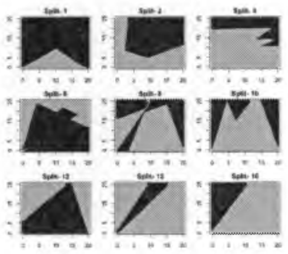

(b) Neural decision tree splits producing (f) predictions.

Figure 3.5: Black/Gray diagrams depicting left/right splits within neural decision trees trained on the artificial patterns of Figure 1.4.

the training and performance of the NT_5 model with the two-class used to split at each internal node of the tree. Similarly, Figures 3.4a and 3.4d show the trianing pattern and prediction from NT_5 across the plane.

We also inspect the splits produced by each of the fit NT_5. Figure 3.5 compares the patterns most discriminative and achieving the highest purity gain from the parent neural split. For example, we see in subfigure (a) that the NT_5 characterized the red triangle most at the first split. At the second split, we see that the yellow circle was then found along with the background blue pattern.

3.6 Conclusion

In this work we have introduced neural decision trees which is a mixture of decision tree and neural network classifer. It is based on techniques popularized in the decision trees. We proposed a variety of performance enhancements to the neural decision tree architectures restricted to using a single layer. Therefore, we considered various techniques of adjusting and setting the size of the hidden units in a neural split. The performance of neural decision trees has shown to be similar and if not better than decision trees offer future changes in architecture that might be exploited. While the design of hand-crafted features and variable selection is a typical problem in general, neural decision trees give a systematic way of potentially utilizing both the advantageous properties of neural networks for this reason, while also using the heirarchical routing of classes done in decision tree learning. The variety of neural decision tree architectures investigated yielded comparable results to those given by decision trees in the Iris and Image Segmentation datasets. Additionally, we found a neural decision trees are capable of reporting intermediate pattern discrimination that can be visualized and help in feature extraction and identifying variable importance.

Chapter 4

Neural Decision Forests

In this chapter, we construct an ensemble of neural decision tree classifiers from chapter 3 by bootstrap aggregating. We detail the main concepts in ensemble learning and nuances specific to neural decision forest constructions. With our original neural decision forest we are capable of constructing a set of unique heirarchical neural networks capable of variable selection and optimal routing. Performance on three datasets and an artifical pattern suggest the neural decision forest has similar performance to the random forest algorithm with potentially distinct advantages.

4.1 Introduction

Ensemble methods in machine learning are popular since they may perform better than any one single classifier [20]. An ensemble, in general, is more accurate than a single classifier when the members have low bias and high variance [28]. The intuition, is that a machine throwing darts may be good with specific darts and worse for others. Introducing more machines, with similar properties, and voting for the most popular, leads to an incrased overall accuracy. An ensemble of decision trees was one of the earliest ensemble methods. The random forest algorithm, in particular, is an ensemble of decision trees originally studied by Breiman [14, 13]. Neural network ensembles, alternatively, were studied prior to the seminal work on random forest by Hansen and Salamon [28].

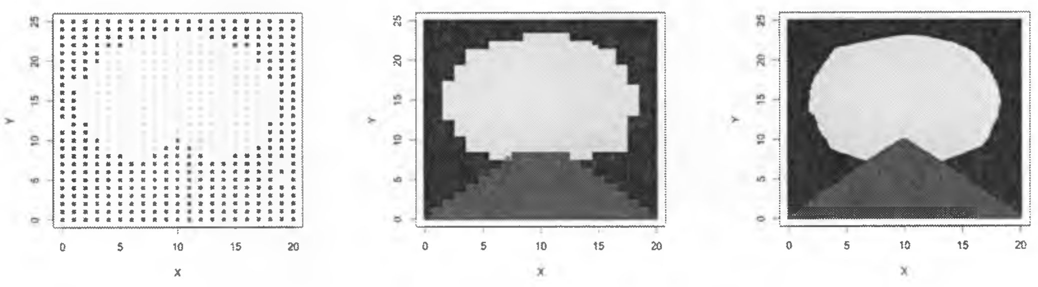
The random forest algorithm has proven to be an efficient techinqiue widely used for image analysis problems [24, 38]. Though individual features and random addition of features are used in the training, these methods are not the most general ways to form linear combinations of features. Similarly, ensembles of neural networks have proven to be efficient techniques in image classification problems [28]. But, some networks need heirarchical structure.

A desire to generalize the random forest algorithm and inject random heirarchical structure into neural networks has lead to our original neural decision forest implementation. We are indebted to the previous work by Bulo and Kontschieder [51]. But our methods vary in that we explore the the limits of using single layer neural networks and the important porperties of bagging, class exchange, and fitting neural trees until leaves are entirely pure. Bagging, is of particular interest within an neural decision forest, since it has been shown to have positive gains in decision tree and neural network ensembles [43].

The core of our method is randomized neural networks with 1-hidden layer (rNet) and the randomization of features which are used as input at all neural splits of the individual neural trees. The randomization of features used in the neural networks is a generalization of the techniques used in the random forest algorithm because the number of units in a given neural network may also be chosen randomly. Bulo and Kontschieder, termed these neural networks rMLPs because they mainly explored varying the number of hidden layers [51]. We use the term rNets to highlight our use of neural networks with 1 hidden layer as opposed to 1 or more. Our methods of bagging and using randomized neural networks simultaneously achieves i) A new assortment of heierarchical neural networks, and ii) A partitioning of the feature space using linear hypersurfaces (i.e. not necessarily a hyperplane).

While, it is known that neural decision trees generalize CART decision trees by using multivariate splitting criterias it is not entirely known how neural decision trees are an improvement. One anecdote mentioned by researches is that neural decision trees partition a data set starting from the easiest and ending with the hardest classes [54]. It is not clear this behavior remains within an neural decision forest or even what the differences may be more generally to the random forest algorithm.

We staged the performance of an neural decision forest on a toy artificial data set with resemblance to problems in computer vision and image analysis to compare with random forests. Figure 4.1 shows a given pattern, which having trained the neural



(a) Artificial Training Pattern (b) Random Forest Predictions (c) Neural Decision Forest Predictions

Figure 4.1: Comparison of hyper-surface configurations of Random Forest and Neural Forest classifiers trained on an articial pattern of 3 classes.

decision forest and random forest algorithms, then predicted over the surrounding 2D grid. We leave it to the reader to decide, but the results are strikingly different in the various borders which are approximated by either algorithm.

Experiments on a variety of image and pattern recognition problems show neural decision forests improve or are comparable in performance to random forests. This is no small feat. Further, neural decision forests provide a variety of presets to perform variable selection at each rNet. In contrast to the random forest algorithm, the neural decision forest enables the inspection at each rNet of the pre and post features extracted. These findings support that ensemble decision tree and neural network mixtures produce a rich field for future research in representation, pattern, and discriminative learning.

4.1.1 Related Work

We mainly focus on related works in decision tree ensembles and multivariate splitting criterias.

There is a distinction in the literature between ensemble and combining decision trees. Both tasks wish to partition a feature space but use different methods. On the one hand, combining decision trees is the task of combining splitting rules from a set of trees, to arrive at one tree [48]. On the otherhand, an ensemble of trees assumes each tree is different and does not combine but maintains a collection of different trees [13]. An ensemble of trees partitions the feature space with potentially overlapping regions and therefore defining a probability that a given region of a feature space should be assigned a specific label. Additionally, ensembles break ties with a majority vote [3].

Stromberg, et al, were one of the first to describe the neural tree algorithm, successfully showing the algorithm is competitive with larger MLPs in speech recognition [54]. Similarly, Gelfand and Guo deomonstrated neural trees reduced tree complexity and maintained performance in waveform and digit recognition problems [26]. Molinari, et al. gave further evidence of neural decision trees comparable performance to MLPs on phonetic and synthetic datasets [40]. Typically, these approaches do not investigate performance using various dynamic methods to set the neural splits topology.

Neural decision trees can also be seen as a class of multivariate decision tree architecures. Apart from Quinlin and Breiman, multivariate decision trees have been studied since 1992. Brodley and Utgoff gave rigorous treatments to overfitting and efficient methods to learning linear combinations of features [57]. One special case in contrast to using MLPs is Support Vector Machines (SVM). Bennett and Blue provide a rigorous approch to support vector trees using dual optimization techniques [9]. Alpaydin and Yildiz generalized upon multivaraite methods by providing statistical measures of choosing which multivariate method to use at any split node in a tree [62].

Last and most importantly is the work of Bulo and Kontschieder outlining the neural decision forest algorithm within a Bayesian framework and the introduction of rMLPs training [51]. As mentioned their work mainly focuses on variations of rMLPs but do not inspect performance with respect to various tree sizes and candidate splitting techniques.

4.2 Neural Decision Forests

A *neural decision forest* is an ensemble of neural decision trees. Each neural decision tree introduces randomization by dynamically changing the neural network architecture at each neural split in the tree. These neural splits utilize *randomized neural networks* (rNet) to highlight the use of one hidden layer neural networks. Injecting each neural decision tree with dynamic neural splits introduces significant and positive imporovements over standalone neural decision trees in classification problems. Neural decision forests build upon the familiar concepts of routing vectors, partitioning feature spaces, and majority voting, found in the random forest algorithm. The techniques used to induce randomization is different, however.

4.2.1 Inference

Inference in neural decision forest is similar to that of random forests. Let $\mathcal{X}$ be a feature space and $\mathcal{Y}$ a set of classes with $t : \mathcal{X} \longrightarrow \mathcal{Y}$ a neural decision tree. A neural decision forest is a learned collection of neural decision tree classifiers $\{t(x, \Omega_k), k = 1, \dots\}$ with $x \in \mathcal{X}$ where the $\{\Omega_k\}$ are indendent identically distirbuted random vectors and each tree casts a unit vote for the most popular class for input x. Ω_k randomly samples from a training set S.

At the individual tree level, $t(x, \Omega_k)$, the inference procedure is simply a neural decision tree restricted to a sample from S with additional randomization at each neural split. A neural decision tree classifies a sample $x \in \mathcal{X}$ by routing it from the root to a leaf node. A leaf node maintains a label which is assigned to an arriving x. The collection of *leafs* partition the space $\mathcal{X}$ into regions labeled by elements of $\mathcal{Y}$. While decision trees partition $\mathcal{X}$ into hypercubes labeled uniquely from elements in $\mathcal{Y}$, neural decision trees parition $\mathcal{X}$ into possibly into linear hyper-regions. With a collection of trees the random forest algorithm defines a probability distribution based on the union and intersection of hypercubes of the feature space and neural decision forests a probability distribution on the union and intersection of hyper-

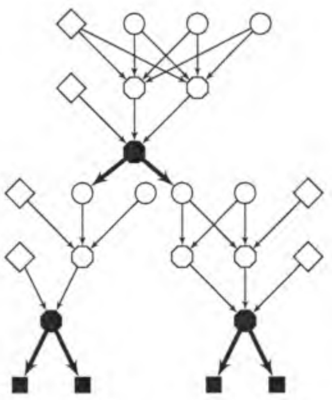


Figure 4.2: One possible tree within a neural decision forest. First net has 3-2-1 architecture and children nets have 1-2-1 and 1-3-1 architectures.

regions. Given $x \in \mathcal{X}$ a final classification is arrived by the equation,

$$\arg\max_{c \in \mathcal{Y}} \frac{1}{N} \sum_{k=1}^{N} \chi_{c=t(x,\Omega_k)} \tag{4.1}$$

The computations used to route samples form $\mathcal{X}$ at internal nodes to arrive at the predicted classification assigned by leaves, is the defining characteristic of neural decision trees. Refer to chapter 3 for more details on neural decision tree construciton details.

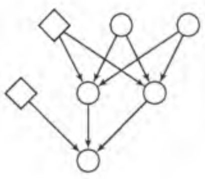
Figure 4.2 is a schematic of one neural decision tree in a trained neural decision forest. The neural networks anchor as split points. Recall, a tree is also a set of learned functions and cutoff pairs $(f_1(x : \gamma_1), \theta_1), \ldots, (f_m(x; \gamma_m, \theta_m))$. Just as in a neural decision tree these pairs are the basis of a split function, the difference is that γ_i are independent identically distirbuted random variables producing the number input and hidden layer neurons. Repeated application of a sequence of pairs $(f_i(x;\gamma_i),\theta_i)$ with comparisons $f_i(x;\gamma_i) < \theta_i$ for an $x \in \mathcal{X}$ results in routing x through a tree. These pairs split a training set $S = \{(x,y) : x \in \mathcal{X}\ y \in \mathcal{Y}\}$ into two subsets S_L and S_R. These subsets are formed with the following properties $S_L = \{(x,y) : f(x;\gamma) < \theta\ (x,y) \in S\}$ and $S_R = \{(x,y) : f(x;\gamma) \geq \theta\ (x,y) \in S\}$.

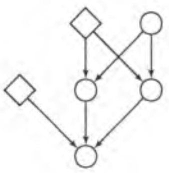
4.2.2 Neural Split Function

A neural split in a neural decision tree from a neural decision forest is produced by a *randomized neural network* (rNet). An rNet is an neural network with input, hidden, and output layers with a chosen activation function. rNets are *randomized* neural networks by the the fact that the number of units in each layer are chosen randomly. The exception is the output layer within a binary tree where we restrict the output layer to one unit. See section 4.3 for more details on the basics of neural network topologies in a neural decision tree.

More formally, let S be a training set at node m of a neural decision tree. With $f : \mathcal{X} \longrightarrow \{0,1\}$ a neural network trained to distinguish two classes. Let $f(x;\gamma,W)$ represent the training rNet with edge weights W and γ is a random vector $\gamma = (A,B)$ such that $A \sim U(1,|\mathcal{X}|)$ and $B \sim U(1,|\mathcal{X}| * 2)$ are uniformly distributed random variables. A denotes the number of input units, B the number of hidden units in a rNet which then define the number of edge weights $|W| = |A||B| + |B|$.

Figure 4.3 is three different rNet architectures. Recall, the nodes in a horizontal


(a) 2-2-1 Net


(b) 1-2-1 Net

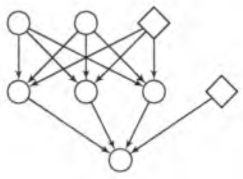

(c) 2-3-1 Net

Figure 4.3: Reference of different net architectures configurable based on class distribution and set size needing splitting.

level are considered layers. Notice, each layer may have a randomly chosen number of units. The only restriction is that the number of units must permit the neural network to find an optimal set of weights W given a sample Ω from training set S.

Letting a rNet be the map $f(\cdot;\gamma, W) : \mathcal{X} \longrightarrow \mathbb{R}$. with W the edge weights, rNets have the nice mathematical structure of neural networks. Let i be the superscript of the layer of the rNet. With $b^{(i)}$ the vector of intercepts, $W^{(i)}$ the matrix of weights with row entries the jth neuron input weights. Then, all the neural networks from Figure 4.3 have the form,

$$\sigma(W^{(2)}\sigma(W^{(1)}x + b^{(1)}) + b^{(2)})$$

The differencences in each subfigure from Figure 4.3 can be seen in the dimension of the weight matrices defining each layer.

4.2.3 Training rNets

Given a node in a tree$t$ Let $x^{(i)} \in \mathcal{X}$ with label $y^{(i)} \in \mathcal{Y} = \{0, 1\}$. We define the cross-entropy loss

$$J(W) = \sum_i \log(f(x^{(i)}; W)^{y_i}(1 - f(x^{(i)}; W))^{1-y_i}) + \frac{\gamma}{2} \sum_{w \in W} w^2$$

we wish to minimize this value by finding the optimal W denoting all the weights in the neural network f. One method is to use BFGS to train the neural network f by updating the weights W,

$$W_{k+1} \longleftarrow W_k - \alpha H_k^{-1} \nabla g_k$$

with $g_i \equiv \nabla J(W_i)$ and $H_{ij} = \frac{\partial J(w)}{\partial w_i \partial w_j}$ and $\alpha \in \mathbb{R}$ the learning rate chosen by the user. With this rule we can then use *backpropogation* to iteratively update weights from successive layers in a neural network. The backpropogation algorithm details can be found any introductory neural network text.

We reduce multi-class problems to binary classification problems in a NET by using class exchange training of Nets. Suppose for example, that $\mathcal{Y} = \{c_1, \ldots, c_K\}$. That is, $\mathcal{Y}$ has K distinct classes. Then, the class exchange method exhausts the $\binom{K}{2}$ Nets distinguishing 2 disjoint classes each made of elements from K. Details of the training a neural network using class exchange methods can be found in Gelfand, et al [26].

4.3 Neural Decision Tree and rNet Topology

The neural decision trees constructed within an neural decision forest are unique in that there is sufficient randomization to ensure no two trees are identical. Comparing Figure 4.2 to Figure 3.2 one can see the neural decision trees within a forest are different than in a stand-alone neural decision trees. The neural decision tree schema in a neural decision forest is driven by the rNet neural splits. On the one hand, at the global tree level, it appears as though a variety of neural networks were randomly placed to serve as the splitting functions. On the other-hand, we see locally that the neural splits trained on a subset of S are *randomized* by the random selection of features and number of layers and units. This leads to the rNet classification.

A variety of methods are employed to randomly select features/input units and hidden layer units. For the feature/input units two methods of randomization are compared. The first, given a feature space of dimension d a number of features $k << d$ are chosen (typically d/2). The second, we randomly select a feature then randomly select a band-width and picking adjacent features. For the hidden number of units we investigate two variations: i) selecting the number of hidden hidden units from the uniform distribution between 1 and the number of input units, and ii) selecting the number of hidden units within a p percentage around the number input units u from the previous step: $(u(1-p), u(1+p))$.

In practice, neural networks are commonly seen to overfit training data, that is, they fit the training data well but do not generalize to new data. The randomization techniques introduced here may be seen to act as a regularization technique helping prevent overfitting. In addition, we see that the typical training of an rNet is not trained to disdcriminate classes and instead two aggregated subsets containing a collection of classes. For example, assuming the neural decision forest is trained to discriminate 3 classes, then at least one rNet is not trained simply to discriminate class labels and is instead trained to discriminate two sets where at least one contains two class labels. It might be worth noting, that since a rNet is within a succession of rNets (typically) it is not of immediate importance it learns to discriminate all classes since that is also the job of successive rNets.

4.4 Learning in Neural Decision Forests

In chapter 3 we discussed neural decision tree learning, here we focus on how an ensemble of neural decision trees reduces prediction errors and how injecting randomization in the learning phase achieves this.

In fact, we have begun to describe traditional forest building popularized by Breiman [14]. The characteristic of tree ensembles is bagging and random splits [13]. Bagging, for exmaple, samples a proportion of vectors from a training set S and constructs a tree from this sample and then the process is repeated as desired. A popular choice for sampling proportion is 2/3. While in random splits a number of features are chosen, usually much smaller than the total number of features, and used as candidates in replace of the full feature space. In the case of neural decision trees we generalize these splits with rNets. The combination of bagging

Let $f(x;\Theta,\Omega)$ be a random forest (possibly neural) constructed with bagging and random splits (rNets). We we wish to approximate the true f from a training set $\hat{f}$. The prediction error reduces to,

$$\mathrm{Err}(x) = \left(E[\hat{f}(x)] - f(x)\right)^2 + E\left[\hat{f}(x) - E[\hat{f}(x)]\right]^2 + \sigma_\epsilon^2 \tag{4.2}$$

where σ_ϵ^2 is the irreducible error [29]. Each tree in the ensemble has a low bias as it fits the training data perfectly. Therefore, the first term in the above summation is zero. On the other hand, each tree has a large variance, the second summation term. Recall equation 4.1 and replace this equation for f in equation 4.2. Letting $N \longrightarrow \infty$ we see only σ_ϵ^2 remains.

4.5 Neural Decision Forest Experiments

In this section we report experimental results on three datasets and an articial pattern. The datasets are the Iris, Image Segmentation, and Handwritten Digits datasets from the popular UCI repository [35]. With the datasets all our experiments compare to a baseline conventional random forest implemented by the randomForest package in R. Within a random forest we preset the number of trees to 500. We implemented an original neural decision forest in R. Within a neural decision forest we preset the number of trees to 100 on the Iris and Image Segmentation datasets and 200 trees on Handwritten Digits. We use 5-fold cross vaidation to derive all performance measures.

As is typical, all trees were constructed until each leaf was pure (with only samples from one class represented) with variations of either information or gini gain. For the neural decision forest we compare the effect of a variety of randomization techniques at neural splits. We induce randomization at the input layer using either i) a number of random features $k << d$ (aka dots), or ii) a random number of adjacent features (aka bars). For the hidden layer we induced a variety of techniques to randomly select the number of hidden units. A major finding was the use of multiple hidden layers did not produce significant gains on the datasets tested. Therefore, we mainly focused on variations of units within a single hidden layer. A last variation is provided by hot starting the neural network weights using a weak learning technique using random projections.

The actual topology selection is compared for the following variations i) neural decision forests with input layer features chosen at random (dots) D, ii) neural decision forests with adjacent input layer features randomly chosen (bars) B, iii) neural decision forests with hidden layer units chosen at random between 1 and the dimension of the feature space, iv) neural decision forests with hidden layer units chosen within a range of the number of features randomly selected and the proportion of the dataset split remaining from the dataset at the root node. Therefore, the naming convention is as follows: $\text{NDF}_{\pi(DH)}$ represents a neural decision forest with input layer randomly chosen in dots, with hidden layer chosen between 1 and the dimension of the feature space. Similarly, NDF$_{(DHC)}$ represents a neural decision forest with input layer randomly chosen with dots, with hiddne layer chosen within the range fo the randomly selected features and proportion of the data set remaining. We report three types of scores, i) **Global** the percentage of all correctly classified instances, ii) **Class-Average** known as *recall* derived from the calculation: $\frac{TP}{TP+FN}$ and, iii) **Jaccard** defined as $\frac{TP}{TP+FP+FN}$ (where TP, FP, FN stand for true positive, false positive, and false negative, respectively. Lastly, we report the average number of splits used in a tree.

4.5.1 Iris Dataset

To assess the behavior of the NDF with respect to a linearly seperable dataset we use this the Iris dataset. The iris dataset has a 4-dimensional feature space and a classification space of 3 labels. The labels are equally reprented at approximately 33.3% making the dataset balanced.

On average using all features we have -0.2/0.1/0.02% change on Global/Class-Average/Jaccard scores compared to the baseline random forest, when using all the neural decision forests in the average, indicating that neural decision forests provide a similar performance to that of the random forest. One of the better performing neural decision forests the NT$_{\pi(DHC)}$ with the 0.7/-.2/.1% change compared to the baseline random forest. The various methods can be compared in Table 4.2.

Methods	Iris			Image-Seg		
	Global	**Class-Avg**	**Jaccard**	**Global**	**Class-Avg**	**Jaccard**
NDF$_{\pi(DH)}$	96.7 ± 2.3	50.6 ± 48.8	49.6 ± 46.6	94.3 ± 3.5	53.3 ± 43.7	52.4 ± 40.3
NDF$_{\pi(DHC)}$	97.4 ± 2.7	50.4 ± 49.6	49.7 ± 48.1	93.3 ± 3.4	52.1 ± 43.8	52.0 ± 38.5
NT$_{DH}$	96.0 ± 4.3	50.1 ± 48.5	49.5 ± 46.1	93.3 ± 4.6	52.4 ± 43.5	51.6 ± 39.6
NT$_{DHC}$	96.1 ± 4.2	50.8 ± 48.0	49.6 ± 46.1	92.4 ± 5.2	53.1 ± 42.1	51.3 ± 37.7
NT$_{BH}$	96.7 ± 3.8	51.0 ± 48.7	49.6 ± 46.9	92.4 ± 3.9	53.6 ± 41.2	53.2 ± 35.2
NT$_{BHC}$	96.0 ± 3.6	51.5 ± 47.2	49.6 ± 46.0	93.8 ± 3.6	53.5 ± 43.1	52.4 ± 38.8
Forest Baseline	96.7 ± 2.3	50.6 ± 48.8	49.6 ± 46.6	92.9 ± 3.7	51.9 ± 44.0	52.1 ± 38.9

Table 4.1: Iris and Image Segmentation Neural Forest experiemntal results with comparisons to baseline Random Forest. All number reported in (%).

When comparing our best global score NT$_{col}$ $\approx$ 97.4% we find similar performance among all the proposed NT frameworks. In fact, the NDF$_{\pi(DH)}$ has the same performance. This leads us to suspect the neural decision forest with random projections may very well simulate the random forest in similar linearly seperable datasets.

4.5.2 Image Segmentation Dataset

To asses the behavior of the NDF with respect to a larger multi-class problem with applications to computer vision and image analys we use the UCI image segmentation data set of 1990. The image segmentation dataset has a 19-dimensional feature space and a classification space of 7 labels. The labels are equally represented at approximately 14.3% making the dataset balanced.

On average using all features we have 0.1/1.1/0.0% change on Global/Class-Average/Jaccard scores compared to the baseline decision tree, when using the

Methods	Handwritten Digits		
	Global	**Class-Avg**	**Jaccard**
NT_{DH}	98.2 ± 0.7	50.2 ± 50.7	49.9 ± 49.2
NT_{BH}	97.6 ± 0.6	50.3 ± 49.9	49.7 ± 48.0
Forest Baseline	97.4 ± 0.5	50.2 ± 49.8	49.4 ± 48.0

Table 4.2: Handwritten Digits Neural Forest experiemntal results with comparisons to baseline Random Forest. All number reported in (%).

NDFs: $\text{NDF}_{\pi(DHC)}$,NDF_{DH}, NDF_{DHC}, NDF_{BH}, NDF_{BHC}, indicating that there is comparable performance, again, with the random forest algorithm. One of the better performing neural decision forests is the $\text{NT}_{\pi(DH)}$ had a 1.4/1.4/0.3% difference in scores from the baseline random forest. The various methods can be compared in Table 4.2.

When comparing our best global score $\text{NT}_{\pi(DH)} \approx 94.3\%$ we find there may be room for performance gain as there is some discrepency among the proposed NDF frameworks. But as each performance measure has a region of overlap with the random forest baseline, we are conservative in what this gain may materialize to.

4.5.3 Handwritten Digits Dataset

To asses the behavior of the NDF with respect to a larger multi-class problem with applications to computer vision and image analys we use the UCI image segmentation data set of 1990. The image segmentation dataset has a 19-dimensional feature space and a classification space of 7 labels. The labels are equally represented at approximately 14.3% making the dataset balanced.

On average using all features we have 0.1/1.1/0.0% change on Global/Class-Average/Jaccard scores compared to the baseline decision tree, when using the NDFs: $\text{NDF}_{\pi(DHC)}$,NDF_{DH}, NDF_{DHC}, NDF_{BH}, NDF_{BHC}, indicating that there is comparable performance, again, with the random forest algorithm. One of the better performing neural decision forests is the $\text{NT}_{\pi(DH)}$ had a 1.4/1.4/0.3% difference in scores from the baseline random forest. The various methods can be compared in Table 4.2.

When comparing our best global score $\text{NT}_{\pi(DH)} \approx 94.3\%$ we find there may be room for performance gain as there is some discrepency among the proposed NDF frameworks. But as each performance measure has a region of overlap with the random forest baseline, we are conservative in what this gain may materialize to.

4.5.4 Artificial Pattern

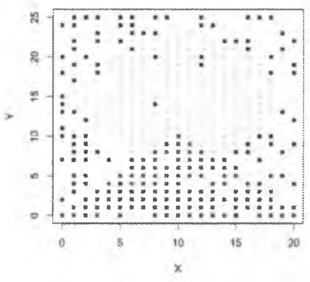
To asses the behavior of the NDF with respect to pattern discrimination with overlapping class distributions we constructed an artificial pattern. The pattern has two dominant shapes: circle and triangle, overlapping, atop a background color.

The pattern was consructed three times. Each time uniform samples were placed in the corresponding regions outlined by the shapes to fill 100%, 75%, and 50%, respectively. To reduced the combinatorial amount of NDF frameworks we choose the NT_{DH} with a floor number of hidden units in each rNET of 2. Figure 4.1 demonstrated the training pattern used on NT_5, this figure is alongside the predictions from NT$_{DH}$ across the entire plane. Similarly, Figures 4.4c and 4.4f show the trianing pattern and prediction from NT$_{DH}$ across the plane.

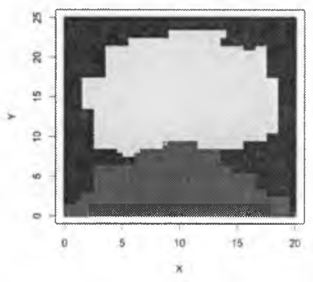
Two aspects worth noting are the relatively stable performance of the NDF to isolate the red triangle. Based on the performance of the NTs of Chapter 3, we anticipated this behavior since this tended to be the first class used in a neural split. The performance of the NDF suggests this class (on average) is isolated first, even within induced randomization. The second observation, is the comparison of the random forest boundary regions to the neural decision forest. In particular, the random forest, appears sufficiently capable of approximating non-orthogaonal boundaries, and also, find interior class segmentation as observed in Figure 4.4b.

4.6 Conclusions

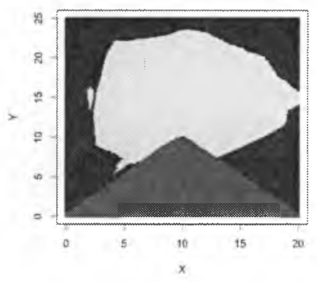
In this work we have introduced our neural decision forest which is an ensemble of neural decision trees with induced randomization. This randomization occurs from bootstrap aggregating individual neural decision trees and embedding rNets as split functions. We proposed a variety of neural decision forest architectures using rNets. In particular, we adjusted the size of input and hidden units based upon the complexity of training set S. The performance of neural decision forest has showsn to be similar if not better than the random forest algorithm over 3 UCI repository datasets. In addition, the neural decision forest offers possible alterations in architectures to be exploited in the future. With an ensemble of neural decision



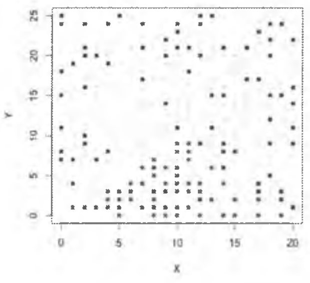
(a) Artificial Pattern 5% reduced sampling.



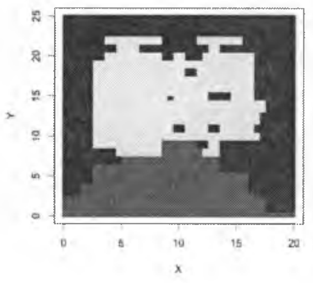
(b) Random Forest Predictions



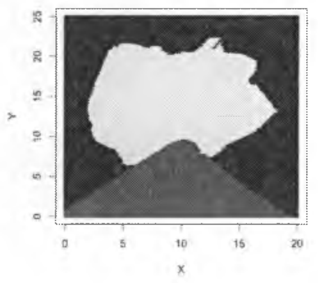
(c) Neural Decision Forest Predictions



(d) Artificial Pattern 15% reduced sampling.



(e) Random Forest Predictions



(f) Neural Decision Forest Predictions

Figure 4.4: Comparitive performance of the neural decision forest and random forest on an artificial pattern with 5% and 25% reduced sampling. (b) and (c) predictions from pattern (a). (e) and (f) predictions from (d).

trees we were able to improve the performance over individual neural decision trees. Lastly, we displayed the differences in hyperplane construction of the random forest compared to neural decision forest on a toy data showing the neural decision forest is capable of forming accurate prediciton with overlapping classes.

Chapter 5

Neural Decision Forests and Personalized EMG Gesture Classification

In this chapter we introdcue the task of Electromyogram (EMG) gesture classificiation. We further discuss how we may alter the randomization within the neural decision forest at each rNet input layer to sample from raw signals in what are called windows. We then describe three experiments comparing one neural decision forest trained on raw signals and a random forest and neural decision forest trained on hand-crafted features.

5.1 Introduction

Machine learning techniques have proven powerful in the control of multifunction prosthesis based gesture classifcaiton problems utilizing electromyogram (EMG) signals. Neural networks [25, 1], decision trees [64], random forest [45], and boosting

[33] techniques have successfully been used in gesture classification. Lacking is a classifier unifying the techniques and a comparative study of bootstrap aggregated (bagged) ensembles of tree-based classifiers. One prerequisite is a a technique capable of working on EMG recordings from different devices and on different features extracted from the signals.

Device differentiation is important to optimally identify the sensitivities posed by inter and intra-individual differences in recordings. Similarly important is how to optimally abstract the data, i.e. choosing the right *data representations* capable of discriminating the varieties of gestures. The rationale for focusing on these two is i) increasing the number of functions of a control scheme without increasing the effort by an amputee, and ii) deriving from the natural contractions in signals more discriminative features facilitating the learning stage of a machine learning algorithm. The neural-machine interface community has an extensive array of alternatives to design appropriate features from the frequency of a signal [46, 64, 55, 10, 16]. These break down into two groups a) domain-specific: the use of features like the signal length, mean, zero-crossings, etc. and b) non-domain-specific: multiple resolution techniques like the fast fourrier transform and wavelet decomposition.

The desire to reduce the dependency on refined and or hand-crafted features has focussed the attention the machine learning community on *representation learning* [6]. The *Neural Decision Forest* algorithm introduced by Bulo and Konschteider was the first deployment successfully learning complex composition in data from signals and therefore combining a form of representation learning within the inductive framework of decision forests [51]. Their approach focused on bayesian optimization and limiting the number of trees leaving in spirit the traditional random forest algorithm.

In this chapter we investigate an original variation of the neural decision forest algorithm where features are generated and selection techniques deployed. This is the first time to our knowledge a comparative study of an ensemble of univariate and multivariate tree classifiers with bootstrap aggregating (bagging) has been peformed on EMG gesture classification. In particular, we propose to benchamark our original variation of the neural decision forest to a baseline random forest classifier.

Ensembles of decision trees known as random forests have been very popular since Breiman authored his seminal work on them in 2001 [14]. The performance, simplicity, and possible generalization for multiple class and parallelization of tree building is reason for their rising popularity [39, 61, 11]. Given the success of random forests [45] and neural networks [25, 1] in EMG signal processing we wonder if improvements might be found if combined. In addition, we know of no existing research applying this algortihm on raw EMG signals and exploiting spectral decomposition possible in various subsets of training data.

As previously mentioned, typically signals have features extracted prior to inducing and training a classifier on the data set. This is restrictive in two senses i) uniformly applying signal processing in windows, or ii) using a multi-resolution technique like wavelet decomposition. Both methods, usually, then subsequently apply a dimensionality technique. To our knowledge there are no known algorithms with strong performance that do not induce a window and sliding technique or multi-resolution technique for processing EMG signals. Our method generalizes the sliding window technique with induced randomization in the number of windows, their size, and location over a given signal[1]. Naturally, this allows generating cascaded features of the data.

While, it is known that neural decision trees generalize decision trees by using multivariate splitting criterias it is not entirely known how neural decision forests improve gesture classification accuracy. Experiments on two datasets gathered from different devices show parallel or comparable performance over stand-alone random forests and proven feature extraction techniques. Further, neural decision forests with randomized windowing techniques provide evidence for future research requiring signal classification.

5.1.1 Related Work

We mainly focus herein on related work in electromyogram (EMG) classificaiton.

For EMG-based gesture classification, existing methods generally consist of three major steps: data segmentation, feature extraction, and pattern recognition. The continuous raw EMG signals are first segmented into analysis windows. Various fea-

[1]One might say this is similar to randomly selecting bases from a multiresolution technique.

tures are then extracted from each analysis window, such as time domain features [63, 1], correlation features [55], and fast-fourrier and wavelet coefficients [25, 60]. In order to achieve high classification accuracy the features usually need to be selected very carefully according to the type of gestures to be recognized and the condition of EMG signals. For pattern classification, neural networks [25], decision trees [64], and boosting [33] techniques have successfully been used. To our knowledge, we are the first group proposing to use bootstrap aggregated (bagged) neural tree-based classifiers for EMG gesture classification. In addition, a novel random windowing scheme is developed, which has the promise of significantly reducing the computational complexity and generalizing the process of data segmentation and feature extraction.

5.2 Neural Decision Forests for Gesture Classification

EMG gesture classification is the problem of labeling a window of EMG signals to a pre-defined gesture. A gesture Y consists of a $(k \times w)$- dimensional tensor with k the number of signals w the size of window. This tensor consists of readings of the muscle fiber action potentials caught in contraction/relaxation. A sample $x \in \mathcal{X}$ which is fed to our neural decision forest is a vector $(u_1, \ldots, u_k)$ concatenating the signals u_i: a window of EMG readings from signal i. The output space $\mathcal{Y}$ consists in a finite set of gesture-labels to be assigned to each recording.

In Chapter 3 we presented neural split functions but omitted how the input layer $f^{(0)}$ may be altered to receive raw signals. In the context of EMG gesture classification, we define the input layer of the rNets at each internal tree node as a function of m_0 random window features. For our signal processing we typically define $m_0 = 2$ two windows over each signal capturing the mean and variance within the window. Let ϕ be the windowing technique with mean and variance extracted then for each $x \in \mathcal{X}$ we have $\phi(x) \in \mathbb{R}^{m_0}$. The input layer function can thus be defined as $f^{(0)} = \phi$.

In figure 5.1 we show a random windowing technique using three windows on three EMG channels. Let u_i be a vector with the mean and varaiance from $\phi(x)$ with $x \in \mathcal{X}$. Then the input to an rNet is the concatenation of the vectors u_i for each random window over each signal. The above example highlights that we may vary the number of windows and statistics computed. In this case, to avoid making a one-shot guess when we determine the set of window-features we balance the use of taking two windows and using the mean and standard deviation within the windows.

5.3 Experiments and Results

We conducted three experiments to evaluate the proposed neural decision forest classification method and the random windowing-based feature extraction strategy. The experiments are i) inter-subject gesture classification using BioPatRec dataset with high sampling frequency, ii) intra-subject gesture classification using BioPatRec

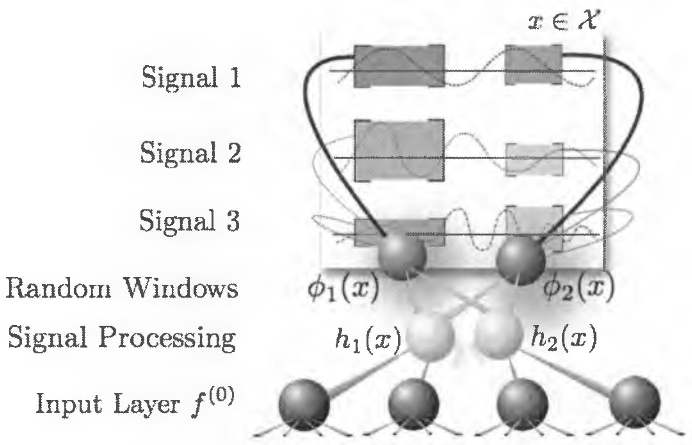


Figure 5.1: Diagram representing two random windows ϕ_1, ϕ_2 generated over 3 signals. The windowes are input to h_1 and h_2. h_1 extracts the mean, h_2 the standard deviation used values for the input layer $f^{(0)}$ of the rNET.

dataset with high sampling frequency, iii) intra-subject gesture classification using Myo dataset with low sampling frequency.

For each experiment, we compare two approaches, the first is a traditional windowing and feature extraction (see Section 5.3.1) approach combined with our novel neural decision forest classifier . The second one is the new random windowing and feature extraction of summary statistics method combined with the neural decision forest classifier. Let NF and RF denote the neural decision forest and random forest, then the combinations are:

- NF := NF + traditional time-domain feature extraction
- RF := RF + traditional time-domain feature extraction
- rNF := NF + random windowing-based feature extraction

5.3.1 Feature Extraction Protocol

Features extracted prior to training consist of the time domain features (TDFs) of a signals: *zero-crossings, length, slope sign change, and mean absolute value.* A windowing technique is commonly used to evaluate the frequency spectrum for the corresponding frame with these features [50]. We window over the signal with a slide overlapping 50%.

The TDFs coupled with a windowing technique produces a large number of coefficients: (# features) $\cdot$ (# slides) $\cdot$ (# signal). Therefore, the amount of TDF coefficients may need be reduced due to the relative imbalance in number of features to training cases (known as the curse of dimensionality). Using *principal component analysis* PCA we find linear combinations of the features explaining the most variance. Picking the largest components, we reduce the number of features and potential to overfit our models.

On the other-hand, using the random window technique on raw signals and computing the mean and variance produces (# windows) $\cdot$ (# signals) $\cdot$ (2) features. In our experiments we produce two random windows over eight signals. We apply the random window technique at the same recording period of each signal.

5.3.2 Experimental Classifier Protocol

Our baseline random forest is implemented by the randomForest package in R [34]. At each split a number of features $k << d$ where d is the dimension of the feature space are chosen. In our experiments we set $k \approx d/3$.

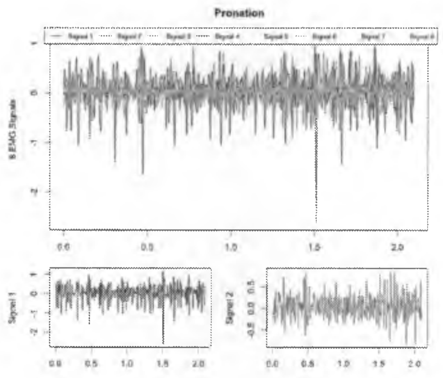
We implemented an original neural decision forest in R. The NF presets are as follows. The input layer to each rNet selects the same number of features as in the random forest. The hidden layer selects the number of neurons from $\pm 20\%$ the number of features used at the input layer. The rNF presets, after random windowing at each rNet, fix the input layer. The hidden layer, randomly selects the number of hidden units $\pm 20\%$ the number of input units.

We construct our rNets using the nnet package in R [58]. Each rNet is trained with a max iteration of 100 to converge at optimal weights and decay factor $5e^{-4}$. We preset the number of trees in the neural and random forest to be equal in each experiment.

In our experiments, as is typical, all trees were constructed until each leaf was pure. In rare cases, we prematurely stop rNet training due to time considerations, so a few leaves may not be pure.

5.3.3 Performance Measurements

We report four types of scores, i) **Global** the percentage of all correctly classified instances, ii) **Precision** how often a classifier predicted a gesture correctly $\frac{TP}{TP+FP}$, iii) **Recall** how often the ground truth of a gesture was predicted by the classifier $\frac{TP}{TP+FN}$, and the **F1-statistic** the harmonic mean of the precision and recall $\frac{\text{Precision}*\text{Recall}}{\text{Precision}+\text{Recall}}$ (where TP, FP, FN stand for true positive, false positive, and false


(a) Pronation Gesture

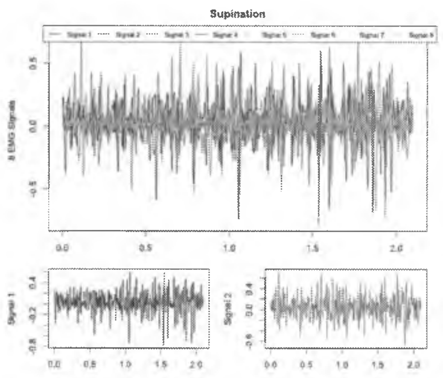

(b) Supination Gesture

Figure 5.2: Two gestures recorded on the Shimmer device with 8 electrodes. Colors denote signals.

negative, respectively.

5.4 BioPatRec Recordings

5.4.1 Data acquisitions system

The BioPatRec recordings [44] correspond to four differentially recorded myoelectric signals digitalized at 2 kHz with a 14-bits resolution. The electrode placement was untargeted but equally spaced around the forearm proximal third. The subjects held between 7 to 27 distinct gestures three times each with a recording of 4s. These recordings were subdivided into five shorter recordings and used for training and testing classifiers.

5.4.2 Personalized Subject Subsytem

5.4.3 Validation Protocol

In order to obtain performance measurements with respect to personaliztion two measurements are needed: 1) Baseline accuracy per user w/out classifiers trained on subject, 2) Personalized accuracy per user w/ classifiers trained on a small portion of data from a subject. For the baseline, the gesture classifier has been trained and validated using a Leave-One-User-Out (LOUO) cross-validation algorithm. In LOUO, each subject is used once for testing on a classifier trained on the rest of the users. This process is repeated for each user, and the performance measurements are computed. For the personalized, a modification in the LOUO protocol has been used. The algorithm needs a small set of data from the subject to be verified for the classifier personalization step, we achieve this small set systematically using folds on data collected from this subject. In this protocol, data belonging to each subject is subject to K-fold subsetting. This process each classifier is tested on the set of gestures in K-1 folds and trained on the Kth plus the LOUO gestures (i.e. all the other subjects). Therefore, we achieve five performance measure per subject and measure the responsiveness of each classifier to the inclusion of the Kth subset in addiition to other subjects.

For RF, NF classifiers the LOUO sets used to train were first decomposed using PCA and the 50 largest principal components (explaining $> 80\%$ of variance) used as the subsequent feature space. The held out user was projected into the principal component space. The rNF constructed two random windows over each signal capturing between 100-300 observations and extracting the mean and standard deviation.

We report the performance measure of 5 randomly chosen subjects due to space considerations. All subjects perform the same 7 gestures: *Pronation, Supintion, Hand-Open, Hand-Close, No Movement, Extension, Flexion.* LOUO training set size is 420 and test 105, repeated 5 times. Personalized training set size is 441 and test 84, over each 5-fold. The performance measurements are reported on forests of size 25 and 50 trees for all classifiers.

5.4.4 Experimental Results

Using the LOUO and Personalized cross-validation and experimental settings described in the former section, the gesture classification performances are shown per forest size in Table 5.1. These tables show the average and standard deviation of each performance measure over all subjects. Figures 5.3 is boxplots per individual with LOUO in blue and Personalization in white. Figure 5.4 displays the overall F1 score per subject with the aforementioned color scheme. Figure 5.5 is boxplots per gesture with a break-out of classifier and tree size comparison.

Table 5.1 highlights the significant gain in all performance measure by the rNF classifier. In terms of LOUO performance, the rNF classifier has an $\approx 10\%$ Global

	Global	Precision	Recall	F1	Global	Precision	Recall	F1
NF	67.39+8.62	67.39+32.92	73.98+26.21	66.2+25.6	85.22+5.77	85.8+19.24	87.53+15.22	84.76+16.27
RF	66.02+8.98	66.02+36.37	70.79+28.49	64.95+27.52	76.64+7.33	77.59+29.12	85.58+19.98	77.22+20.43
rNF	76.57+11.28	76.57+35.55	82.21+27.73	81.74+22.17	93.3+5.96	94.06+15.35	94.56+11.9	93.23+14.23

(a) 25 Trees

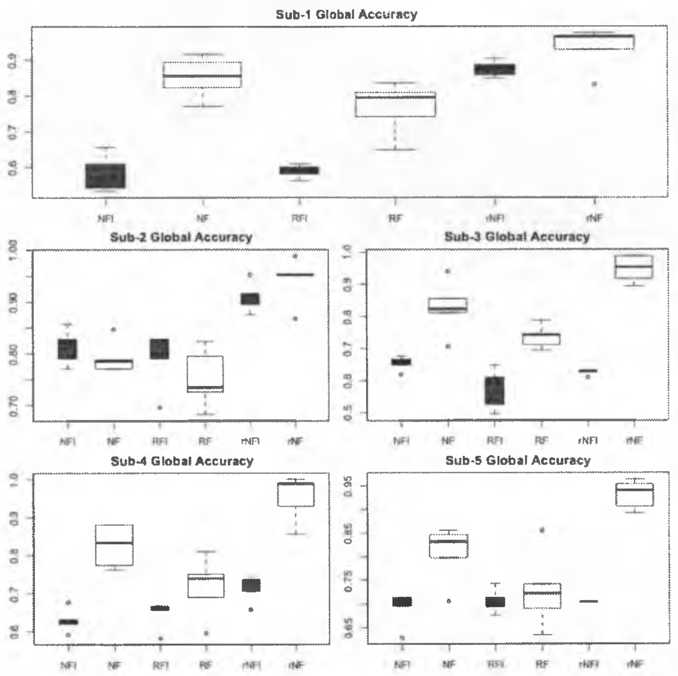
	Global	Precision	Recall	F1	Global	Precision	Recall	F1
NF	69.45+9.59	69.45+33.75	78.15+26.12	67.14+27.53	80.96+5.92	81.42+24.73	87.06+18.35	79.79+19.87
RF	68.8+8.2	68.8+36.34	74.83+28	68.31+27.52	75.21+5.15	75.54+30.95	83.25+22.07	74.98+21.75
rNF	75.89+10.72	75.89+36.25	81.46+27.62	80.73+22.99	95.97+3.49	95.96+7.67	96.34+8.26	95.9+6.8

(b) 50 Trees

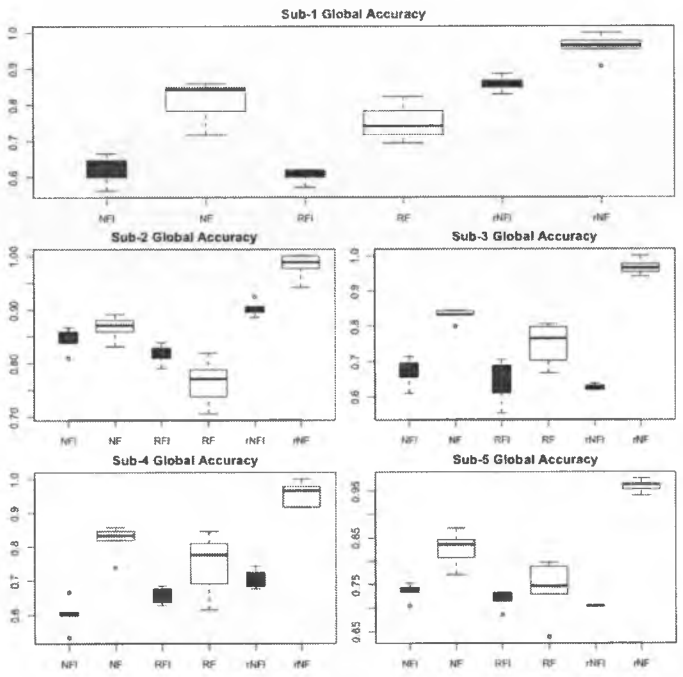
Table 5.1: Inter-subject peformance measures of NF, RF, rNF classifiers displayed in a matrix of LOUO v. Personalized and 25 v. 50 tree forests.

gain on the NF and RF algorithm with 25 trees and $\approx$ 6 Global gain with 50 trees. Similar results apply to the personalized Global performance with rNF $\approx$ 20% better Global performance then RF with 50 trees. Figures 5.3 provide side-by-side comparison performance of each classifier per subject. Of particular note, is the relative improvement of each classifier with Personalized data. Here the rNF shows consistent dramatic increase in subject 3,4,5 in comparison to the NF, RF classifiers. Figure 5.4 displays improvement by all classifiers F1 statistic with personalization. Figure 5.5 displays the F1 statistic per gesture, highlighting the rNF classifier has relative difficulty with two gestures: i) HandOpen, and ii) Pronation.

In conclusion, all activities are classified with increased accuracy due to personalization. But the rNF classifier shows increased sensitivity and improved performance with personaliztion. The rNF classifier show larger mean performance measures in all Global, Precision, Recall, and F1 statistics. In addiiton, the rNF classifier shows more dramatic increase from LOUO to Personalized training. From Figure 5.5 we have reason to believe gestures related per individual can be classified more accurately by a method recognizing class heirarchies such as the NF and rNF classifiers.

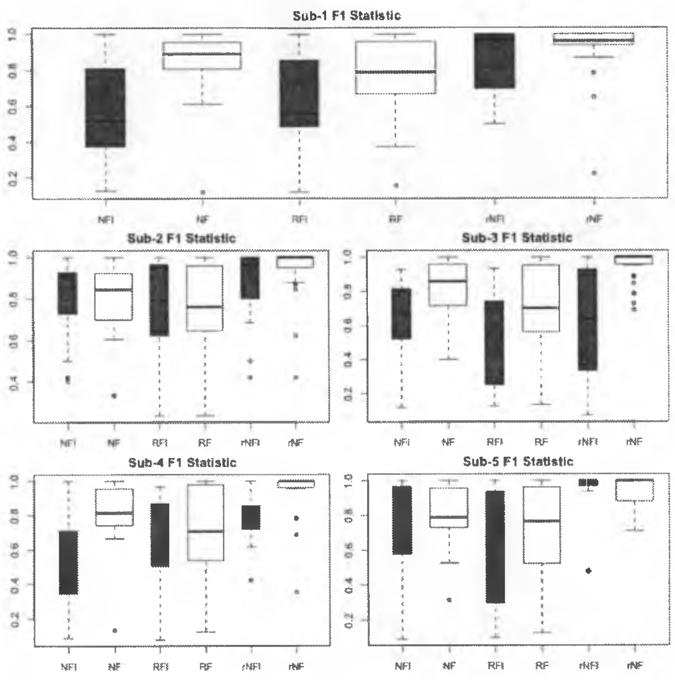


(a) 25 Trees.

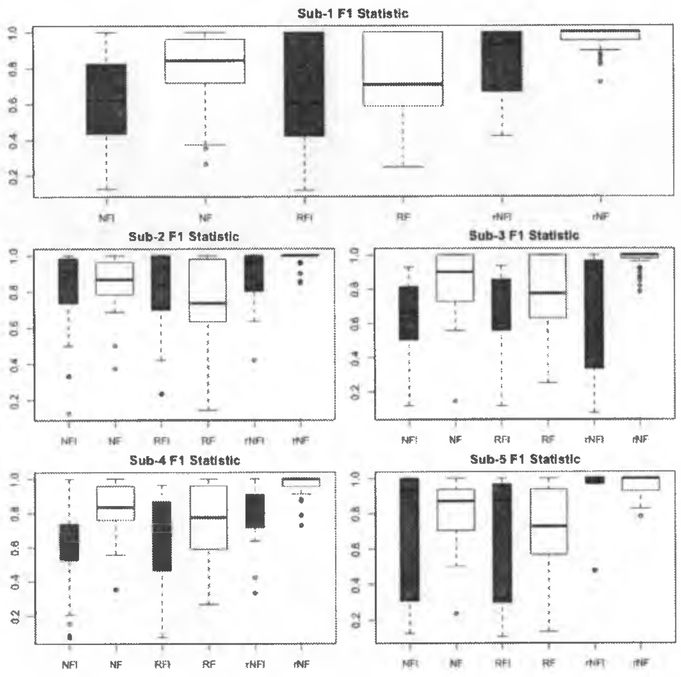


(b) 50 Trees.

Figure 5.3: Intra-subject Global performance of NF, RF, rNF classifiers with contrast of LOUO performance (Blue) and Personalized (White).

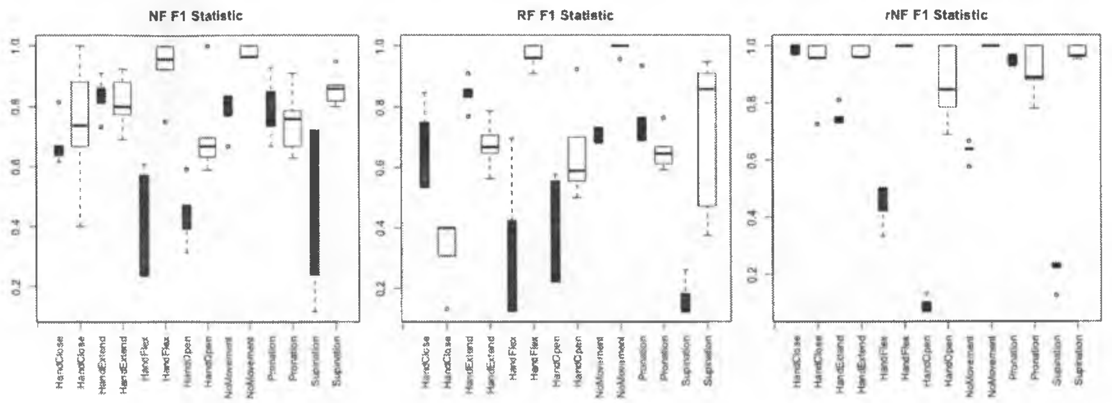


(a) Initial condition

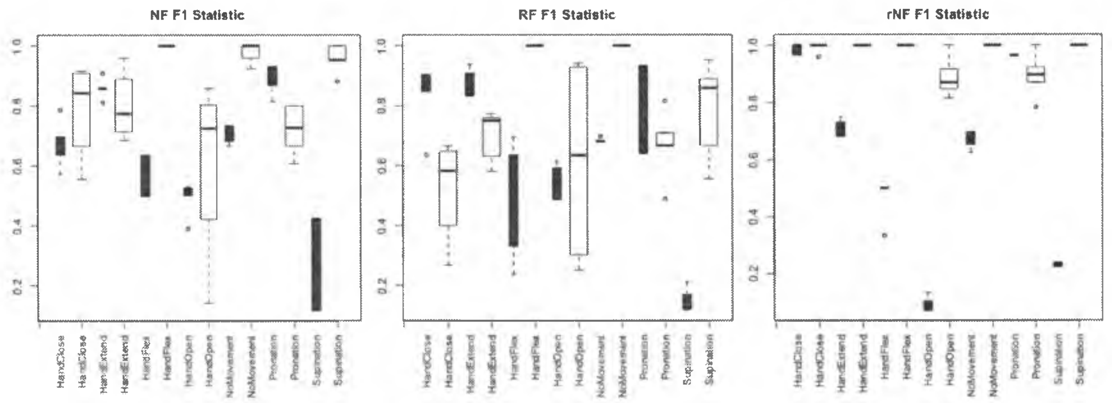


(b) Rupture

Figure 5.4: Intra-subject F1-statistic performance of NF, RF, rNF classifiers with contrast of LOUO performance (Blue) and Personalized (White).



(a) 25 Trees.



(b) 50 Trees.

Figure 5.5: Inter-subject F1-statistic gesture performance of NF, RF, rNF classifiers with contrast of LOUO performance (Blue) and Personalized (White).

5.4.5 27 Gestures Recognition

5.4.6 Validation Protocol

In order to obtain performance measurements, the gesture classifier has been trained and validated using a K-Fold cross-validation algorithm. In K-Fold cross-validation, the dataset is random split into K sets, each set is used once for testing on a classifier trained on the other K-1 sets. This process is repeated for each set, i.e. K times, and the performance measurements are computed. For RF, NF classifiers the K-1 sets used to train were first decomposed using PCA and the 50 largest principal components used as the subsequent feature space. The Kth set was projected into the principal component space. The rNF constructed two random windowes over each signal of capturing between 100-300 samples an extracting the mean and standard deviation.

All subjects perform the same 27 gestures which can be derived from concatenating the following 7 gestures: *Pronation, Supintion, Hand-Open, Hand-Close, No Movement, Extension, Flexion.* The simplest gestures contain one movement, the most complext contain 3 gestures in one recording. The training set size is 324 and test 81 per individual, over each 5-folds. The performance measurements are reported on forests of size 25, 50, and 100 trees for all classifiers.

Model	Global	Precision	Recall	F1
NF	89.24+6.16	90.1+19.94	90.16+19.37	89.47+15.01
RF	88.75+5.58	89.41+21.01	89.62+20.38	89.22+15.42
rNF	93.38+3.46	93.72+16.18	93.79+16	93.71+11.45

(a) 25 Trees

Model	Global	Precision	Recall	F1
NF	90.97+5.46	91.43+19.44	91.2+20.1	91.53+14.19
RF	89.62+7.04	90.6+20.1	90.34+20.79	90.43+14.76
rNF	94.52+3.37	94.75+15.78	94.75+14.83	94.66+11.04

(b) 50 Trees

Model	Global	Precision	Recall	F1
NF	91.96+5.3	92.03+19.84	92.45+17.82	92.64+13.36
RF	91.06+5.38	91.52+19.2	91.75+18.19	91.26+14.29
rNF	94.71+3.27	94.55+15.81	95.26+13.18	94.73+11.02

(c) 100 Trees

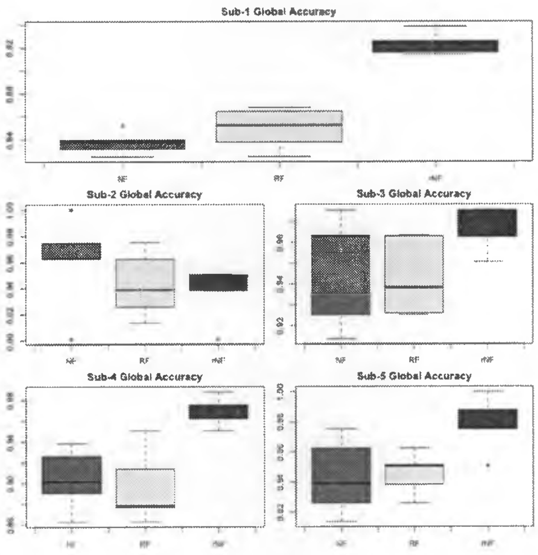
Table 5.2: Inter-subject performance measures comparing NF, RF, rNF classifiers with a) 25 trees, b) 50 trees, c) 100 trees.

5.4.7 Experimental Results

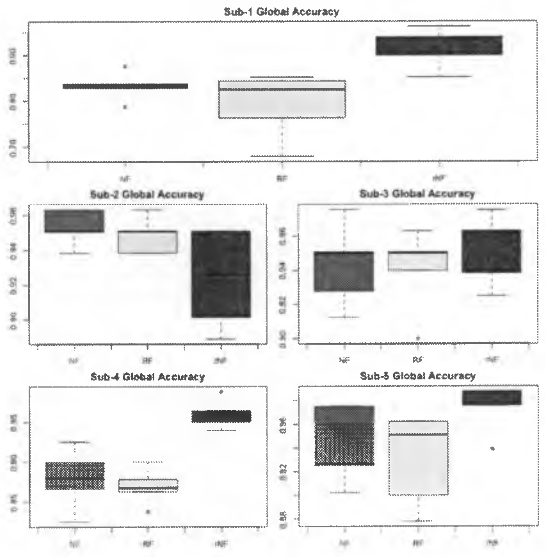
Using the K-Fold cross-validation and experimental settings described in the former section, the gesture classification performances are shown per forest size in Table ??. These tables show the average and standard deviation of each performance measure over all subjects. Figures 5.6 is boxplots per individual. Figure 5.7 displays the F1 score per gesture for subjects 1 and 3, who were randomly selected for space considerations.

Tables ?? shows that with 25 trees the rNF Global performance outperforms both the RF and NF classifiers by $\approx 3\%$ on average. Increasing the number of trees to 100 for alassifiers, the rNF classifiers maintains an $\approx 3\%$ increase in Global performance over RF and NF. Of particular note, the large standard deviance of each classifier in Precision, Recall, and F1 statistic can be explained by the relative disparity of the performance measures on each individual. That is, sometimes rNF is considerably better on one subject than another, for example. Figures 5.7 display a consistent trend that the rNF classifier perform better than both RF and NF. The exception in Subject 2, where RF and NF show improved performance. Lastly, Figures 5.7 provide insight into what differentiate the NF, RF, and rNF classifiers. Consider Subject 3 and the accuracy of each classifier having 50 trees in the forest. The rNF classifier has particular difficulty with Supination gestures; Supination in rNF we might suspect is the root of a subtree with the concatenation of Supination gestures as a subset: therefore, if Supination is wrongly classified subsequent Supination combinations will also.

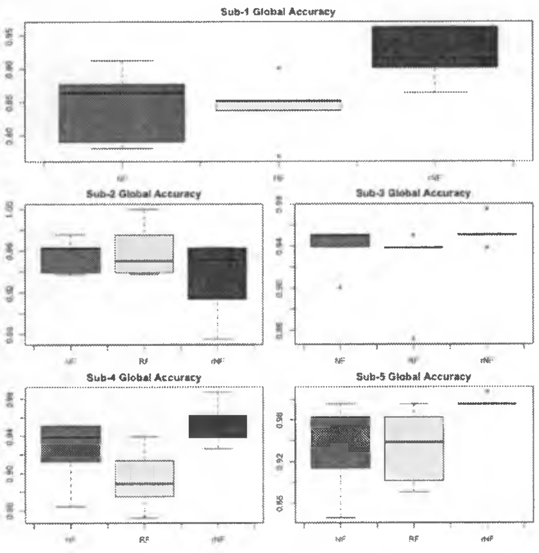
In conclusion, all activities are classified with good peformance. The rNF shows mild improvement over the RF and NF classifiers on both inter/intra subject performance in Global, Precision, Recall, and F1 statistics. It is worth noting, based on Figure 5.7 we have reason to believe gestures closely related can be classified more accurately by a method recognizing class heirarchies such as the NF and rNF classifiers.



(a) 25 Trees.

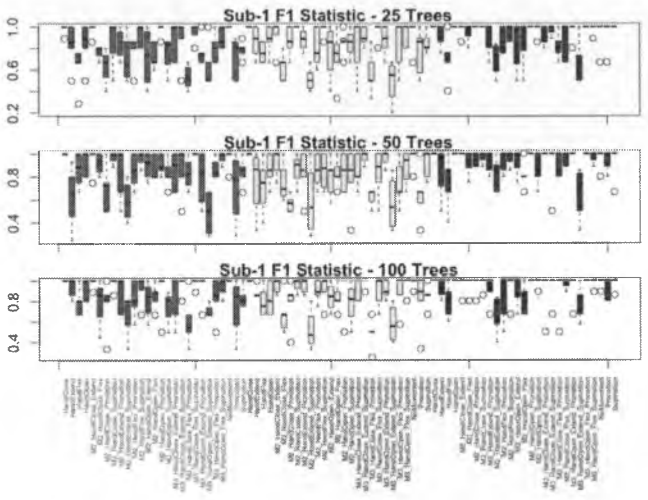


(b) 50 Trees.

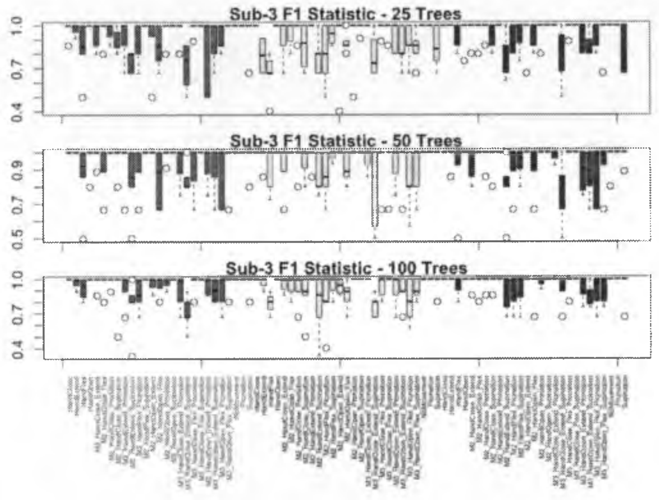


(c) 100 Trees.

Figure 5.6: Intra-subject Global performance of NF, RF, rNF classifiers with a) 25 trees, b) 50 trees, c) 100 trees.

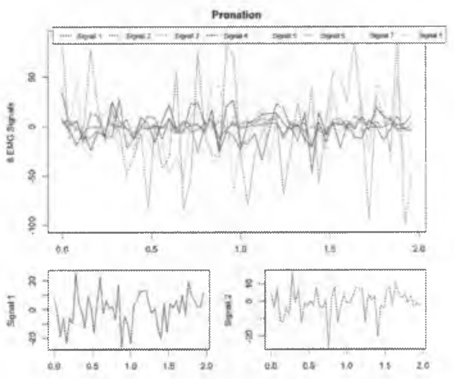


(a) Subject 1

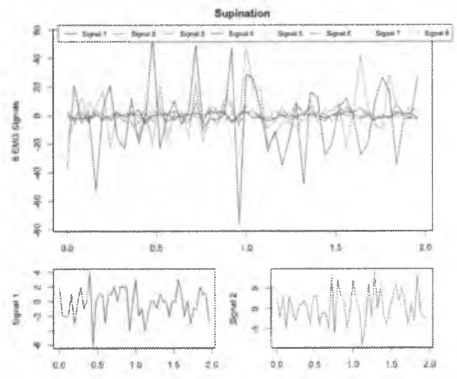


(b) Subject 3

Figure 5.7: F1 statistic on 27 gestures from NF, RF, rNF classifiers for Subject 1 and 3.



(a) Pronation Gesture



(b) Supination Gesture

Figure 5.8: Two gestures recorded on the Myo Thalmic device with 8 electrodes. Colors denote signals.

5.5 Myo Thalmic Dataset

5.5.1 Data acquisitions system

The Myo Thalmic^{TM} band interfaces with a computer through bluetooth 4.0 technology. Signals are digitized in A/D conversion. The sampling rate was set to 25 Hz over 8 signals. The band places 8 sensors in a circular configuration against the superficial level muscles controling hand, palm, and finger movement. Data was recorded from one subject performing 7 gestures 30 times each with a recording of 2s.

5.5.2 Validation Protocol

In order to obtain performance measurements, the gesture classifier has been trained and validated using a K-Fold cross-validation algorithm. In K-Fold cross-validation, the dataset is random split into K sets, each set is used once for testing on a classifier trained on the other K-1 sets. This process is repeated for each set, i.e. K times, and the performance measurements are computed. For RF, NF_i classifiers the K-1 sets used to train were first decomposed using PCA and the 50 largest principal components used as the subsequent feature space. The Kth set was projected into the principal component space. The rNF constructed two random windowes over each signal capturing between 5-40 samples and extracting the mean and standard deviation.

The subject performed the 7 gestures: *Pronation, Supintion, Hand-Open, Hand-Close, No Movement, Extension, Flexion.* The training set size is 168 and test 42, over each 5-folds.

We report the performance measure of two variations. One variation uses a NF selecting 50/5 of the features for each rNET, and the rNF slecting between 5-40 samples per signal for each random window. The second variation, uses an NF selecting 50/3 simulating the ≈ 16 features used in the RF, and rNF selecting between 10-40 features. The performance measurements are reported on forests of size 25, 50, 100, and 500.

	Global	Precision	Recall	F1
NF	51.89+13.14	52.64+29.49	56.77+28.38	50.49+25.12
RF	65.05+5.07	65.53+32.85	68.63+29.78	64.87+27.41
rNF	71.2+8.46	71.82+24.33	73.04+20.55	70.63+20.89

(a) 25 Trees

	Global	Precision	Recall	F1
NF	56.02+4.55	56.78+32.37	55.28+31.06	60.87+23.45
RF	72.22+5.8	72.97+26.89	74.34+24.73	71.64+24.36
rNF	73.5+7.85	72.54+27.32	75.7+18.14	72.52+18.59

(b) 50 Trees

	Global	Precision	Recall	F1
NF	64.63+6.36	64.51+29.66	63.64+28.45	64.4+25.86
RF	73.2+3.87	73.15+29.59	74.24+27.47	73.42+24.03
rNF	75.55+5.87	76.78+20.54	77.36+19.65	75.31+16.67

(c) 100 Trees

	Global	Precision	Recall	F1
NF	71.72+3.96	71.52+26.92	72.02+23.77	70.14+23.22
RF	73.14+6.56	72.55+29.07	73.83+28.07	73.55+24.27
rNF	71.85+5.98	72.28+24.41	74.18+21.62	70.8+21.61

(d) 500 Trees

Table 5.3: Performance measure with NF sampling 50/5 features and rNF random boxes between 5-40 samples compared between a) 25 trees, b) 50 trees, c) 100 trees, d) 500 trees.

5.5.3 Experimental Results

Using the K-Fold cross-validation and experimental settings described in the former section, the gesture classification performances are shown per forest size in Table 5.4. These tables show the average and standard deviation of each performance measure broken into the two variations of NF and rNF. Figures 5.9 is windowplots of classifier accuracy per tree size, and by the two variations of NF and rNF. Figure 5.10 displays the F1 score per gesture per tree size, and by the two variations of NF and rNF.

Tables 5.4 highlight the rNF algorithm improves with the rNF setting random windowes from 10-40. The largest Global accuracy for rNF is $\approx 79\%$ which is a 2% gain over the RF and 12% gain over NF classifiers best Global within the second variation. Of particular importance, RF outperformed the rNF algorithm wihtin

	Global	Precision	Recall	F1
NF	55.96+1.8	57.18+31.04	56.73+31.35	55.55+27.07
RF	64.11+9.34	68.06+28.06	69.42+27.31	65.92+25.19
rNF	69.43+5.25	72.67+22.26	73.24+28.21	70.21+20.28

(a) 25 Trees

	Global	Precision	Recall	F1
NF	60.32+5.77	61.32+35.49	55.64+29.8	63.44+24.15
RF	73.19+6.24	71.45+29.66	72.72+29.07	75.28+23.13
rNF	78.47+2.89	78.88+20.78	78.83+19.52	76.89+17.7

(b) 50 Trees

	Global	Precision	Recall	F1
NF	66.99+10.87	68.08+29.74	67.03+25.13	67.46+22.13
RF	71.35+7.09	71.92+31.39	67.61+29.73	72.93+25.3
rNF	76.13+8.88	75.86+24.3	76.37+21.57	74.61+21.27

(c) 100 Trees

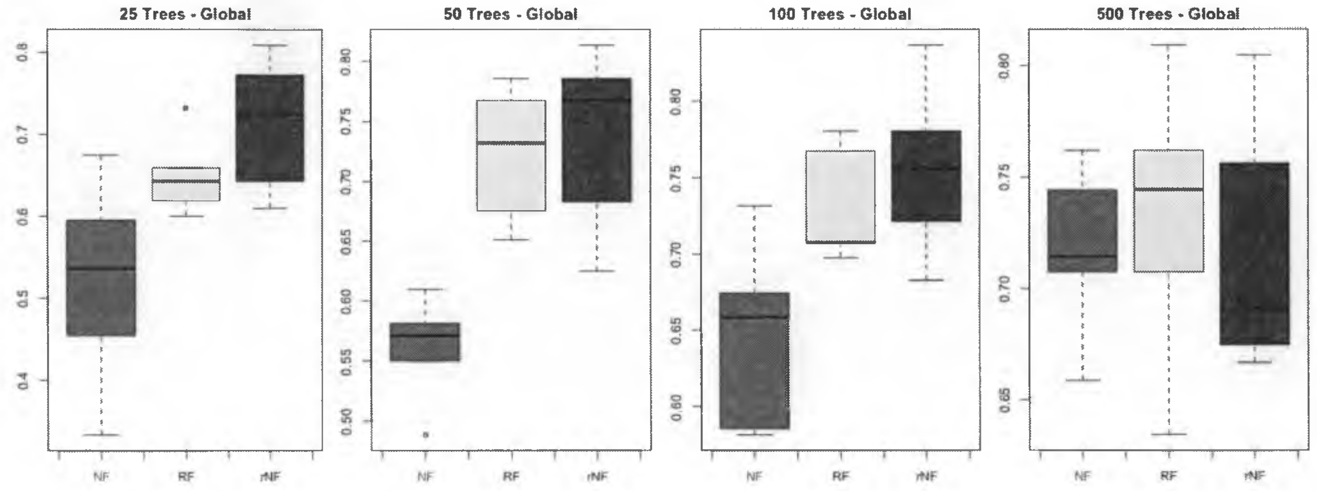
	Global	Precision	Recall	F1
NF	66.95+7.33	67.91+31.75	68.36+27.66	66.72+25.65
RF	76.07+5.36	76.7+25.83	77.22+23.3	75.21+22.97
rNF	79.47+7.79	79.52+20.62	79.84+20.39	78.04+17.64

(d) 500 Trees

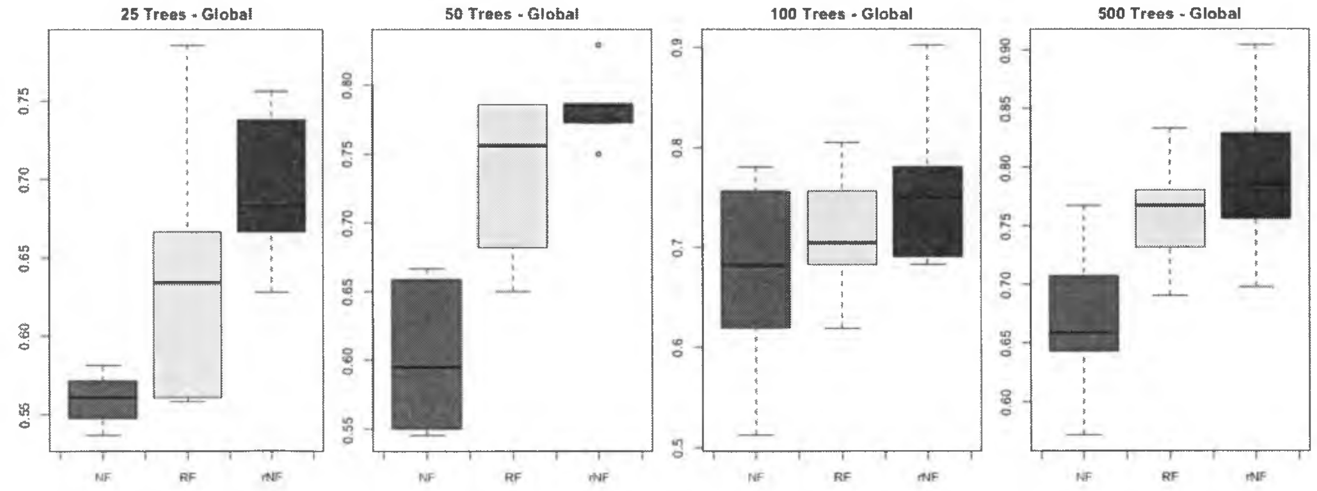
Table 5.4: Performance measure with NF sampling 50/3 features and rNF random boxes between 10-40 samples compared between a) 25 trees, b) 50 trees, c) 100 trees, d) 500 trees.

the first version of rNF settings in all performance measures. Figure 5.10 highlights consisitent performance of the rNF classifier with respect to RF and NF with all tree sizes. Second to the rNF classifier is typically the RF classifier typically within whisker plots of the rNF. Figure 5.10 for concreteness details the F1 statistic per gesture.

In conclusion, all activities are classified with good peformance. The rNF shows mild improvement over the RF and NF classifiers, with drastic improvement with decreased tree size. It is worth noting, based on Figure ?? we detect the rNF has consistent increased performance but cannot be easily verified by per gesture F1 analysis. We suspect the mild improvements of the rNF classifier are due to the lack of class heirarchies to be exploited as seen in previous experiments.

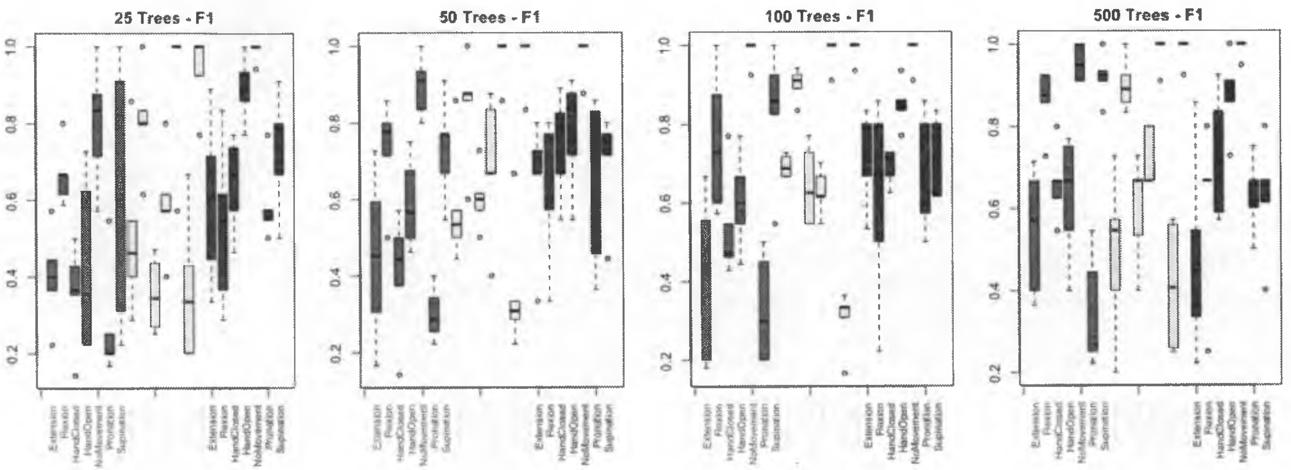


(a) NF sampling 50/5 features and rNF random boxes between 5-40 samples.

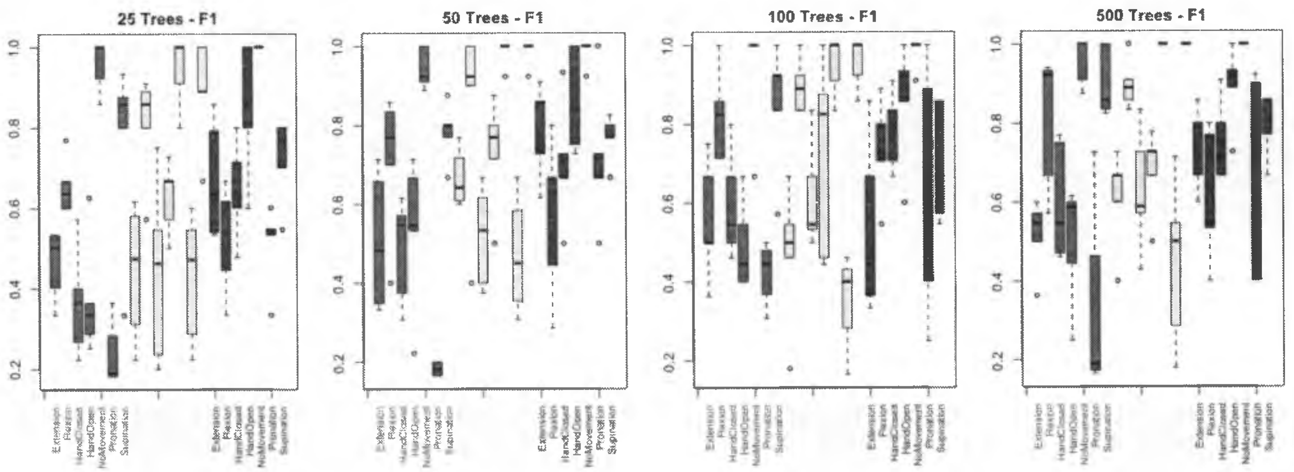


(b) NF sampling 50/3 features and rNF random windowes between 10-40 samples.

Figure 5.9: Inter-gesture performance of NF, RF, rNF classifiers over two variations of sampling: i) NF (50/5) rNF (5-40), ii) NF (50/3) rNF (10-40).



(a) NF sampling 50/5 features and rNF random windowes between 5-40 samples.



(b) NF sampling 50/3 features and rNF random windowes between 10-40 samples.

Figure 5.10: Intra-gesture performance of NF, RF, rNF classifiers over two variations of sampling: i) NF (50/5) rNF (5-40), ii) NF (50/3) rNF (10-40).

5.6 Conclusions

In this work we have introduced Neural Trees (NT) which is a mixture of decision tree and multi-layer perception classifer. It is based on techniques popularized in the decision trees. We proposed a variety of performance neural tree architectures restricted to using a single layer to ease interpretability. Therefore, we considered various techniques of adjusting and setting the size of the hidden units in a neural split. The performance of neural trees has shown to be similar and if not better than decision trees offer future changes in architecture that might be exploited in the future. While the design of hand-crafted features is a typical problem in general, neural trees give a systematic way of potentially utilizing both the advantageous properties of NETs for this reason, while also using the heirarchical routing of classes done in decision tree learning. Using the variety of NT architectures we yielded comparable results to those given by decision trees in the Iris and Image Segmentation datasets. Additionally, we found a NTs have are capable of reporting intermediate pattern discrimination that can be visualized and help in feature extraction.

Chapter 6

Conclusion

In this conclusion I discuss the major themes covered in this thesis from neural trees, forest, and EMG gesture classificaiton. In addition, I remark on future directions.

6.1 Neural Decision Trees and Forests

In the previous chapters we have seen that neural trees provide competitive methods with the CART decision tree algorithm from various classification tasks. Similalry, we have seen that neural decision forest provide competitive methods with the random forest algorithm.

For both neural trees and forest, we evaluated the performance of each on a toy data set showing the strengths of non-orthogaonal hyperplanes can better approximate some shapes.

6.2 Electromyogram (EMG) Gesture Classification

Similarly, we showed that neural decision forests trained on raw signals provide an alternative to previous known methods using a combination of feature extraction and dimensionality reduction techniques prior to training a learning algorithm.

Of particular importance, was how the neural decision forest trained on raw signals was sensitive to newly introduced data in our personalized gesture experiments.

6.3 Future Directions

Two main directions stand out at the conclusion of this thesis:

1. Optimizing the speed of the neural decision forest algorithm.

2. Exploring neural networks acting as auto-encoders.

For the first point, I would like to remark that while I am proud of the performance of the neural deicison forest implemented herein, there is much that can be improved. For example, exchange some routines with calls to C or C++ routines would significantly improve performance. Additionally, while this thesis explored many options and presets available in a neural decision forest, there are many features which can be scaled back which may burden RAM sensitive devices.

And for the second point, the rNets embedded in this neural decision forest implementation beg the question whether alternative gating functions may be used.

For example, is it possible to embed auto-encoders in place of this rNets possibly mimicking algorithms more closely related to soft decision trees [30]. This work though similar, at least conceptually, may involve major rehauling of the underlying tree framework.

Appendices

Appendix A

Source Code

Herein is the neural decision forest package broken into sub-modules where various components and logic are defined. The neural decision forest package is made of files termed: Tree, Forest, Node, Models, and Purity.

A.1 Tree Source

```
source("R/node.R")
source("R/models.R")
source("R/purity.R")


MulTree <- function (args = 0) {
    data <- list(
                tree = list(),
                call = args
                )
    class(data) <- append(class(data), "MulTree")
    return(data)
}

get.top.percent <- function (Y, ...) {
    percent <- sort(table(Y), decreasing = TRUE) / length(Y)
    round(percent[[1]], 2)
}

get.top <- function (Y, ...) {
    names(sort(table(Y), decreasing = TRUE))[[1]]
}

get.X <- function (XY, subset, ...) {
    droplevels(XY[[2]][subset,])
}

get.Y <- function (XY, subset, ...) {
    droplevels(XY[[3]][subset])
}

setup.queue <- function (id, X, Y, ...) {
    list(list(id = id, X = X, Y = Y, label = get.top(Y), percent = get.top.percent(Y)))
}

push.queue <- function (q, id, X, Y, ...) {
    q[[length(q) +1]] <- list(id = id,X = X,Y= Y,label= get.top(Y), percent = get.top.percent(Y)
        )
    q
}

children.payload <- function (XY, s, id,  ...) {
    children <- list(r.id = id + 1 , l.id = id + 2)
    c(XY, s, children)
}

nochildren.payload <- function (XY, s, ...) {
    c(XY, s)
}

grow.tree <- function (t, n, ...) UseMethod("grow.tree")
grow.tree.MulTree <- function (t, n, ...) {
    t$tree[[length(t$tree)+1]] <- n
    t
}

get.branch <- function(t, n, ...) UseMethod("get.branch")
get.branch.MulTree <- function (t, n, ...) {
    for (node in t$tree) {
        if (node[["id"]] == n) {
            return(node)
        }
    }
}

get.leaves <- function (t, ...) {
    leaves <- list()
    for (node in t$tree) {
        if (node$l.id == 0 && node$r.id == 0) {
            leaves[[length(leaves) + 1 ]] <- node
        }
    }
    leaves
}

get.internal <- function (t, ...) {
    internal <- list()
    for (node in t$tree) {
        if (node$l.id != 0 && node$r.id != 0) {
            internal[[length(internal) + 1 ]] <- node
        }
    }
    internal
}

is.a.leaf <- function (n, ...) {
    n$l.id == 0
}

get.parent <- function (t, id) {
    for (node in t$tree) {
        if (node$l.id == id || node$r.id == id) {
            return(node)
        }
    }
}

#' Predicting classes from a multivariate decision tree
#'
#' This function predict classes from a dataframe 'x' based on a 'multree' object.
```

```
#'
#' @param t is a tree from fitted by 'multree'
#' @param X is a dataframe with the same columns fitting 't'.
#'
#' @return a vector of class labels for each row from 'x'
#'
#' @author David Rodriguez
#' @details
#' This function provides class predictions from a dataframe 'x' based on a 'multree' object.
   Nested in this prediction function are the associated predict functions for a 'glm', 'svm',
   'nnet', 'lm', 'lda' object.
#'
#' @seealso \code{glm, svm, nnet, lm, lda}
#' @examples
#' Y  <- iris[,5]
#' X  <- iris[,1:4]
#' dt <- multree(Y, X)
#' p  <- m.predict(dt, X)
#' table(pred = p, actu = Y)

m.predict <- function(t, X, ...) {
    predictions <- c()
    cnt          <- nrow(X)
    pb           <- txtProgressBar(min = 0, max = cnt, style = 3)

    for (iter in seq_along(1:cnt)) {
        setTxtProgressBar(pb, iter)
        node <- get.branch(t, 1)

        while (node$r.id != 0 && node$l.id != 0) {
            go.right <- node$model$predictor(node$fit, X[iter, , drop = FALSE])

            if (go.right) {
                node <- get.branch(t, node$r.id)
            } else {
                node <- get.branch(t, node$l.id)
            }
        }
        predictions  <- c(predictions, node$label)
    }
    predictions
}

a.bag <- function (X, Y, n = 200, ...) {
    s <- length(Y)
    sub <- sample(1:s, n, replace = TRUE)
    list(X = X[sub,], Y = Y[sub])
}

get.children <- function (X, Y, subset, ...) {
    RX <- droplevels(X[subset,])
    RY <- droplevels(Y[subset])
    LX <- droplevels(X[!subset,])
    LY <- droplevels(Y[!subset])
    list( "Right" = list(X = RX, Y = RY), "Left" = list( X = LX, Y = LY))
}

push.children <- function (q, n.id, children, ...) {
    q <- do.call(push.queue, c(list(q = q, id = n.id+1), children[["Right"]]))
    q <- do.call(push.queue, c(list(q = q, id = n.id+2), children[["Left"]]))
    q
}

is.pure <- function (Y, a, ...) {
    if ( length(Y) < 2) {
        return(TRUE)
    }
    n <- table(Y)/length(Y) > a
    sum(n) > 0
}

no.children <- function (t, XY, s, ...) {
    payload <- nochildren.payload(XY, s)
    n       <- do.call(Node, payload)
    t       <- grow.tree(t, n)
}

yes.children <- function (t, XY, s, id, ...) {
    payload  <- children.payload(XY, s, id)
    n        <- do.call(Node, payload)
    t        <- grow.tree(t, n)
    t
}

all.true.false <- function (Y, ...) {
    n <- sum(Y)/length(Y)
    if (n == 0 || n  ==1) {
        return(TRUE)
    } else {
        return(FALSE)
    }
}

rangle <- function(vec) {
    if (length(vec) == 1) {
        return(function(x) vec)
    } else {
        return(function(x) sample(vec, 1))
    }
}

a.tune <- function(tune) {
    if (!is.null(tune)) {
        if ("size" %in% names(tune)) {
            tune[["size"]] = rangle(tune[["size"]])
        }
        if ("degree" %in% names(tune)) {
            tune[["degree"]] = rangle(tune[["degree"]])
        }
    }
    tune
}

init.args <- function(args, Y, X, ...) {

    args[["dist.y"]] <- table(Y)
    args[["m.x"]]     <- nrow(X)
    args[["tune"]]    <- a.tune(args[["tune"]])
    args[["purity"]] <- Purity(args[["purity"]])
    args
}


#' Building a multivariate decision tree
#'
#' This function develops a multivariate decision tree classifier in 'x' to the class of 'y'.
#'
#' @param X a dataframe of continuous or categorical values.
#' @param Y a vector of classes set as factors.
#' @param purity may be: \code{gini, information, twoing}. Sets impurity measure for a node.
#' @param split may be: \code{glm, svm, net, lm, lda}. Sets the hyper surface spliter at nodes
    in the tree.
#' @param a may be: any real value. Cuts-off tree growth if subset has purity greater than \code
    {a}.
#' @param tune may be a list of arguments used instead of presets for \code{glm, svm, net, lm,
    lda}.
#' @param feature.space List identifying the columns of the matrix, and features to extract
    using \code{window, w, k, features} variables.
#' @param window maybe be: \code{all,dots,bars,kbars}. This provides at each split, either
    selecting all the columns (all) or randomly selecting #(cols/2) (dots), or selecting a
    random sequence of columns (bars).
#' @param w may be: used with \code{kbars} specifies window size.
#' @param k may be: used with \code{kbars} specifies the slide (number of \code{w} windows).
#' @param features may be: \code{i, s, m, a}. These represent: i = identity, s = sum, m = mean,
     a = aboslute mean aggregates over filter columns.
#'
#'
#' @return an MulTree object with a tree consisting of nodes and their attributes.
#' @author David Rodriguez
#' @details
#' This function develops a multivariate decision tree classifier to predict an unseen 'x's
     class 'y'. The options of using a generalized linear model, support vector machine, neural
     network, or linear model to construct a hyper-surface splitting plane in ways not found in
     decision trees.
#'
#' @seealso \code{glm, svm, nnet, lm, lda}
#' @examples
#' Y <- iris[,5]
#' X <- iris[,1:4]
#' dt <- multree(Y, X, "svm")

multree <- function(Y, X, split = "svm", a = .8, tune = NULL, purity = "gini", feature.space =
     list(window = "all", w = 0, k = 0, features = "i"), ...) {
    args <- mget(names(formals()),sys.frame(sys.nframe()))[-c(1,2)]

    args <- init.args(args, Y, X)

    id <- 1
    q  <- do.call(setup.queue, list(id, X, Y))
    t  <- MulTree(args)

    while(length(q) > 0) {
        XY <- q[[1]]; q <- q[-1]

        if (is.pure(XY[["Y"]], a)) {
            n <- do.call(Node, XY)
            t <- grow.tree(t, n)
        } else {
            args[["model"]] <- do.call(Model, c(args, XY))
            s               <- do.call(MSplit, c(XY, args))
            subset          <- s[["candidates"]]

            if (is.null(subset) || all.true.false(subset)) {
                t         <- no.children(t, XY, s)
            } else {
                children <- do.call(get.children, c(XY, list(subset = subset)))
                q         <- push.children(q, id, children)
                t         <- yes.children(t, XY, s, id)
                id <- id + 2
            }
        }
    }
    t
}
```

A.2 Forest Source

```
source("R/tree.R")

MulForest <- function (args = 0) {
    data <- list(
                forest = list(),
                call   = args
                )
    class(data) <- append(class(data), "MulForest")
    return(data)
}

m.top <- function(m, ...) {
    stats <- lapply(apply(m,1, list), table)
    top   <- lapply(stats, function(x) names(sort(x, decreasing = TRUE))[1])
    unlist(top)
}


#' Predicting classes from a multivariate decision forest
#'
#' This function predict classes from a dataframe 'x' based on a 'mulforest' object.
#'
#' @param f is a tree from fitted by 'mulforest'
#' @param X is a dataframe with the same columns fitting 'f'.
#'
#' @return a vector of class labels for each row from 'x'
#'
#' @author David Rodriguez
#' @details
#' This function provides class predictions from a dataframe 'x' based on a 'mulforest' object.
    Nested in this prediction function are the associated predict functions for a 'glm', 'svm',
    'nnet', 'lm', 'lda' object.
#'
#' @seealso \code{glm, svm, nnet, lm, lda}
#' @examples
#' Y  <- iris[,5]
#' X  <- iris[,1:4]
#' f  <- mulforest(Y, X)
#' p  <- mf.predict(f, X)
#' table(pred = p, actu = Y)

mf.predict <- function(f, X) {
    p   <- lapply(f$forest, function(x) m.predict(x, X))
    d   <- do.call(cbind, p)
    m.top(d)
}

add.tree <- function (f, t, ...) UseMethod("add.tree")
add.tree.MulForest <- function (f, t, ...) {
    f$forest[[length(f$forest)+1]] <- t
    f
}

#' Building a multivariate decision forest
#'
#' This function develops a multivariate decision forest classifier in 'x' to the class of 'y'.
#'
#' @param X a dataframe of continuous or categorical values.
#' @param Y a vector of classes set as factors.
#' @param purity may be: \code{gini, information, twoing}. Sets impurity measure for a node.
#' @param split may be: \code{glm, svm, net, lm, lda}. Sets the hyper surface spliter at nodes
    in the tree.
#' @param a may be: any real value. Cuts-off tree growth if subset has purity greater than \code
    {a}.
#' @param tune may be a list of arguments used instead of presets for \code{glm, svm, net, lm,
    lda}.
#' @param feature.space List identifying the columns of the matrix, and features to extract
    using \code{window, w, k, features} variables.
#' @param window may be: \code{all,dots,bars,kbars}. This provides at each split, either
    selecting all the columns (all) or randomly selecting #(cols/2) (dots), or selecting a
    random sequence of columns (bars).
#' @param w may be: used with \code{kbars} specifies window size.
#' @param k may be: used with \code{kbars} specifies the slide (number of \code{w} windows).
#' @param features may be: \code{i, s, m, a}. These represent: i = identity, s = sum, m = mean,
    a = aboslute mean aggregates over filter columns.
#' @param size may be: \code{int}. Determines the number of multivariate trees to fit.
#'
#' @return an MulForest object with a forest consisting of trees (multree objects) and their
    attributes.
#' @author David Rodriguez
#' @details
#' This function develops a multivariate decision forest classifier to predict an unseen 'x's
    class 'y'. That is, this replicates the random forest algorithm but with multivariate
    splitting functions. The options of using a generalized linear model, support vector machine
    , neural network, or linear model to construct a hyper-surface splitting plane in ways not
    found in random forest (by Breiman).
#'
#' @seealso \code{glm, svm, nnet, lm, lda}
#' @examples
#' Y <- iris[,5]
#' X <- iris[,1:4]
#' dt <- mulforest(Y, X, "net")


mulforest <- function(Y, X, split = "svm", a = .8,  tune = NULL, purity = "info", feature.space
    = list(window = "dots", w = 0, k = 0, features = "i"), size = 10, ...) {
    args <- mget(names(formals()),sys.frame(sys.nframe()))[-c(1,2)]

    f    <- MulForest(args)
    pb   <- txtProgressBar(min = 0, max = size, style = 3)

    #randomize net size, svm degree, glm to glmnet or lasso
    for (i in seq(size)) {
        setTxtProgressBar(pb, i)
        ss   <- sample(seq(nrow(X)), round(4/5*nrow(X),0), replace = FALSE)
        X.x <- droplevels(X[ss,])
        Y.y <- droplevels(Y[ss])
        t    <- multree(Y.y,X.x, split, a,  tune, purity, feature.space)
        f    <- add.tree(f, t)
    }
    f
}
```

A.3 Node Source

```
source("R/purity.R")
source("R/models.R")

Node <- function (id, X, Y, label, model = 0, fit = 0, r = 0, percent= 0,  cutoff= .5, l.id = 0,
      r.id = 0, candidates = 0, gain = 0) {
    data <- list(
                    id          = id,
                    X           = X,
                    Y           = Y,
                    label       = label,
                    candidates  = candidates,
                    r           = r,
                    gain        = gain,
                    l.id        = l.id,
                    r.id        = r.id,
                    model       = model,
                    fit         = fit,
                    percent     = percent,
                    cutoff      = cutoff
                    )
    class(data) <- append(class(data), "Node")
    return(data)
}

swap <- function (i, r) {
    if (i %in% r) {
        if (length(r) == 1) {
            r
        } else {
            setdiff(r, i)
        }
    } else {
        tmp <- unlist(lapply(r, as.character))
        c(tmp, as.character(i))
    }
}

load.it <- function (model, fit, Rcandidates, g, r, ...) {
    list(model = model, fit = fit, candidates = Rcandidates, gain = g, r = r)
}

load.set <- function (Y, X, model, r, purity, ...) {
    fit         <- model$call(Y, X, r)
    Rcandidates <- model$predictor(fit, X)
    i           <- gain(purity, Y, Rcandidates)

    load.it(model, fit, Rcandidates, i, r)
}

load.swap <- function (c, Y, X, model, r, purity) {
    r <- swap(c,r)
    load.set(Y, X, model, r, purity)
}

init.split <- function (c, ...) {
    mid <- floor(length(c)/2)
    sample(c, mid)
}

get.max <- function (i.delta, ...) {
    gains <- unlist(lapply(i.delta, function(x) x$gain))
    ord   <- order(gains, decreasing = TRUE)
    i.delta[ord][[1]]
}

exchange <- function (Y, X, model, fit, r, purity, ...) {
    c <- unique(Y)

    if (length(setdiff(c,r)) == 0 ) {
        return(load.it(model, fit, Y[Y %in% r], 0, r))
    } else if (length(setdiff(c,r)) == 1) {
        a <- setdiff(c,r)
        c <- as.factor(setdiff(c,a))
    }

    i.delta  <- lapply(c, function(x) load.swap(droplevels(x), Y, X, model, r, purity))
    get.max(i.delta)
}

MSplit <- function (Y, X, model, purity, ...) {
    c <- unlist(lapply(unique(Y), as.character))

    i.r         <- init.split(c)
    i.candidate <- load.set(Y, X, model, i.r, purity)
    i.gain      <- i.candidate[["gain"]]
    i.fit       <- i.candidate[["fit"]]

    max.swap    <- exchange(Y, X, model, i.fit, i.r, purity)
    iter.gain   <- max.swap$gain
    while(iter.gain - i.gain > 0) {

        i.candidate <- max.swap
        i.gain      <- i.candidate[["gain"]]
        i.fit       <- i.candidate[["fit"]]
        i.r         <- i.candidate[["r"]]

        max.swap    <- exchange(Y, X, model, i.fit, i.r, purity)
        iter.gain   <- max.swap$gain
    }
    i.candidate
}
```

A.4 Models Source

```
require(nnet)
require(MASS)
require(e1071)
require(neuralnet)


Model <- function (split, tune = NULL, feature.space, Y, X, dist.y, m.x, ...) {
    args <- mget(names(formals()),sys.frame(sys.nframe()))

    filter     <- do.call(toupper(feature.space$window), c(x.n = ncol(X), feature.space))
    cplx       <- do.call(Complexity, c(args, filter = filter))
    extractors <- lookup.features(feature.space$features)
    presets    <- lookup.presets(split, cplx)
    pcall      <- pre.call(split, tune, presets, filter, extractors)
    ppredict   <- pre.predict(split, filter, extractors)

    data <- list(
                 type      = split,
                 tuned     = tune,
                 presets   = presets,
                 filter    = filter,
                 extrators = extractors,
                 call      = pcall,
                 predictor = ppredict
                 )
    class(data) <- append(class(data), "Model")
    return(data)
}

Complexity <- function(Y, X, dist.y, m.x, feature.space, filter, ...) {

    m   <- nrow(X)
    n   <- ncol(X)
    u.y <- unique(droplevels(Y))
    t.y <- table(droplevels(Y))

    feats <- length(unlist(strsplit(feature.space$features, ":")))

    if (feature.space$window == "kbars") {
        f.vect <- feature.space$k * feats # k > 1
    } else {
        f.vect <- length(unlist(filter))

    }

    data <- list(x.m       = m,
                 x.n       = n,
                 c.n       = length(u.y),
                 o.c.prop  = length(u.y)/length(names(dist.y)),
                 t.prop    = m/m.x,
                 c.prop    = t.y/ unlist(lapply(u.y, function(x) dist.y[x])),
                 f.prop    = f.vect/n,
                 f.n       = f.vect
                 )
    class(data) <- append(class(data), "Complexity")
    return(data)
}

lookup.features <- function(features, ...) {
    f.list <- list("i" = I,
                   "s" = sum,
                   "m" = mean,
                   "e" = sd,
                   "k" = kurtosis,
                   "u" = skewness,
                   "a" = function(x) mean(abs(x)),
                   "d" = function(x) sum(x > c(0,x[-length(x)])) - 1,
                   "r" = function(x) sum(x < c(0,x[-length(x)])),
                   "o" = function(x) sum(x > (mean(x) + sd(x))) + sum(x < (mean(x) - sd(x))),
                   "l" = function(x) sum(abs(x[2:length(x)] - x[-length(x)])),
                    "f" = function(x) sum((x[2:length(x)] - x[-length(x)])^2),
                    "p" = function(x) {y <- (x - c(0,x[-length(x)]))[-c(1)]; if (sum(y>0) == 0)
                        return(0) else mean(y[y > 0])},
                    "n" = function(x) {y <- (x - c(0,x[-length(x)]))[-c(1)]; if (sum(y<0) == 0)
                        return(0) else mean(y[y < 0])},
                    "q" = function(x) {y <- (x - c(0,x[-length(x)]))[-c(1)]; if (sum(y>0) == 0)
                        return(0) else sd(y[y > 0])},
                    "w" = function(x) {y <- (x - c(0,x[-length(x)]))[-c(1)]; if (sum(y<0) == 0)
                        return(0) else sd(y[y < 0])}
                    )
    decode <- unlist(strsplit(features, ":"))
    f.list[names(f.list) %in% decode]
}

lookup.presets <- function (model, cplx, ...) {
    preset.list <- list("net" = list(size      = cplx$x.n,
                                     trace     = FALSE,
                                     decay     = 5e-4),
                                     #Wts        = pre.wts(Y.hat, X, n,length(r))),
                        "rnet" = list(size     = rNET(cplx)$size,
                                      MaxNWts = 15000,
                                      trace    = FALSE,
                                      decay    = 5e-4),
                                      #Wts       = pre.wts(Y.hat, X, n,length(r))),
                        "rmlp" = list(hidden   = rMLP(cplx)$n.units,
                                      act.fct = "logistic",
                                      rep      = 5,
                                      linear.output = FALSE),
                        "mlp"  = list(hidden   = c(5,5),
                                      act.fct = "logistic",
                                      rep      = 3,
                                      linear.output = FALSE),
                        "svm" = list(),
                        "lda" = list(),
                        "glm" = list(family = "gaussian"),
                        "lm"  = list())
    preset.list[[model]]
}

quick.p <- function (x.n, f.n, t.prop) {
    k <- round(x.n*1.5,0) - f.n
    p <- (k+1):1/(k+1)
    p <- p/sum(p)
    p * (1- t.prop) + p[length(p):1] * t.prop
}

rNET <- function(cplx) {
    f.n      <- cplx$f.n
    x.n      <- cplx$x.n
    t.prop   <- cplx$t.prop
    o.c.prop <- cplx$o.c.prop

    if (f.n == x.n) {
        size <- sample(seq(round(f.n * (.5 + o.c.prop),0), round(f.n*(.5 + 1.25),0)),1)
    } else {
        size <- sample(seq(f.n, round(x.n*1.5,0)),1, prob = quick.p(x.n, f.n, t.prop))
    }
    data <- list(size = size
                 )
    class(data) <- append(class(data), "rNET")
    return(data)
}


rMLP <- function(cplx) {
    c.n      <- cplx$c.n
    o.c.prop <- cplx$o.c.prop
    f.n      <- cplx$f.n
    x.n      <- cplx$x.n
    t.prop   <- cplx$t.prop

    layers      <- if(c.n == 2) 1 else max(round(o.c.prop * 2,0), 1)
    mine        <- max(round(f.n * 1/2,0),2)
    maxe        <- max(round(f.n * (1/2) + x.n * t.prop, 0),2)
    a           <- seq(mine, maxe)
    layer.units <- sample(c(a), layers, replace = TRUE)

    data <- list(n.layers  = layers,
                 n.units   = layer.units
                 )
    class(data) <- append(class(data), "rMLP")
    return(data)
}

ALL <- function(x.n, ...) {
    list(seq(x.n))
}

DOTS <- function(x.n, ...) {
    if (x.n == 1) {
        return(list(x.n))
    } else if (x.n == 2) {
        a <- sample(x.n, 1)
        return(list(sample(x.n, a, replace = FALSE )))
    } else {
        return(list(sample(x.n, round(x.n/2, 0), replace = FALSE )))
    }
}

BARS <- function(x.n, low = 1, high = x.n, ...) {
    l   <- sample(seq(low,high, 1), 1)
    bar <- 1:l
    p   <- sample(seq(0,x.n-1), 1)

    list(bar + p)
}

KBARS <- function(w, k, low = 1, high = w, ...) {
    print(c(w, k, low, high))
    lapply(c(0,seq(k-1)), function(x) c(x*w + unlist(BARS(w, low, high))))
}

random.box <- function(X, filter, extractors) {
    f <- lapply(filter, function(x) X[, x, drop = FALSE])
    f <- lapply(extractors, function(x) do.call(cbind, Map(function(y) apply(y, 1, x), f)))
    data.frame(do.call("cbind", f))
}

run.tuners <- function(tune) {
    if ("size" %in% names(tune)) {
        tune[["size"]] = tune[["size"]]()
    }
    if ("degree" %in% names(tune)){
        tune[["degree"]] = tune[["degree"]]()
    }
    if ("hidden" %in% names(tune)){
        tune[["hidden"]] = tune[["hidden"]]()
    }
    tune
}

pre.call <- function (model, tune, presets, filter, extractors,  ...) {
    if (is.null(tune)) {
        params <- force(presets)
    } else {
        tune <- run.tuners(tune)
        params <- force(tune)
    }
    if ("i" %in% names(extractors)) {
        mod <- function(Y, X, r, ...) {
            do.call(toupper(model), list(Y = Y, X = X[,unlist(filter), drop = FALSE], r = r,
                 params = params))
        }
    } else {
        mod <- function(Y, X, r, ...) {
            mm.dat <- random.box(X, filter, extractors)
            print(ncol(mm.dat))
            do.call(toupper(model), list(Y = Y, X = mm.dat, r = r, params = params))
        }
    }
}

pre.predict <- function(model, filter, extractors, ...) {
    name <- force(paste0(toupper(model),".predict"))

    if ("i" %in% names(extractors)) {
        predictor <- function(n, X, ...) {
            do.call(name, list(n = n, X = X[,unlist(filter), drop = FALSE]))
        }
    } else {
        predictor <- function(n, X, ...) {
            mm.dat <- random.box(X, filter, extractors)
            do.call(name, list(n = n, X = mm.dat))
        }
    }
}

tune <- function (model, tuners, ...) UseMethod("tune")
tune.Model <- function (model, tuners, ...) {
    model$tuned <- tuners
    model
}

set.call <- function (model, ...) UseMethod("set.call")
set.call.Model <- function (model, ...) {
    if (is.null(model$tuned)) {
        params <- force(model$presets)
    } else {
        params <- force(model$tuned)
    }

    model$call <- function(Y, X, r, ...) {
        do.call(toupper(model$type), list(Y = Y, X = X, r = r, params = params))
    }
}

classEx <- function (Y, r, ...) {
    unlist(lapply(Y, function(x) x %in% r))
}

NET <- function (Y, X, r, params, ...) {
    Y.hat <- classEx(Y,r)

    forms <- list(formula = as.formula("Y.hat ~ ."),
                  data = X)
    #w <- pre.wts(Y.hat, X, params$size, 1)
    #do.call("nnet", c(forms, params, list(Wts = w)))
    do.call("nnet", c(forms, params))
}

RNET <- function (Y, X, r, params, ...) {
    Y.hat <- classEx(Y,r)
    forms <- list(formula = as.formula("Y.hat ~ ."),
                  data = X)
    #w <- pre.wts(Y.hat, X, params$size, 1)
    #do.call("nnet", c(forms, params, list(Wts = w)))
    do.call("nnet", c(forms, params))
}

MLP <- function (Y, X, r, params, ...) {
    Y.hat <- classEx(Y,r)
    Y.hat <- as.numeric(Y.hat)


    dat   <- cbind(1 = Y.hat, X)
    forms <- list(formula = as.formula(paste("1 ~ ", paste(colnames(X), collapse = "+"),
         collapse = "")),
                  data = dat)

    do.call("neuralnet", c(forms, params))
}

RMLP <- function (Y, X, r, params, ...) {
    Y.hat <- classEx(Y,r)
    Y.hat <- as.numeric(Y.hat)


    dat   <- cbind(1 = Y.hat, X)
    forms <- list(formula = as.formula(paste("1 ~ ", paste(colnames(X), collapse = "+"),
         collapse = "")),
                  data = dat)

    do.call("neuralnet", c(forms, params))
}

GLM <- function (Y, X, r, params, ...) {
    Y.hat <- classEx(Y,r)

    forms <- list(formula = as.formula("Y.hat ~ ."),
                  data = X)

    do.call("glm", c(forms, params))
}

SVM <- function (Y, X, r, params, ...) {
    Y.hat <- classEx(Y,r)
    Y.hat <- as.factor(Y.hat)

    forms <- list(formula = as.formula("Y.hat ~ ."),
                  data = X)

    do.call("svm", c(forms, params))
}

LDA <- function (Y, X, r, params, ...) {
    Y.hat <- classEx(Y,r)

    forms <- list(formula = as.formula("Y.hat ~ ."),
                  data = X)

    do.call("lda", c(forms, params))
}

LM <- function (Y, X, r, params, ...) {
    Y.hat <- classEx(Y,r)

    forms <- list(formula = as.formula("Y.hat ~ ."),
                  data = X)

    do.call("lm", c(forms, params))
}

NET.predict <- function(n, X) {
    preds <- c(predict(n, X))
    preds > .5
}

RNET.predict <- function(n, X) {
    preds <- c(predict(n, X))
    preds > .5
}

MLP.predict <- function(n, X) {
    preds <- compute(n, X)$net.result
    preds > .5
}
```

```
RMLP.predict <- function(n, X) {
    preds <- compute(n, X)$net.result
    preds > .5
}

GLM.predict <- function(n, X) {
    preds <- c(predict(n, X))
    preds > .5
}

LM.predict <- function(n, X) {
    preds <- c(predict(n, X))
    preds > .5
}

SVM.predict <- function(n, X) {
    as.logical(as.vector(predict(n,X)))
}

LDA.predict <- function(n, X) {
    c(predict(n, X)[["class"]])
}
```

A.5 Purity Source

```
Purity <- function (type = "info") {

    pcall <- pre2.call(type)

    data <- list(
                type       = type,
                call       = pcall,
                gain       = NULL,
                gain.ratio = NULL
                )
    class(data) <- append(class(data), "Purity")
    return(data)
}

pre2.call <- function (type, ...) {
    purity <- function(Ys, ...) {
        do.call(toupper(type), list(Ys = Ys))
    }
}

gain <- function(Purity, Y, Rcandidates, ...) UseMethod("gain")
gain.Purity <- function (Purity, Y, Rcandidates, ...) {
    IR      <- Purity$call(Y[Rcandidates])
    IL      <- Purity$call(Y[!Rcandidates])
    p       <- sum(Rcandidates)/length(Y)

    if (identical(Purity$type,"twoing")) {
        return(abs.sum(IL, IR, p))
    } else {
        return(get.gain(Purity$call(Y), IL, IR, p))
    }
}

gain.ratio <- function (Purity, Y, Rcandidates, ...) UseMethod("gain.ratio")
gain.ratio.Purity <- function (Purity, Y, Rcandidates, ...) {
    p <- sum(Rcandidates)/length(Rcandidates)
    g <- gain(Purity, Y, Rcandidates)
    g/potential(p)
}

INFO <- function (Ys, ...) {
    Ys <- droplevels(Ys)
    p  <- table(Ys)/length(Ys)
    -1*sum(p*log(p))
}

GINI <- function (Ys, ...) {
    Ys <- droplevels(Ys)
    p <- table(Ys)/length(Ys)
    1 - sum(p^2)
}

TWOING <- function (Ys, ...) {
    table(Ys)/length(Ys)
}

potential <- function (p, ...) {
    -(p * log(p) + (1-p) * log(1-p))
}

get.gain <- function (I, IL, IR, p, ...) {
    I - p * IR - (1-p) * IL
}

abs.sum <- function (IL, IR, p, ...) {
    p * (1-p) * sum(abs(IL - IR))^2 / 4
}
```

Bibliography

[1] M.R. Ahsan, M.I. Ibrahimy, and O.O. Khalifa, *Electromygraphy (emg) signal based hand gesture recognition using artificial neural network (ann)*, Mechatronics (ICOM), 2011 4th International Conference On, May 2011, pp. 1–6.

[2] Kamal M Ali and Michael J Pazzani, *Error reduction through learning multiple descriptions*, Machine Learning **24** (1996), no. 3, 173–202.

[3] Eric Bauer and Ron Kohavi, *An empirical comparison of voting classification algorithms: Bagging, boosting, and variants*, Machine learning **36** (1999), no. 1-2, 105–139.

[4] George Bebis and Michael Georgiopoulos, *Feed-forward neural networks*, Potentials, IEEE **13** (1994), no. 4, 27–31.

[5] Richard Bellman, *Dynamic programming princeton university press*, Princeton, NJ (1957).

[6] Y. Bengio, A. Courville, and P. Vincent, *Representation learning: A review and new perspectives*, Pattern Analysis and Machine Intelligence, IEEE Transactions on **35** (2013), no. 8, 1798–1828.

[7] Yoshua Bengio, Olivier Delalleau, and Nicolas L Roux, *The curse of highly variable functions for local kernel machines*, Advances in neural information processing systems, 2005, pp. 107–114.

[8] Yoshua Bengio, Olivier Delalleau, and Clarence Simard, *Decision trees do not generalize to new variations*, Computational Intelligence **26** (2010), no. 4, 449–467.

[9] K. P. Bennett and J. A. Blue, *A support vector machine approach to decision trees*, 1997.

[10] F. Bitar, N. Madi, E. Ramly, M. Saghir, and F. Karameh, *A portable midi controller using emg-based individual finger motion classification*, Biomedical Circuits and Systems Conference, 2007. BIOCAS 2007. IEEE, Nov 2007, pp. 138–141.

[11] Henrik Boström, *Concurrent learning of large-scale random forests.*, SCAI, vol. 227, 2011, pp. 20–29.

[12] Leo Breiman, *Classification and regression trees*, Chapman & Hall, New York, 1993.

[13] ______, *Bagging predictors*, Mach. Learn. **24** (1996), no. 2, 123–140.

[14] ______, *Random forests*, Machine Learning **45** (2001), no. 1, 5–32.

[15] Carla E. Brodley, Carla E. Brodley, and Paul E." Utgoff, *Multivariate decision trees*, (1992).

[16] Liyu Cai, Zhizhong Wang, and Haihong Zhang, *Classifying emg signals using t-f representation and svd [for artificial limb control]*, [Engineering in Medicine and Biology, 1999. 21st Annual Conference and the 1999 Annual Fall Meetring of the Biomedical Engineering Society] BMES/EMBS Conference, 1999. Proceedings of the First Joint, vol. 1, 1999, pp. 575 vol.1–.

[17] David Maxwell Chickering, David Maxwell Chickering, Christopher Meek, and Robert" Rounthwaite, *Efficient determination of dynamic split points in a decision tree*, IN PROCEEDINGS OF THE 1ST IEEE INTERNATIONAL CONFERENCE ANYTIME INDUCTION OF LOW-COST, LOW-ERROR CLASSIFIERS ON DATA MINING (ICDM-2001 (2001), 91–98.

[18] Adam Coates and Andrew Ng, *The importance of encoding versus training with sparse coding and vector quantization*, Proceedings of the 28th International Conference on Machine Learning (ICML-11) (New York, NY, USA) (Lise Getoor and Tobias Scheffer, eds.), ICML '11, ACM, June 2011, pp. 921–928.

[19] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to algorithms, third edition*, 3rd ed., The MIT Press, 2009.

[20] Thomas G. Dietterich, *Ensemble methods in machine learning*, Proceedings of the First International Workshop on Multiple Classifier Systems (London, UK, UK), MCS '00, Springer-Verlag, 2000, pp. 1–15.

[21] Thomas G Dietterich, *An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization*, Machine learning **40** (2000), no. 2, 139–157.

[22] Nir Friedman, Nir Friedman, and Moises" Goldszmidt, *Learning bayesian networks with local structure*, (1996).

[23] Ken-Ichi Funahashi, *On the approximate realization of continuous mappings by neural networks*, Neural networks **2** (1989), no. 3, 183–192.

[24] Pall Oskar Gislason, Jon Atli Benediktsson, and Johannes R. Sveinsson, *Random forests for land cover classification*, Pattern Recogn. Lett. **27** (2006), no. 4, 294–300.

[25] Nihal Fatma Guler and Sabri Kocer, *Classification of emg signals using pca and fft.*, J Med Syst **29** (2005), no. 3, 241–250 (eng).

[26] H. Guo and S.B. Gelfand, *Classification trees with neural network feature extraction*, Neural Networks, IEEE Transactions on **3** (1992), no. 6, 923–933.

[27] Aria Haghighi, *Numerical optimization: Understanding l-bfgs*, (December 2014).

[28] L. K. Hansen and P. Salamon, *Neural network ensembles*, IEEE Trans. Pattern Anal. Mach. Intell. **12** (1990), no. 10, 993–1001.

[29] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: Data mining, inference, and prediction, second edition*, Springer Series in Statistics, Springer, 2009.

[30] O. Irsoy, O.T. Yildiz, and E. Alpaydin, *Soft decision trees*, Pattern Recognition (ICPR), 2012 21st International Conference on, Nov 2012, pp. 1819–1822.

[31] Ian Jolliffe, *Principal component analysis*, Wiley Online Library, 2002.

[32] Ron Kohavi, Ron Kohavi, and Ross" Quinlan, *Decision tree discovery*, IN HANDBOOK OF DATA MINING AND KNOWLEDGE DISCOVERY **6** (1999), 267–276.

[33] Zhijun Li, Baocheng Wang, Chenguang Yang, Qing Xie, and Chun-Yi Su, *Boosting-based emg patterns classification scheme for robustness enhancement.*, IEEE J Biomed Health Inform **17** (2013), no. 3, 545–552 (eng).

[34] Andy Liaw and Matthew Wiener, *Classification and regression by randomforest*, R News **2** (2002), no. 3, 18–22.

[35] M. Lichman, *UCI machine learning repository*, 2013.

[36] R.P. Lippmann, *An introduction to computing with neural nets*, ASSP Magazine, IEEE **4** (1987), no. 2, 4–22.

[37] Dong C Liu and Jorge Nocedal, *On the limited memory bfgs method for large scale optimization*, Mathematical programming **45** (1989), no. 1-3, 503–528.

[38] Barrett Lowe and Arun Kulkarni, *Multispectral image analysis using random forest*, International Journal on Soft Computing (IJSC) **6**, no. 1.

[39] Lawrence Mitchell, *A parallel random forest implementation for r*, Technical report, EPCC (2011).

[40] R. Molinari and A. El-Jaroudi, *Complex decision boundaries using tree connected single layer nets*, Circuits and Systems, 1992., Proceedings of the 35th Midwest Symposium on, Aug 1992, pp. 883–886 vol.2.

[41] John C Nash, *Compact numerical methods for computers: linear algebra and function minimisation*, CRC Press, 1990.

[42] Andrew Y. Ng, *Feature selection, l1 vs. l2 regularization, and rotational invariance*, Proceedings of the Twenty-first International Conference on Machine Learning (New York, NY, USA), ICML '04, ACM, 2004, pp. 78–.

[43] David Opitz and Richard Maclin, *Popular ensemble methods: an empirical study*, Journal of Artificial Intelligence Research **11** (1999), 169–198.

[44] Max Ortiz-Catalan, Rickard Branemark, and Bo Hakansson, *Biopatrec: A modular research platform for the control of artificial limbs based on pattern recognition algorithms*, Source Code for Biology and Medicine **8** (2013), no. 1, 11.

[45] Travis Peters, *An assessment of single-channel emg sensing for gestural input*, 2014.

[46] Angkoon Phinyomark, Pornchai Phukpattaranont, and Chusak Limsakul, *Feature reduction and selection for emg signal classification*, Expert Syst. Appl. **39** (2012), no. 8, 7420–7431.

[47] Tomaso Poggio and Federico Girosi, *Networks for approximation and learning*, Proceedings of the IEEE **78** (1990), no. 9, 1481–1497.

[48] J. Ross Quinlan, *Simplifying decision trees*, International journal of man-machine studies **27** (1987), no. 3, 221–234.

[49] J.R. Quinlan, *Induction of decision trees*, Machine Learning **1** (1986), no. 1, 81–106.

[50] MBI Raez, MS Hussain, and F Mohd-Yasin, *Techniques of emg signal analysis: detection, processing, classification and applications*, Biological Procedures Online **8** (2006), 11–35.

[51] S. Rota Bulo and P. Kontschieder, *Neural decision forests for semantic image labelling*, Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on, June 2014, pp. 81–88.

[52] Cynthia Rudin, *15.097 prediction: Machine learning and statistics, spring 2012*, (Massachusetts Institute of Technology: MIT OpenCourseWare), http://ocw.mit.edu (Accessed 12 Apr, 2016). License: Creative Commons BY-NC-SA, Massachusetts.

[53] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams, *Neurocomputing: Foundations of research*, MIT Press, Cambridge, MA, USA, 1988, pp. 696–699.

[54] J. Stromberg, J. Zrida, and Alf Isaksson, *Neural trees-using neural nets in a tree classifier structure*, Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on, Apr 1991, pp. 137–140 vol.1.

[55] Xueyan Tang, Yunhui Liu, Congyi Lv, and Dong Sun, *Hand Motion Classification Using a Multi-Channel Surface Electromyography Sensor*, SENSORS **12** (2012), no. 2, 1130–1147 (English).

[56] PAUL E. UTGOFF, *Perceptron trees: A case study in hybrid concept representations*, Connection Science **1** (1989), no. 4, 377–391.

[57] Paul E. Utgoff and Carla E. Brodley, *Linear machine decision trees*, Tech. report, 1991.

[58] W. N. Venables and B. D. Ripley, *Modern applied statistics with s*, fourth ed., Springer, New York, 2002, ISBN 0-387-95457-0.

[59] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol, *Extracting and composing robust features with denoising autoencoders*, Proceedings of the 25th International Conference on Machine Learning (New York, NY, USA), ICML '08, ACM, 2008, pp. 1096–1103.

[60] Gang Wang, Yanyan Zhang, and Jue Wang, *The analysis of surface emg signals with the wavelet-based correlation dimension method.*, Comput Math Methods Med **2014** (2014), 284308 (eng).

[61] Marvin N Wright and Andreas Ziegler, *ranger: A fast implementation of random forests for high dimensional data in c++ and r*, arXiv preprint arXiv:1508.04409 (2015).

[62] Olcay Taner Yldz, Olcay Taner Yldz, and Ethem" Alpaydn, *Univariate and multivariate decision trees.*

[63] Xiaorong Zhang, Yuhong Liu, Fan Zhang, Jin Ren, Y.L. Sun, Qing Yang, and He Huang, *On design and implementation of neural-machine interface for artificial legs*, Industrial Informatics, IEEE Transactions on **8** (2012), no. 2, 418–429.

[64] Xu Zhang, Xiang Chen, Yun Li, Vuokko Lantz, Kongqiao Wang, and Jihai Yang, *A Framework for Hand Gesture Recognition Based on Accelerometer and EMG Sensors*, IEEE TRANSACTIONS ON SYSTEMS MAN AND CYBERNETICS PART A-SYSTEMS AND HUMANS **41** (2011), no. 6, 1064–1076 (English).

[65] Hui Zou and Trevor Hastie, *Regularization and variable selection via the elastic net*, Journal of the Royal Statistical Society: Series B (Statistical Methodology) **67** (2005), no. 2, 301–320.