CLASSIFICATION TREES

by

Maya McNichol

A senior project submitted in partial fulfillment of the requirements
for graduation in Mathematics.

Whitman College
2018

ABSTRACT

A classification (or decision) tree is a predictive model used to analyze data. This method allows us to assign a classification to a data point, using a tree that was grown on data from the same measurement space. In this paper, we discuss the methodology for growing these trees. We review background material in probability and information theory, consider the accuracy of a tree, and discuss the advantages of random forests. Finally, we grow a tree using data about mushrooms and use this to classify mushrooms as either edible or poisonous.

ACKNOWLEDGEMENTS

This paper would not be possible with the support of Professor Doug Hundley. Thank you for the help feedback and guidance throughout this project. Thank you to Luke Adams for his many edits and comments on this paper, and thank you to Professor Patrick Keef for providing constant advice and encouragement. Finally, thank you to all of the math majors of 2018 for all the support and enthusiasm in our final year together.

Contents

1 Introduction **1**

2 Background **3**
2.1 Probability Theory 3
2.2 Information Theory 13

3 Classifiers **16**
3.1 Structure of the Trees 18
3.2 Impurity 20
3.3 Best Split 22
3.4 Splitting vs. Stopping 25
3.5 Class Assignment 26
3.6 Pruning 27
3.6.1 Iris Example 28

4 Post Training Analysis **31**
4.1 Error Estimation 32
4.2 Confusion Matrix 34

5 Forests **36**
5.1 Bagging 36
5.2 Ionosphere Example 37

6 Applications **39**

7 Further Research **44**

1 Introduction

We will be looking at classification trees as a way to analyze data. By the end of this project, we hope to understand the algorithms for building these decision trees as well as look at real world examples of where this method is useful. We will work to understand the algorithms behind building the decision trees by working in MATLab with data as well as looking at the mathematics behind the decisions, placing a strong emphasis on probability theory to build a strong foundation.

A classification (or decision) tree is a predictive model to classify data. Each new data point, which can be regarded as a vector with components, we wish to classify enters the tree at the root node. This node then splits based on the value of some component of the data point, sending the data point to a smaller node. The nodes are then repeatedly split until the data point ends up in a node with no further splitting. Whatever the classification of this node, we assign it to the data point. To initially grow a tree, we use a sample of data points whose classifications are known.

To gain a firm understanding of the algorithms used to build the trees, we will be looking at specific criteria. On one extreme, we could split the data repeatedly so that after the last split there was a single data point in each node. This would be computationally inefficient and would undermine the whole purpose of our model. Therefore we want to minimize our computations while maintaining an acceptable output. The first criterion is the

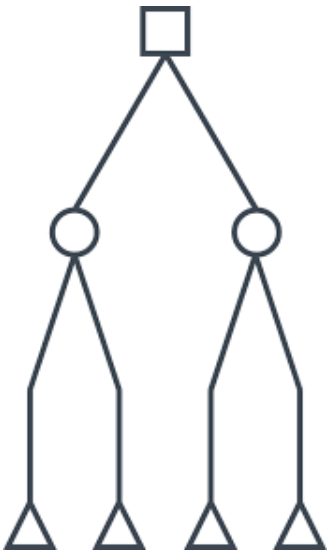


Figure 1: Structure of a generic classification (or decision) tree.

function that determines the 'best split' or the best partition of the data in a single node in order to minimize our total number of splits. The second major criterion is when to stop splitting and use the current partitions of the data to determine our classes. We also need to determine what is 'acceptable' in order to work with both of these criteria.

We will discuss methods to determine the accuracy of a decision tree as well as looking at a real world example. Lastly, we will discuss what happens when we put many tress together to make a forest. The resulting classifier can be compared to a single decision tree, both in the structure of the actually tree and its accuracy for classifying data. Overall, we hope to give a comprehensive overview of decision trees from the ground up.

2 Background

Before we grow a tree, we need some groundwork to get us on solid footing.

2.1 Probability Theory

We will begin our groundwork with an introduction to probability theory where we will use the notation from *A Primer on Information Theory, with Applications to Neuroscience* [3]. Consider tossing a regular 6-sided die. It will either result in 1, 2, 3, 4, 5, or 6. In this case, the set $\{1, 2, 3, 4, 5, 6\}$ form the sample space for the experiment. That is, the *sample space* is the set of all possible outcomes from an experiment, where an *outcome* is the result of a single experiment. An *event* is a set of outcomes (i.e. $\{1, 2, 3\}$ is the event that the outcome is less than 4). According to Pierre-Simon Laplace: "The probability of an event to occur is the number of cases favorable for the event divided by the number of total outcomes possible." In terms of our definitions, the *probability* for an event to occur is the number of outcomes in the sample space that correspond to that event divided by the size of the sample space. Throughout our discussion of probabilities, it is important to remember that the values are theoretical and are used to analyze data but cannot be directly observed in real life.

Thus, the probability for an event to occur is a number between 0 and 1, where 0 corresponds to the event never happening and 1 corresponding to the event definitely happening. Since at least one outcome must always occur,

the sum of the probability of each outcome in the sample space is 1. While we note the distinction between *discrete samples spaces* (like the 6-sided die described above) and *continuous sample spaces* (like $\mathbb{R}$) where the outcomes form a continuum, we will primarily be considering discrete sample spaces in this paper.

In order to formally define probability, we need to define a probability space. However, this relies on some knowledge of measure spaces which brings us farther from the point than necessary. Therefore, we suggest that the interested reader do more research on the subject while we use only the parts that we need.

We let a *probability space* be a triple $(\Omega, \mathcal{F}, P)$ where

- Ω is a nonempty set of individual outcomes,
- $\mathcal{F}$ is the set of events, where each event is a set containing 0 or more outcomes,
- P is a function called a *probability measure,* meaning that $P : \mathcal{F} \to [0, 1]$ such that
  - $P(\emptyset) = 0$,
  - if $E_i \in \mathcal{F}$ for $i = 1, 2, 3, ...$ are pairwise disjoint sets, then we find $P(\cup_i E_i) = \sum_i P(E_i)$,
  - noting that $\Omega \in \mathcal{F}$, we let $P(\Omega) = 1$.

Going back to our 6-sided die, we let $\Omega = \{1, 2, 3, 4, 5, 6\}$, the set of all outcomes, and $\mathcal{F} = \mathcal{P}(\Omega)$, the set of all events or all combinations of outcomes. The probability that we roll a 3 is $P(\{3\}) = \frac{1}{6}$ and the probability of rolling an odd number is

$$P(\{1, 3, 5\}) = P(\{1\}, \{3\}, \{5\}) = P(\{1\}) + P(\{3\}) + P(\{5\}) = \frac{1}{6} + \frac{1}{6} = \frac{1}{6} = \frac{1}{2}.$$

We can also look at the example of a fair coin toss. Here $\Omega = \{\text{heads, tails}\}$ and $\mathcal{F} = \{\emptyset, \{\text{heads}\}, \{\text{tails}\}, \Omega\}$. Since this is a fair coin, we find that

$$P(\emptyset) = 0, \quad P(\{\text{head}\}) = \frac{1}{2}, \quad P(\{\text{tails}\}) = \frac{1}{2}, \quad \text{and} \quad P(\Omega) = 1.$$

This makes sense because we would expect the chance of getting a heads or tails to be equal, while the probability that we get neither a heads or a tails is zero and the probability that we would get either a heads or a tails when flipping a coin is 1.

Now we will consider more than one event. Say there are two events we want to consider, A and B, where $A, B \in \mathcal{F}$. These two events are either independent or conditional. We will first consider independent events. Two events are independent if the occurrence of one does not effect the occurrence of the other. Suppose we saw a bird today and it is Friday. Since birds are around every day, not just certain days of the week, these two events are independent. Therefore, we can say that the probability that we saw a bird

today, $P(\text{bird})$, and the probability that it was Friday, $P(\text{Friday})$, is

$$P(\text{bird and Friday}) = P(\text{bird}) \cdot P(\text{Friday}).$$

To formalize this, we say two events A and B are *independent* if and only if $P(A \cap B) = P(A)P(B)$. Then we can define *conditional probability* such that the probability of A given B with $P(B) > 0$ is

$$P(A|B) = \frac{P(A \cap B)}{P(B)}. \tag{1}$$

Notice that if A and B are independent, then

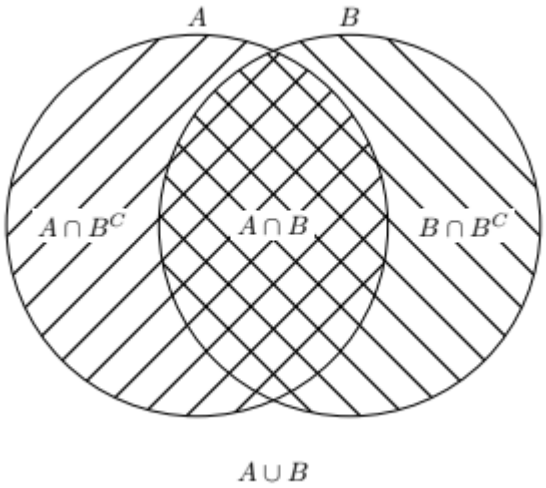
$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A)P(B)}{P(B)} = P(A).$$

Now that we have formally defined independent and conditional probabilities, we can look at Bayes' Theorem:

Theorem 2.1 (Bayes' Theorem)**.** If A and B are two events, in $\mathcal{F}$ with $P(B) > 0$, then the conditional probability of A given B is stated:

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}.$$

Proof. By Equation 1 we know $P(B \cap A) = P(A)P(B|A)$ and we also know $P(A \cap B) = P(B)P(A|B)$. Since $A \cap B = B \cap A$, we can equate the two



[H]

Figure 2: For the two events, A and B, we can observe their intersection and their union.

prior equations:

$$\begin{aligned} P(B)P(A|B) &= P(A)P(B|A) \\ P(A|B) &= \frac{P(A)P(A|B)}{P(B)}, \end{aligned}$$

completing the proof. □

To more concretely understand Bayes' Theorem (or Bayes' rule) consider the following example:

A student has some homework she needs to do. There is a 40% chance she will do her homework on Monday. If she does her homework on Monday, there is a 10% chance she does homework on Tuesday. If she does not do her homework on Monday, there is an 80% chance she will do it on Tuesday. She

is not a model student. If we let A be the event that she does her homework on Monday (and A' be the even that she does not) and B be the event that she does her homework on Tuesday, then:

$$P(A) = 0.4 \qquad P(A') = 0.6 \qquad P(B|A) = 0.1$$
$$P(B'|A) = 0.9 \quad P(B|A') = 0.8 \quad P(B'|A') = 0.2$$

Given what we know, we can calculate the probability that she did homework on Tuesday or $P(B)$, where the two disjoint possibilities are: "she did homework on Monday and she did homework on Tuesday" and "she did not do homework on Monday and she did homework on Tuesday." Then we find

$$\begin{aligned} P(B) &= P(B \cap A) + P(B \cap A') \\ &= P(A)P(B|A) + P(A')P(B|A') \\ &= 0.4 \cdot 0.1 + 0.6 \cdot 0.8 \\ &= 0.52 \\ &= 52\%. \end{aligned}$$

From here, we have enough information to calculate the probability that she did homework on Monday given that she did some on Tuesday. Using Bayes' rule with the values $P(B|A) = 0.1, P(A) = 0.4,$ and $P(B) = 0.52$, we find

$$P(A|B) = \frac{0.4 \cdot 0.1}{0.52} = 0.0769,$$

so there is a 7.69% that she did homework on Monday if she did on Tuesday.

Moving on to more probability theory, we will define a *random variable* X to be a function that maps every element in Ω to a number (generally an integer for discrete probabilities or real number for continuous) in Ω', the set of values corresponding to Ω. We will primarily consider real valued random variables that map an outcome $E \in \Omega$ to a number $X(E) \in \mathbb{R}$.

To illustrate this concept we will again use the example of a fair coin toss with $\Omega = \{\text{heads, tails}\}$, $\mathcal{F} = \{\emptyset, \{\text{head}\}, \{\text{tails}\}, \Omega\}$, where $\emptyset$ means that neither heads nor tails occurs and Ω means that either heads or tails occurs, and P is the probability distribution that assigns both heads and tails to the probability of $\frac{1}{2}$. One way to define a random variable X_1 is

$$X_1 = \begin{cases} 0 & \text{if heads} \\ 1 & \text{if tails.} \end{cases}$$

We can define another random variable on a similar set, Ω^3. Define X_2 as the number of heads in 3 tosses.

Given our random variables, we can define the probability distribution such that the probability distribution on the random variable is induced by the actual probability. For example, the distribution for X_1 induced from P is

$$P(X_1 = 0) = P(\{\text{heads}\}) = \frac{1}{2}$$

$$P(X_1 = 1) = P(\{\text{tails}\}) = \frac{1}{2}.$$

For the second example of a random variable, we can define the probability distribution based on the following outcomes:

	heads heads heads		$P(X_2 = 0) = \frac{1}{8}$
tails heads heads	heads heads tails	tails heads tails	$P(X_2 = 1) = \frac{3}{8}$
heads tails heads	tails tails heads	heads tails tails	$P(X_2 = 2) = \frac{3}{8}$
	tails tails tails		$P(X_2 = 3) = \frac{1}{8}$

We will assume that P, the probability function for $\mathcal{F}$, can be used on the random variable as well, in order to simplify notation and because they result in the same value. Thus, $P(X = x) = P(x)$.

Since our random variables take on integer or real values, we can use their total ordering to define a *cumulative distribution function* or *cdf* corresponding to X such that $F(x) := P(X \leq x)$, where for continuous probabilities

$$P(X \leq x) = \int_{\tau \leq x} P(X = \tau) d\tau$$

and for discrete probabilities

$$P(X \leq x) = \sum_{k \leq x} P(X = k).$$

We will now consider two important properties of random variables: the *expected value* and the *variance*. The expected value, or expectation, for the

random variable is the theoretical mean and is given by

$$E[X] := \int_{\mathbb{R}} xP(x)dx,$$

for continuous probability distributions and

$$E[X] := \sum_{x\in\mathbb{Z}} xP(x),$$

for discrete probabilities. This is the average value of the random variable, X, with each numerical value weighted according to the probability density function. Next, we define the variance of X, which is a measure for how far the values of the random variable are spread from the expectation value, to be

$$\sigma^2 = \mathrm{var}[X] := E[(E[X] - X)^2].$$

While these are both very important terms, in this paper we will generally consider the applications of these: the *sample mean* and *sample variance.* The sample mean is the observed mean for a set of data and can be calculated

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i.$$

The sample variance is the measure of the observed distance between the

observed values and the sample mean and can be calculated

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(\bar{x} - x_i)^2.$$

Looking back at some of our examples, we can calculate the expectation value and variance for the coin toss:

$$E(X) = \frac{1}{2}\cdot 0 + \frac{1}{2}\cdot 1 = \frac{1}{2}$$

and

$$\mathrm{var}(X) = \frac{1}{2}(0 - \frac{1}{2})^2 + \frac{1}{2}(1 - \frac{1}{2})^2 = \frac{1}{4}.$$

For the 6-sided die example, we find:

$$E(X) = \frac{1+2+3+4+5+6}{6} = 3.5$$

and

$$\mathrm{var}(X) = \frac{91}{6} - \frac{49}{4} \approx 2.92.$$

It is worth noting again, that many of the terms we discuss are purely theoretical values and cannot be directly observed. For example, we cannot roll a 3.5 with one roll of a standard 6-sided die. However, these values can often be estimated by taking sample measurements of a random process.

2.2 Information Theory

We need some sort of measure to determine the information we gain from some event. We can read information in a newspaper or observe some event. Reading about an earthquake in California does not give us an much information as an earthquake on the East Coast, where earthquakes are much less likely. Therefore events that are less common provide more information. Also, consider reading a three news paper articles: one about a shoe company, and two about a city council election. Chances are, some of the information in one of the articles about the city council election was also in the other, while the information in the article about the shoe company was completely different. Therefore, independent events should add information while events with elements in common should be sub-additive. Lastly, we do not lose information so the amount of information we gain from an event should never be negative. We will use these three rules to define a function $h : \mathcal{F} \to \mathbb{R}^+$ such that:

- For all events $E_1, E_2 \in \mathcal{F}$, we find
  - $h(E_1 \cap E_2) \leq h(E_1) + h(E_2)$,
  - $h(E_1 \cap E_2) = h(E_1) + h(E_2)$ if and only if E_1 and E_2 are independent.
- $h(x) = 0$ for all $x \in \mathcal{F}$ with $P(x) = 1$, meaning that if an outcome is a 'sure thing,' we gain no information from it.

- The function h is continuous and decreasing with respect to the probability distribution function P, meaning the more unexpected the event, the higher the information from it. As the probability decreases, the information increases.

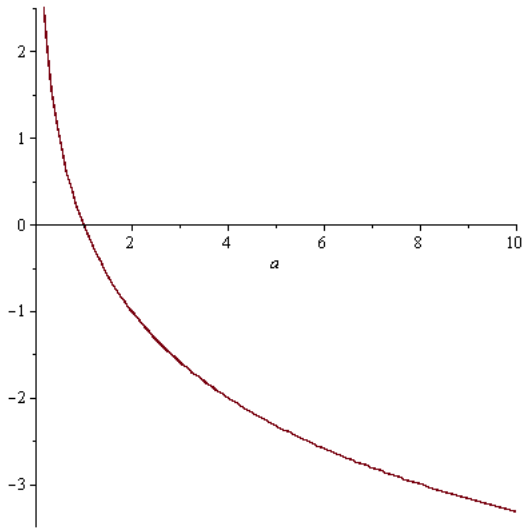


Figure 3: Negative base 2 logarithmic function.

We notice that a negative logarithmic function fits each of these conditions if we restrict the domain between to 0 and 1. We know that for independent events E_1 and E_2,

$$P(E_1 \cap E_2) = P(E_1)P(E_2).$$

Then we can observe

$$\log(P(E_1 \cap E_2)) = \log(P(E_1)P(E_2)) = \log(P(E_1)) + \log(P(E_2)),$$

revealing that the information from independent events is the sum each information from each event, as desired.

It is important to note that we are not attempting to define any one function. We are using the above statements as properties we need in an information function. Therefore, we can use this to mathematically define information.

Definition 2.1. Let $(\Omega, \mathcal{F}, P)$ be a triple. Then the *information* h of an event $E \in \mathcal{F}$ is

$$h(E) := h(P(E)) = -\log_b(P(E)).$$

We will use $b = 2$ as it is convenient to store information in 'bits.'

Let us again go back to the 6-sided die. Since any roll provides us with the same amount of information, we find that for $x \in \{1, 2, 3, 4, 5, 6\}$, the information gain $h(x) = -\log_2(\frac{1}{6}) = 2.58$. In our 2 bit system, this is the number of bits we need to store this information.

We can also consider the information as the smallest number of yes or no questions we need to ask to determine what event occurred. We could first ask if the roll was less that or equal to 3. Either answer gets rid of one half of the possible events. Then we only have 3 events to choose from. Thus, on average, it would take about 2.58 questions to determine which number was rolled.

Now that we have defined information, we can consider the expected value of information for an event, or the average of the information. We call this

the *entropy.*

Definition 2.2. Let X be a random variable on a triple $(\Omega, \mathcal{F}, P)$. Then the *entropy* is

$$H(X) := E[h(X)] = \sum_{j=1}^{J} -p_j \log_2(p_j),$$

where $p_1, ..., p_J$ is the set of discrete probabilities of events in $\mathcal{F}$.

The entropy is the amount of information we expect to gain from looking at the realizations of the random variable X.

This completes the background discussion. More information and details can be found in [3].

3 Classifiers

In this section we will use the notation from *Classification And Regression Trees* [1]. Suppose you are doctor working in the emergency rooms and a patient comes in after having a heart attack. When this patient entered the emergency room, 19 variables were measured about the patient's age, blood pressure, and other information about their medical history. These variables were all measured within the first 24 hours of observation. We can use this data to classify the patients as high or low risk, where high risk patients tend to not survive more more than 30 days. The aim of this study was to create an accurate classifier. This is a useful tool because, not only can we accurately and quickly classify patient, but we can discover which variables

Figure 4: Classification tree for heart attack patients.

are needed to predict an outcome. We will discuss how such a classification tree is produced, but for now we can see that such a tree was created using only two of the measured variables, as shown in Figure 4

Suppose we have a set of measurements. We can refer to the information about one patients as $(x_1, x_2, x_3, ..., x_k)$, where if we were referring to the medical study above, k would be 19. We call $\mathbf{x} = (x_1, x_2, x_3, ..., x_k)$ the *measurement vector* and the *measurement space* X is the collection of all possible measurement vectors. Referring to the same medical study, X is a 19-dimensional space. Each dimension in the measurement vector can either be *ordered* if the values are real number or *categorical* if the values are from a finite set with no natural ordering. From the previous example, we suppose that the 1st dimension, from which the variable x_1 comes, is age, an ordered variable since the values come from the integers. The second dimension could

be the whether or not the patient has experienced a prior heart attack; in this case x_2 is either yes or no, a categorical variable.

In the medical study, a patient was classified as either high risk or low risk: one of two classes. This is not always the case, so we will let C be the set of classes such that $C = \{1, 2, 3, ..., J\}$, when there are J classes. We want a rule that assigns each $\mathbf{x} \in X$ to some class in C.

Definition 3.1. [1] A *classifier* or *classification rule* is a function $d : X \to C$ such that for every $\mathbf{x} \in X$, we find $d(\mathbf{x}) = j$ for some $j \in C$.

We can also consider a classification as a partition of X where $X = \bigcup_j A_j$ and each $A_j = \{\mathbf{x} | d(\mathbf{x}) = j\}$.

It is important to note that classifiers are constructed using past data, not created out of thin air. The data that we use to construct a classifier is called a *learning sample*. The learning sample for the heart attack patients was the past data from 215 patients including whether or not they survived the first 30 days, i.e. their classification. Mathematically, the learning sample, $\mathcal{L}$, is a set of ordered pairs $(\mathbf{x}_n, j_n)$ consisting of the measurement vector and its assigned class; that is, $\mathcal{L} = \{(\mathbf{x}_i, j_i) | \mathbf{x}_i \in X, j_i \in C\}$.

3.1 Structure of the Trees

In this work, we will concern ourselves only with binary trees. This means that given a data set t, each decision will split that set into two smaller sets, t_l and t_r, where $t = t_l \cup t_r$.

In order to construct a tree, the first thing we need to consider is how to use the learning sample $\mathcal{L}$ to make a set of binary splits on $\mathcal{L}$. We want a tree structure that eventually splits the whole learning sample into smaller subsets, each having only one class label contained in it. Before we get too far, we will define some terminology:

Definition 3.2. A *node* is a subset of the measurement space X, denoted t.

We begin with all of X, which we call the *root node,*. Next, we split the measurement space with binary splits, creating more and more nodes. A *split* is the resulting partition of a binary question that depends on only one variable from a measurement vector. When a node is split into two smaller nodes, the original node is referred to as the parent node, while the two smaller are referred to as the daughter nodes. Eventually we reach a point where splitting any further is not beneficial.

Definition 3.3. The nodes at the bottom of a tree, those that are not split into smaller nodes, are called the *terminal nodes*, or leaves, and every data point in that node is assigned to the same class.

The terminal nodes are important because when we put a data point with an unknown class into the tree in order to predict its class, the terminal node is where that label will be assigned. Therefore, choosing which nodes are terminal is critical to the accuracy the of tree.

The last thing to mention is that we can look at a *branch* of the tree, or a subtree. A branch begins with any single node and contains every node that

forms the partition of the initial node.

When growing a tree, we need it to act according to the following steps:

1. Begin with the entire learning sample and examine every possible split on every variable.

2. Select the split with the greatest decrease in impurity (which will be discussed in the sections entitled Impurity and Best Split) up to the stopping rule (which will be discussed in the Splitting vs. Stopping section)

3. Apply the split

4. Repeat these steps on the two child nodes

We will now look at these steps in more detail.

3.2 Impurity

An important property of a node relates to its elements. We want to know how 'pure' a node is when deciding whether or not to split it further, meaning how much are distinct classes mixed in that node.

Definition 3.4. Let t be a node. We define the *node proportion* $p(j|t)$, where $j \in C$, to be the proportion of the cases in t belonging to class j.

Given a node t, we will refer to the node proportion $p(j|t)$ as p_j. This definition is useful because it allows us to quantify the proportion of each class in a single node.

Next, we want to consider the impurity of a node. We can this of this as, given a set of data, what is the chance of being incorrect if we were to randomly assign a class to a randomly selected data point from that set?

Let us consider the two extremes. First, suppose all the the elements of the set have the same class label. Then there is no chance that we can incorrectly classify our data point since there is only one class label to choose from. On the other hand, suppose that we have n elements in our data set, each with a distinct class label. Then there is a $\frac{1}{n}$ of choosing correct label so a $1 - \frac{1}{n}$ chance of picking the incorrect label.

This is called the Gini Impurity measure, and we will refer to it as simply *impurity.* Given a node t with J classes and node proportions p_j where $j = 1, 2, ..., J$, there is a p_j chance of an element of class j being chosen and a $1 - p_j$ chance of incorrectly classifying it. Then we can formally define the impurity as follows:

Definition 3.5. The *impurity* of a node t is given as

$$i_t = \sum_{j=1}^{J} p_j(1 - p_j) = 1 - \sum_{j=1}^{J} p_j^2.$$

Now that we have the concept impurity, we can see that more impure a node, the less likely that we will assign it as a terminal node.

3.3 Best Split

When we want to split a node into two smaller nodes, we want a way to measure how good the split was. There are two ways we can do this. First, we can use the impurity. As we build the tree from the root node, we want to split each node in a way that decreases its impurity. We want the impurity of the parent node to be larger than that of the daughter nodes.

Letting t be the parent node and s the split that gives t_l and t_r as the daughter nodes, we want to find the split that give the greatest decrease in impurity. We do this by looking that the change in impurity, $\Delta i_t(s)$, for the split s of node t. We find that

$$\Delta i_t(s) = i_t - (i_{t_l} + i_{t_r}) \tag{2}$$

is the change in impurity of the given split. We want to find the split that gives the greatest decrease in impurity.

The second way to measure the goodness of a split uses the entropy. Recall from Definition 2.2 that the entropy in node t is given by

$$H(t) = \sum_{j=1}^{J} -p_j \log_2(p_j)$$

where p_j is the given node proportion. From here we can find the difference in entropy from the parent node to the daughter nodes. This is similar to what we calculated with impurity; we want the split that results in the largest

decrease in entropy. We will use a weighted average of the entropy from the daughter nodes to calculate the change, resulting in the following:

$$\Delta H = H(t) - \left(\frac{|t_l|}{|t|} H(t_l) + \frac{|t_r|}{|t|} H(t_r) \right). \tag{3}$$

Let us look at both of these concepts through an example. Suppose our node t has 12 elements: 2 from class a, 4 from class b and 6 from class c. The node t is split into t_l and t_r, where t_l has 7 elements (2 from class a, 3 from class b and 1 from class c) and t_r has 5 elements (5 from class c).

We can calculate the impurity of t:

$$i_t = 1 - \sum_{j=1}^{J} p_j^2 = 1 - \left(\frac{2}{12} \right)^2 - \left(\frac{4}{12} \right)^2 - \left(\frac{6}{12} \right)^2 = \frac{11}{18}.$$

Similarly, we find that $i_{t_l} = 4/7$ and $i_{t_r} = 0$, since there is only one label in t_r. Now we can calculate the change on impurity of our split:

$$\Delta i_t(s) = i_t - i_{t_l} - i_{t_r} = \frac{11}{18} - \frac{4}{7} - 0 = \frac{5}{126},$$

revealing that the impurity decreased.

We can also calculate the change in entropy from the split. We find that

$$\begin{aligned} H(t) &= -\frac{2}{12}\log_2\left(\frac{2}{12}\right) - \frac{4}{12}\log_2\left(\frac{4}{12}\right) - \frac{6}{12}\log_2\left(\frac{6}{12}\right) \approx 1.46, \\ H(t_l) &= -\frac{2}{7}\log_2\left(\frac{2}{7}\right) - \frac{4}{7}\log_2\left(\frac{4}{7}\right) - \frac{1}{7}\log_2\left(\frac{1}{7}\right) \approx 0.98, \\ H(t_r) &= -\frac{5}{5}\log_2\left(\frac{5}{5}\right) = 0. \end{aligned}$$

Using Equation 3, we find

$$\Delta H = H(t) - \left(\frac{|t_l|}{|t|}H(t_l) + \frac{|t_r|}{|t|}H(t_r)\right) = 1.46 - \left(\frac{7}{12}\cdot 0.98 + \frac{5}{12}\cdot 0\right) = 0.48,$$

revealing that the entropy also decreased after the split.

Now that we have defined these two methods and explored both methods through an example, we will focus our attention to the impurity. To find the best split for each node, we must look at every possible split and pick the one with the largest decrease in impurity.

In order to formally define the best split, we need the notion of the set of all possible splits of a node t. The set of splits, S, is the set of all yes or no questions that can be asked about a single variable. Then we define the best split as follows:

Definition 3.6. Let $s^* \in S$. Then s^* is the *best split* when

$$\Delta i_t(s^*) = \max\{\Delta i_t(s) | s \in S\}.$$

We want to find the question about a specific variable such that a split of the node based on how each measurement vector in that node responds to the question, results in the maximum decrease in impurity.

3.4 Splitting vs. Stopping

Once we have split the root node into smaller nodes and the tree has grown, we want a way to decide when it is big enough. Thus, we need a measure that tells us when to stopping splitting a node.

If a node is completely pure, it only contains measurement vectors of a single class; splitting it further would only add to the complexity of the tree without increasing accuracy at all. Therefore, if a node is pure, we stop splitting on that node.

There are three other criteria we use to decide when to stop splitting an impure node. These are numerical values that we set to avoid over-fitting the learning sample. Over fitting is when the tree grows so big that the terminal nodes are so small that the tree has basically memorized the data. This leads to more error when trying to predict the class of an unknown measurement vector. To avoid this, we use these three numerical restrictions:

1. A minimum parent size. When a node has less than this number of cases from the learning sample in it, we do not split it further.

2. A minimum leaf size. When a split will create a daughter node with fewer than this number of cases from the learning sample, we do not

make that split.

3. A maximum number of splits. When the tree has more than the maximum number of splits, we do not split further.

Whenever any of these three conditions or the condition that the node is pure are met, we stop splitting that node and call it a terminal node.

3.5 Class Assignment

Once we have built a tree and have a set of terminal nodes, we need to assign each terminal node a class. It is simple when the terminal node is pure; we assign the only class that occurs in that node to every measurement vector in that node.

If more that one class occur in the terminal node, we assign every measurement vector in that node to the class with the highest node proportion. For example, if we had 2 elements from class a, 4 from class b, and 6 from class c, we would assign the node to class c. If there is more that one class with equal and highest node proportion, we randomly choose between those classes. Whichever class is randomly selected is assigned to every measurement vector in the terminal node.

Going back to the heart attack example, consider the split based on the systolic blood pressure. For patients with a systolic blood pressure greater than or equal to 91, the node proportion was higher for high risk patients than low risk. Thus, that terminal node was classified as high risk. The same

comparison of node proportions was done for the other two terminal nodes, resulting in the tree shown in Figure 4.

3.6 Pruning

When growing a decision tree, we can grow a large, intricate tree with smaller node sizes and more nodes, or we can grow a smaller, simpler tree. In theory, we can grow a tree in which every terminal node only has one element from the learning sample. As we mentioned before, this tree would over fit and essentially memorize the learning sample. This would result in a highly accurate tree when tested with the same learning sample, but when tested with new data, the accuracy would be much lower. On the other hand, we can grow a smaller tree with less splits and fewer nodes. This tree will seem less accurate when tested on the same learning sample but will maintain that accuracy when tested on a new data set; the tree will be more robust.

There are two ways to adjust the size of our trees. The first is by adjusting the parameters we used for the stopping criteria. Recall, we used minimum parent size, minimum leaf size, and maximum number of splits to determine when to stop growing the tree. If we were to decrease the minimum parent size and minimum leaf size and increase the maximum number of splits, the tree would grow larger before reaching any of these cutoffs. This leads to a bigger, more complex tree. Conversely, if we were to increase the minimum parent size and minimum leaf size and decrease the maximum number of splits, we would get a smaller, simpler tree.

The second method is called *pruning.* Here, we set the three conditions to their extremes and grow a large tree. Consider a branch of that tree. We cut off the branch just below root node, making that root node a terminal node. Then use the class assignment rule to assign a class to that node; that is, assign the class the occurs with greatest frequency within that node.

We can do this to many levels of branches to achieve different pruning levels. These will all result in trees of varying complexity. To choose the best pruned tree, we will test all possible pruning levels using cross validation (see Error Estimation section) and choose the one with the smallest error.

3.6.1 Iris Example

To better understand pruning, we will look at an example about iris flowers. We have a set of data about three species of irises from MATLAB. Each measurement has four predictors: sepal length, sepal width, petal length and petal width. The sepal is the small green piece that encases the flower bud. Based on these predictors, each flower is classified as one of three iris species: virginica, setosa, or versicolor. Using MATLAB to build a complex tree, we get Figure 5.

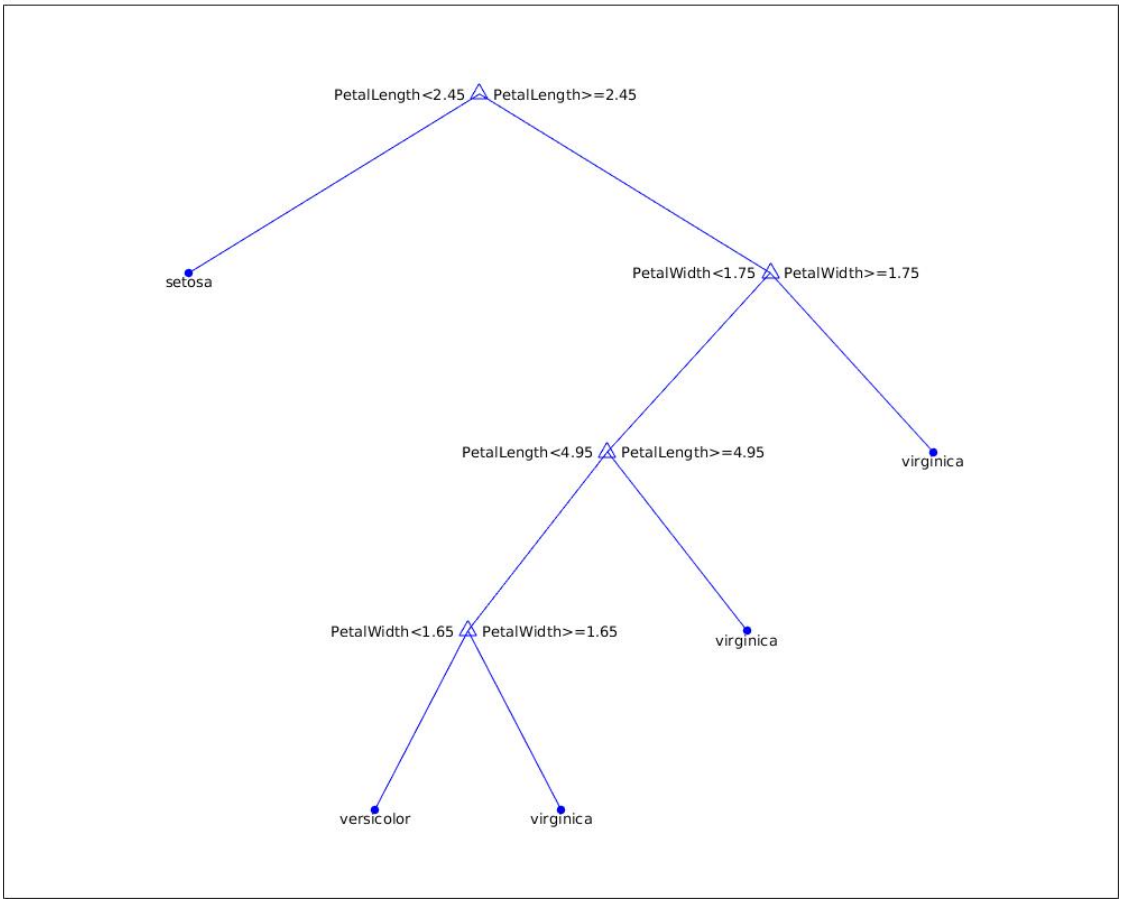


Figure 5: Classification tree for the iris data.

This tree has four levels of pruning. We can graph the cross-validated error vs. the level of pruning to see where the error is the smallest. We find that three is the best level of pruning for this tree.

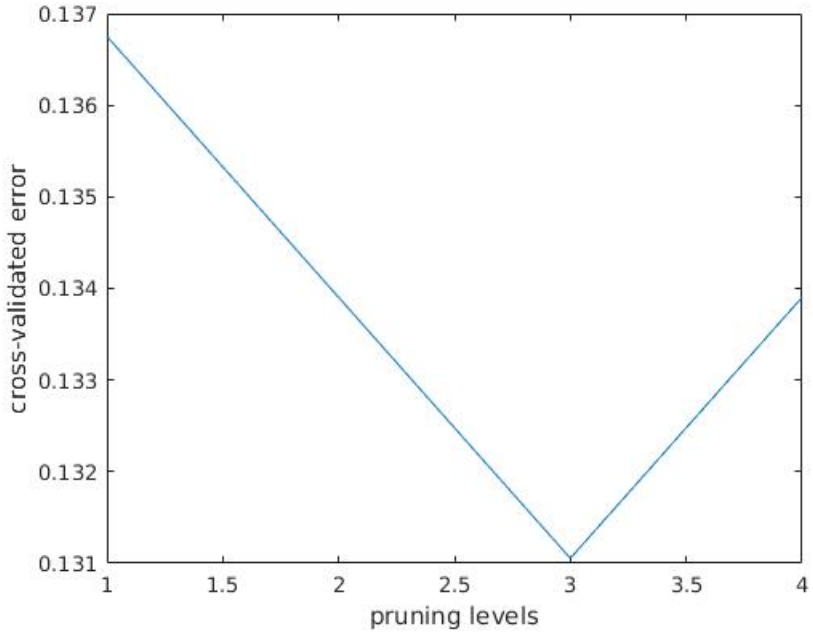


Figure 6: Plot of pruning level versus error for the iris tree.

Going back to the first method of controlling the tree complexity, we can look at the accuracy of the tree versus the minimum leaf size.

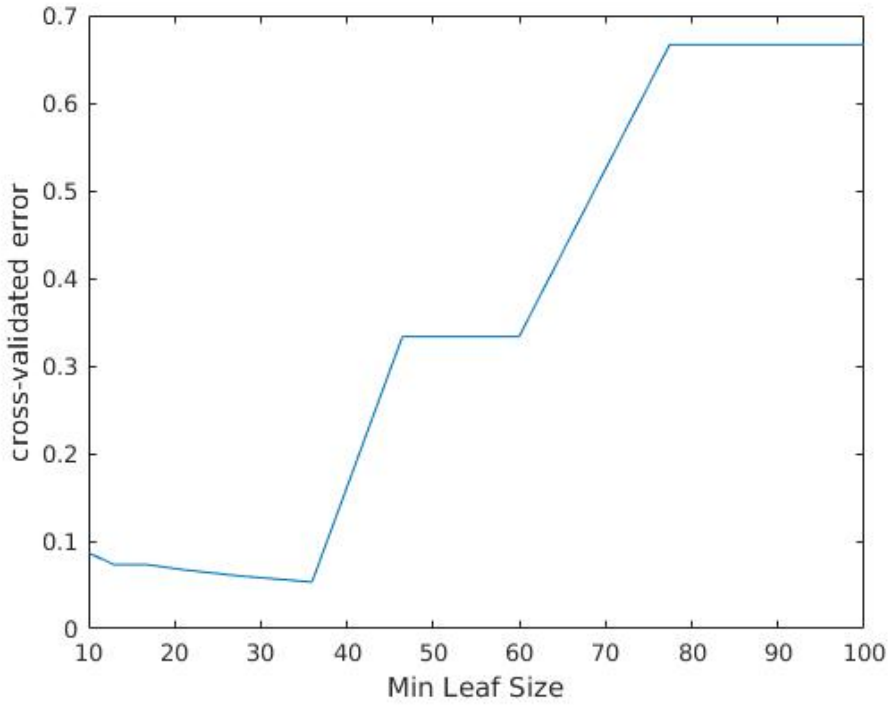


Figure 7: Plot of the minimum leafs size versus error for the iris tree.

From here, we can see that, keeping all the other parameters constant, the most accurate tree occurs when the minimum leaf size is between 30 and 40.

4 Post Training Analysis

After we grow a tree, it is useful to know how accurate that tree is at classifying new data. We often want to know the percent of cases that were misclassified. However, we may also want to see which cases were misclassi-

fied and what they were misclassified as. We will look at these two problems separately.

4.1 Error Estimation

When trying to determine the overall misclassification rate, we need to know the class of every case in the measurement space. Unfortunately, we often do not have more data with known classes (much less the whole measurement space), since we used the known data to grow the tree. Therefore, we have to find a way to estimate the error.

One way to do this is using the *resubstitution* estimate. This is the difference between the known class of a measurement vector and the class assigned by the tree. The higher the resubstitution error, the more cases that were misclassified and the less accurate the tree. This method tends to be overly optimistic in its error estimation, due to the fact that it is being test on the same data that trained it.

To more formally define the resubstitution estimate, suppose we have a learning sample of size N with J classes and a classification rule $d(\mathbf{x})$. Letting $\mathbf{x}_n$ be a measurement vector from the learning sample and j_n be its corresponding class from the set of J classes, we define the following:

Definition 4.1. Let $\mathcal{X}(\cdot)$ be 1 when the statement inside the parentheses is

true and 0 otherwise. Then the *resubstitution estimate*, $R(d)$, is

$$R(d) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{X}(d(\mathbf{x}_n) \neq j_n).$$

An example of an extreme case is letting A_j include all vectors $\mathbf{x}_n \in \mathcal{L}$ with $j_n = j$ and all other vectors in X are randomly assigned to one or the other of A_j. This gives us a resubstitution error $R(d) = 0$. However, it is probably not the case that the actual accuracy of the tree is perfect.

Another type is called the *test sample* estimate. Here, we partition the learning sample into two subsets $\mathcal{L}_1$ and $\mathcal{L}_2$, making sure that the cases in each subset are independent of each other. We use $\mathcal{L}_1$ to construct the classifier d and then use $\mathcal{L}_2$ to estimate the accuracy of the tree. Letting N_2 be the number of cases in $\mathcal{L}_2$, the test sample estimate is

$$R^{ts}(d) = \frac{1}{N_2} \sum_{(\mathbf{x}_n, j_n) \in \mathcal{L}_2} \mathcal{X}(d(\mathbf{x}_n) \neq j_n).$$

Although there is no theoretical justification for this, the common practice is to let $\mathcal{L}_1$ be $\frac{2}{3}$ of $\mathcal{L}$ and $\mathcal{L}_2$ be $\frac{1}{3}$ of $\mathcal{L}$. One of the drawbacks of this estimation is that it limits the sample size used to construct the tree. Therefore, this is a good method to use for a large data set.

For smaller data sets, another method called *V-fold cross-validation* estimate can be used. Here, the elements of $\mathcal{L}$ are randomly partitioned into V subsets of roughly equal size $\mathcal{L}_1, \mathcal{L}_2, ..., \mathcal{L}_V$. We will construct V classifiers;

for each $v \in \{1, 2, ..., V\}$, we will construct a classifier $d^{(v)}(\mathbf{x})$ on $\mathcal{L} \setminus \mathcal{L}_v$ and use the remaining elements of $\mathcal{L}$, namely $\mathcal{L}_v$, to calculate the test sample estimate of the accuracy of the classifier $d^{(v)}(\mathbf{x})$. We find

$$R^{ts}(d^{(v)}) = \frac{1}{|\mathcal{L}_v|} \sum_{(\mathbf{x}_n, j_n) \in \mathcal{L}_v} \mathcal{X}(d^{(v)}(\mathbf{x}_n) \neq j_n).$$

Since each $d^{(v)}(\mathbf{x})$ classifier is built from almost all of $\mathcal{L}$, each of these estimate is approximately equal to the true accuracy of the tree. Using this, the cross-validation estimate is

$$R^{cv}(d) := \frac{1}{V} \sum_{v=1}^{V} R^{ts}(d^{(v)}).$$

This gives us several ways to estimate the accuracy of a tree.

4.2 Confusion Matrix

While knowing the overall accuracy of a tree is important, we might also want to consider how the cases were misclassified. In the heart attack example discussed previously, there is a crucial distinction on how the patients could be misclassified. If a patient who was actually low risk was misclassified as high risk, that would result in more monitoring and medical attention which would not put the patient in any danger. On the other hand, if a high risk patient were accidentally classified as low risk, resulting in less monitoring and medical attention, then the patient's life could be at risk. Therefore, we

want a way to understand how this misclassifications occur.

We do this using a confusion matrix. This is a matrix with the predicted class on the horizontal axis and the true class on the vertical axis. It allows us to see which classes were misclassified as which other classes.

Going back to the iris data from section 3.6.1, we can discuss the confusion matrix for the classification tree.

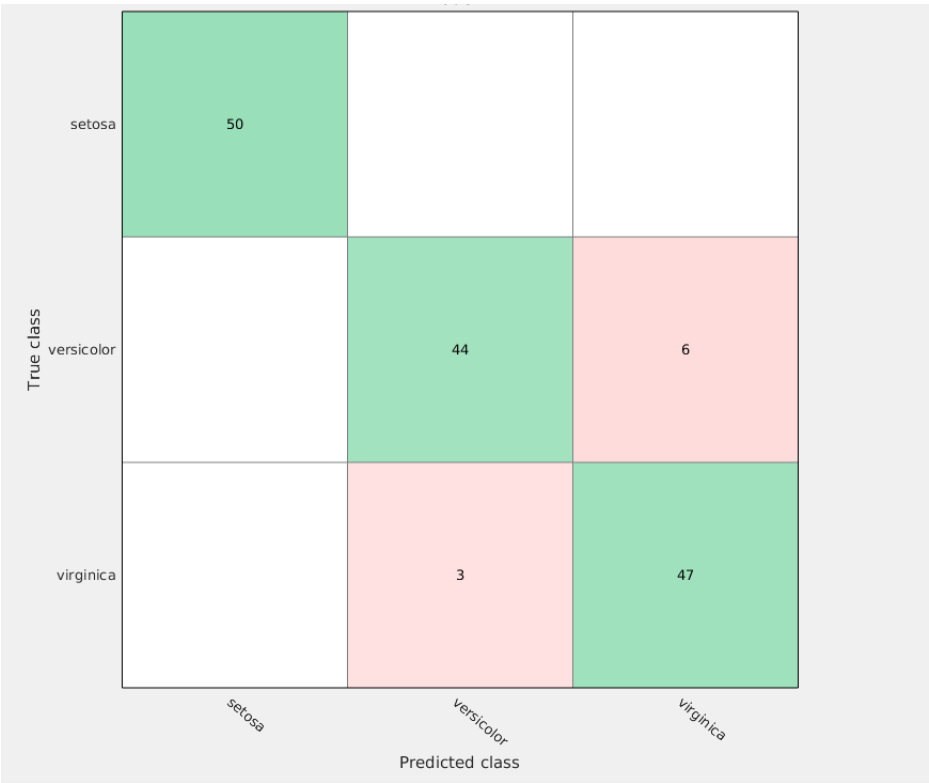


Figure 8: Confusion matrix for the iris classification tree.

From Figure 8, we can see that we classified 141 cases correctly. There were three cases we predicted were versicolor but were actually viginica and

six cases we predicted were virginica but were actually versicolor.

While the ramifications for misclassifying irises is minimal, imagine we had been predicting the species of mushrooms, where some are tasty and some are deadly.

5 Forests

Now that we understand and can grow trees, we will consider multiple trees, or a forest.

5.1 Bagging

Although there are a few different type of forests, we will only consider one method of using multiple trees. This is called *bootstrap aggregating* or *bagging*.

Suppose we have a data set where the correct classification of each measurement vector is known, and we call this the learning sample, $\mathcal{L}$. Let $|\mathcal{L}| = n$. Now we take a random sample from $\mathcal{L}$ that has approximately $2n/3$ cases and call this D_1, or a bag. Repeat this m times. We now have m subsets of $\mathcal{L}$, where $|D_m| \approx 2n/3$.

Using each D_i, we build a classification tree. Now, when we input an unknown measurement vector into the forest, each classification tree gives a class for that case. We let each tree have one vote and the class with the most votes is the class we assign to the measurement vector.

This method takes the emphasis off building a highly accurate tree but still results in an effective classifier.

5.2 Ionosphere Example

We will look at an example to see how a forest can be more accurate. We have data about the ionosphere built into MATLAB. There are 351 measurement vectors with 34 predictors. We want to classify each case as either a good (g) or bad (b) radar return. Using MATLAB, we can build a complex classification tree (Figure 9), a pruned tree (Figure 10), and a forest of 50 bagged trees.

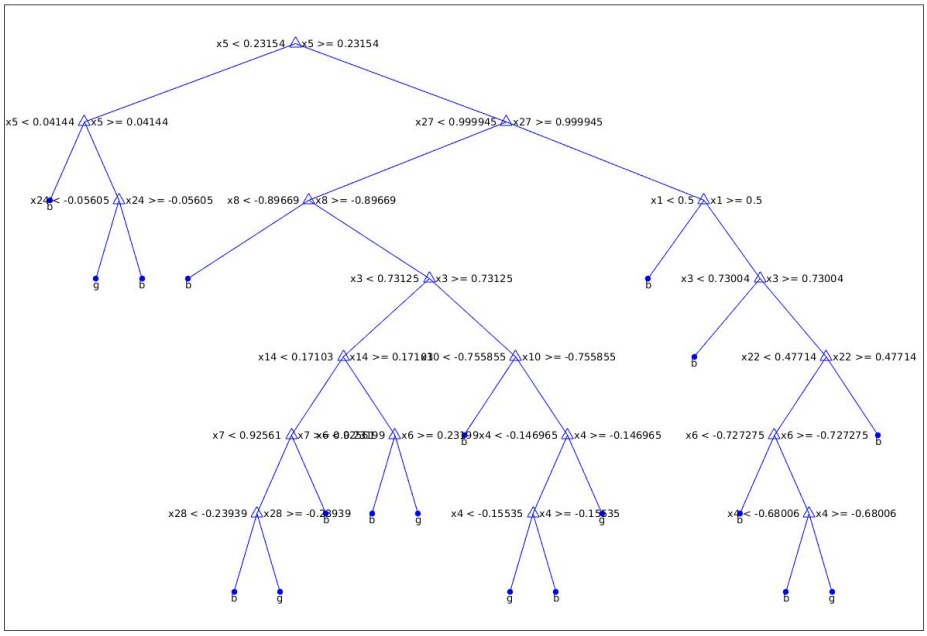


Figure 9: Complex classification tree for ionosphere data.

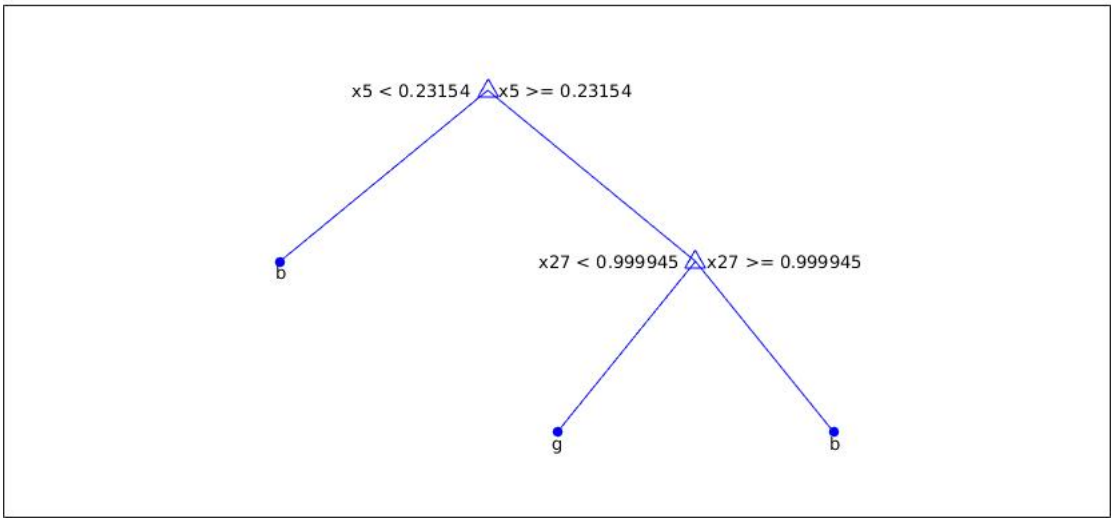


Figure 10: Pruned classification tree for ionosphere data.

We can then plot the out-of-bag classification error versus the number of trees grown for the forest. The out-of-bag classification error is the misclassification rate for a tree that is tested using the samples from $\mathcal{L}$ that were not in the bag used to build the tree. It is shown in [1] that using the out-of-bag samples to estimate the true misclassification rate is as accurate as using a test sample.

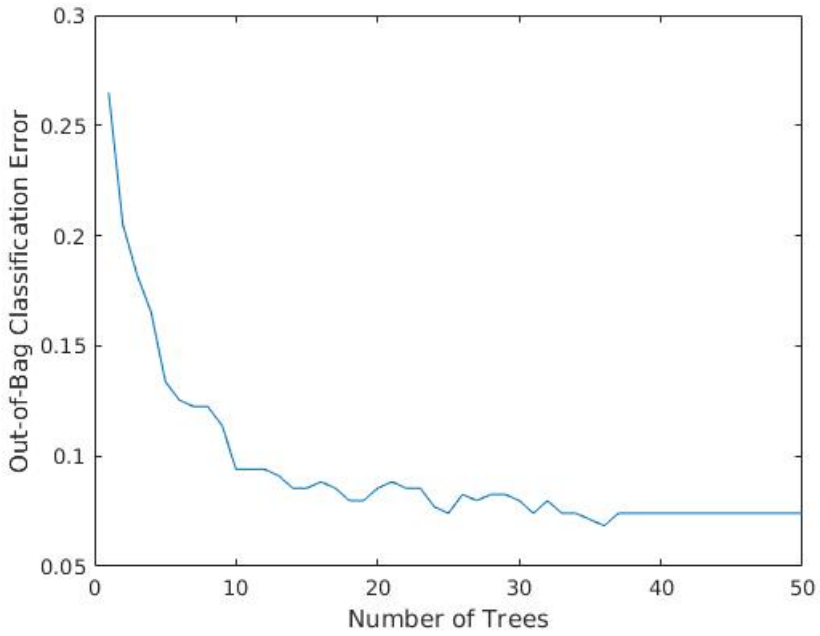


Figure 11: The out-of-bag classification error vs. number of trees.

As we can see, the more trees in the forest, the lower the error. That is, although the individual trees were not always highly accurate at classifying new data, when we put many of these trees together, we were able to build a classifier that could much more accurately predict the class of an unknown measurement vector.

6 Applications

By now, we understand the methods and motivations behind building trees. We will use what we have learned to build a classifier for mushrooms, classi-

fying each mushroom as either poisonous or edible.

This data comes from the Adubon Society Field Guide to North American Mushrooms [2]. In this data, there are 8124 cases and 22 categorical predictors. We hope to classify each mushrooms as either edible or poisonous.

We begin by building a simple tree in MATLAB.

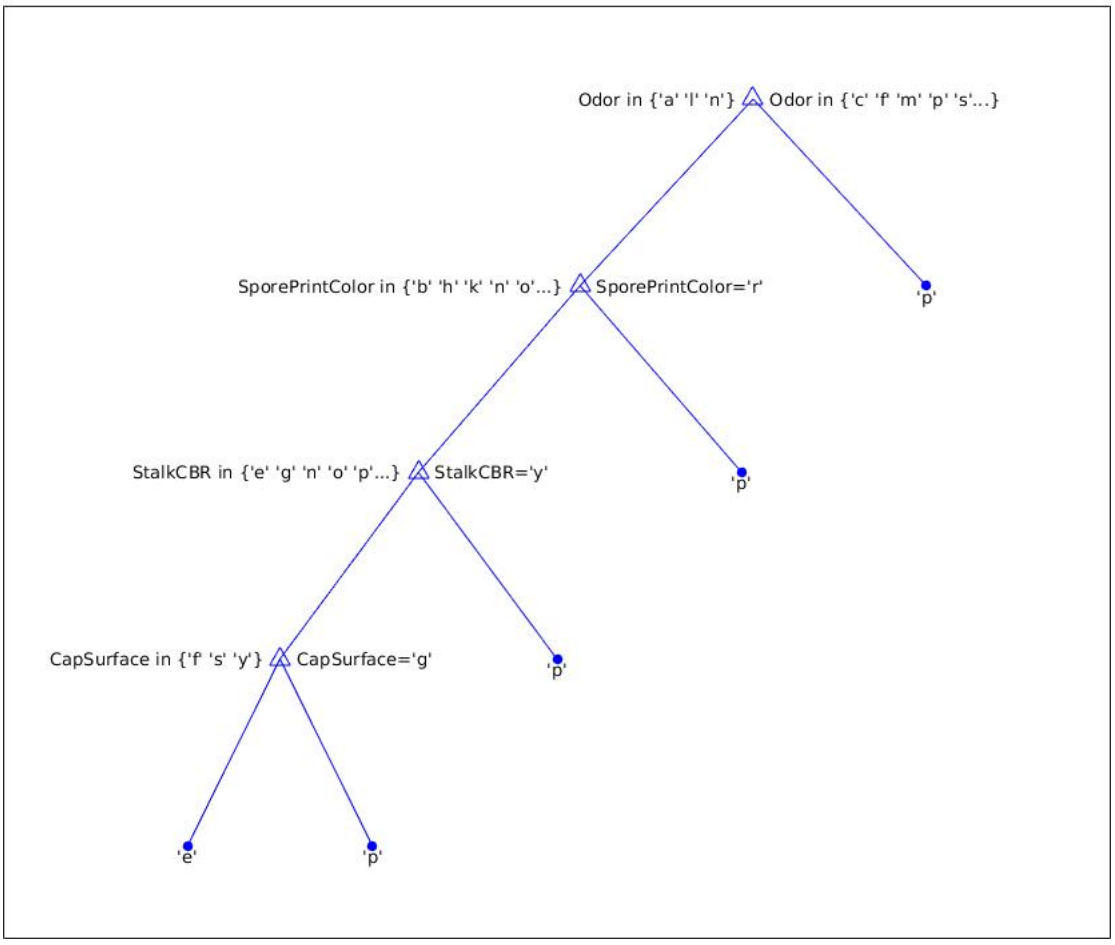


Figure 12: Simple tree for the mushroom data.

Here, the tree splits the data first according to the odor, then spore print color, stalk color below the ring, and finally cap surface. This gives a tree with 99.7% accuracy.

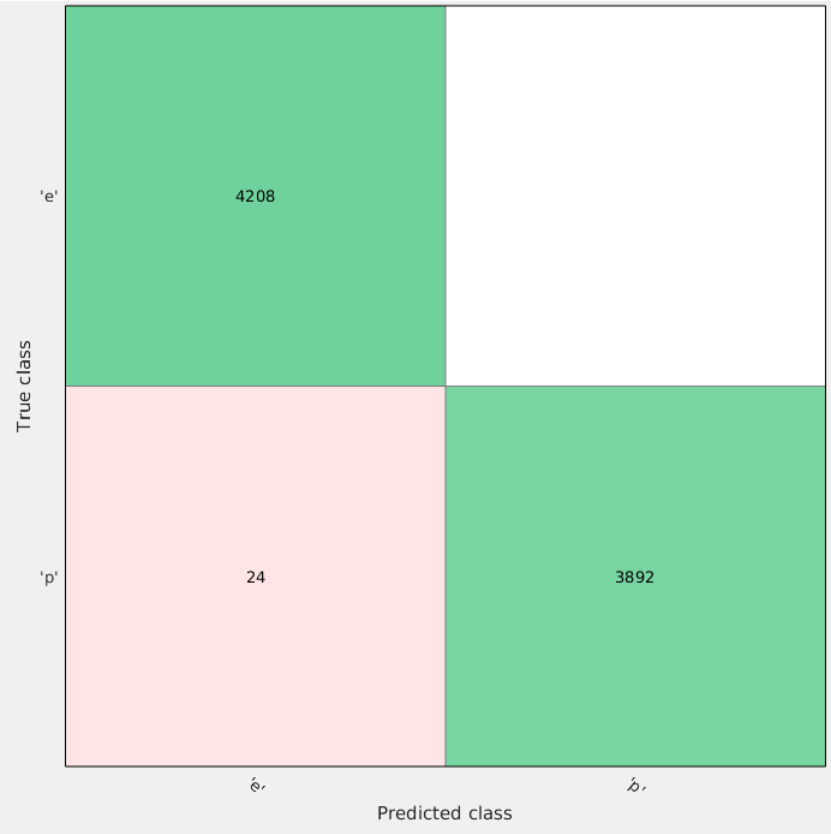


Figure 13: Confusion matrix for the simple mushroom tree.

This 99.7% accuracy appears to be a good result. However, looking at the confusion matrix, we can see that of the 24 cases that were misclassified, all of the were classified as edible but were actually poisonous. Therefore, this would not be a reliable way to determine if a mushroom were edible or

poisonous, because the 0.3% chance of being incorrect could be deadly.

We can build a more complex tree from the same data and we arrive at Figure 14.

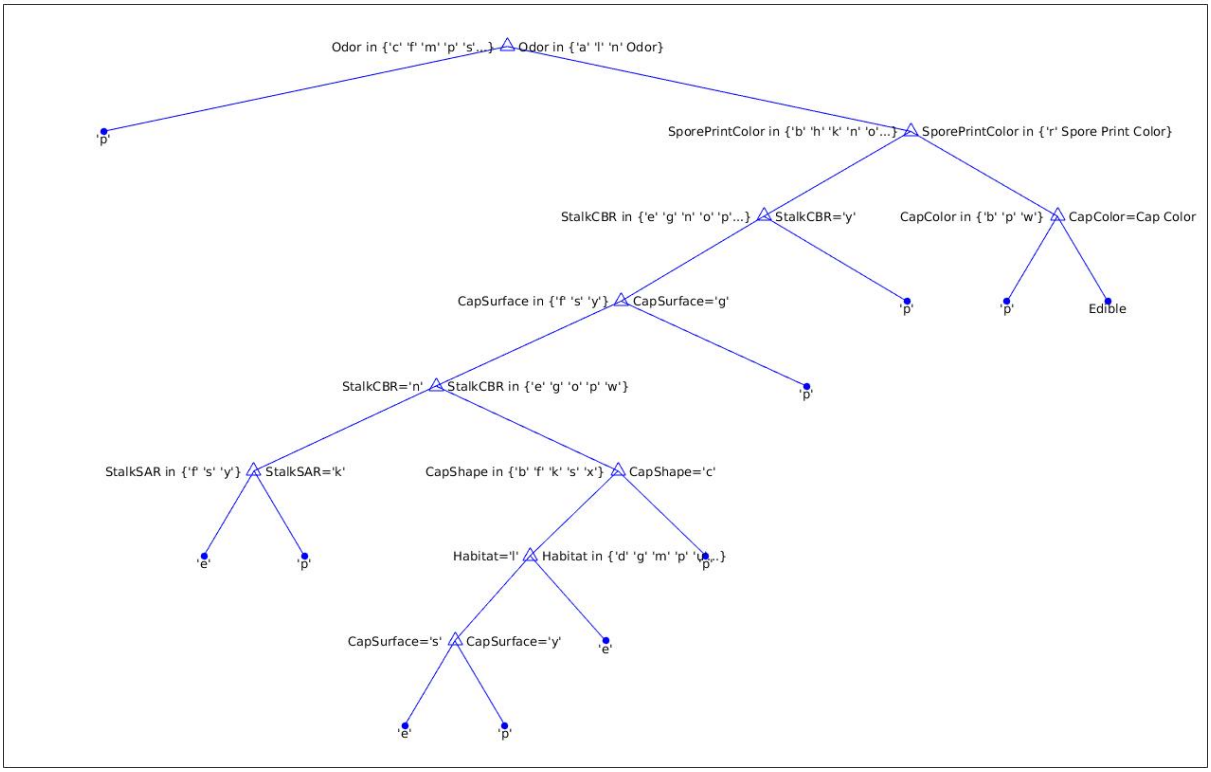


Figure 14: Complex tree for the mushroom data.

This tree is 100% accurate according to MATLAB. We can compare the confusion matrix of this tree to the confusion matrix in Figure 13.

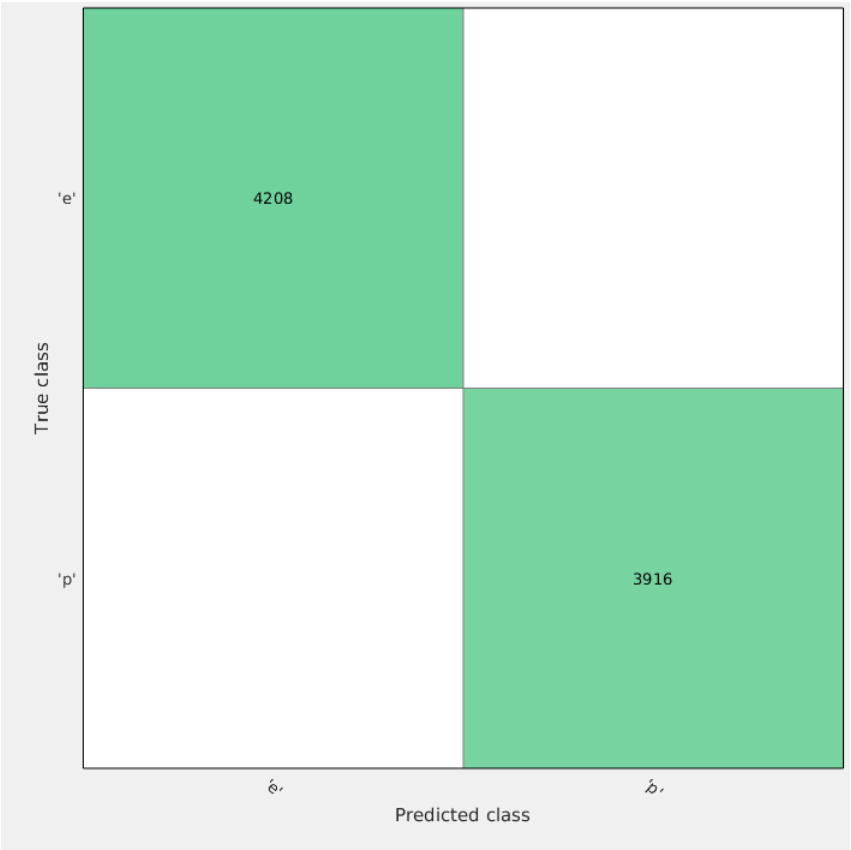


Figure 15: Confusion Matrix for the complex mushroom tree.

Here we can see that since there were no misclassified mushrooms, there were none accidentally classified as edible that were actually poisonous. Therefore, this would be a much better classifier to use if one were to eat there mushrooms (which we do not recommend).

The mushroom example illustrates why a decision tree can be useful. There are many reasons why classification trees are useful as a predictive model. One is the flexibility of the type of data, categorical (binary or not)

or numerical. Another is the easy graphical interpretation. We can represent the tree on a piece of paper, as we did with the mushrooms trees above, and we can use that tree to classify new cases; we do not have to imagine some n-dimensional space. We can use decision trees on any sort of general classification problem.

7 Further Research

We should now have a solid understanding of what a decision tree is and how we build one. We also looked at some basic probability and information theory in section 2 to create a solid foundation from which to build. Now, we will consider ways to improve our models.

In this paper, we only considered binary trees. To make our trees more accurate, we could split three or more times. However, as stated above, one of the advantages of decision trees is their ease of visual interpretation. Making our trees more complicated by adding more splits would make them less easily interpreted.

This same trade off is valid for forests versus single trees. While one tree is easy to visualize and even write down, things get more complicated when looking at multiple trees, especially when we build a forest on the order of 100 trees. There is always a balance between simplicity and accuracy that needs to be struck.

One last improvement that could be made is adding a cost to misclassi-

fications. While we want to minimize the overall misclassification rate, from looking at the mushroom example, we can see that some misclassifications are more consequential then others. Then, making it more 'expensive' to incorrectly classify a mushroom as edible than to incorrectly classifying one as poisonous, we could build a tree that would be less likely to misclassify in that way.

Decision trees can be used for any sort of classification problem. Despite their simplicity, decision trees prove to be robust and accurate, especially when built together into a forest. Overall, decision trees are an effective and efficient method of predictive analysis that can be visually interpreted and easily understood.

References

[1] Breiman, L., J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Boca Raton, FL: Chapman & Hall, 1984.

[2] Dua, D. and Karra Taniskidou, E. (2017). UCI Machine Learning Repository `http://archive.ics.uci.edu/ml`. Irvine, CA: University of California, School of Information and Computer Science.

[3] Effenberger F. "A Primer on Information Theory with Applications to Neuroscience." In: Rakocevic G., Djukic T., Filipovic N., Milutinović V. (eds) *Computational Medicine in Data Mining and Modeling,* April 2013. Springer, New York, NY.

[4] The Mathworks, Inc. (2018). *Statistics and Machine Learning Toolbox, R2018a.* Retrieved from `https://www.mathworks.com/help/pdf_doc/stats/`