Illuminating the Blackbox: Evaluating the use of Decision Trees in Enhancing Blackbox Model Interpretability

Pritika Mehra

Advisor: Mark Braverman

Submitted in partial fulfillment

of the requirements for the degree of

Bachelor of Science in Engineering

Department of Computer Science

Princeton University

May 2018

I hereby declare that I am the sole author of this thesis.

I authorize Princeton University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I pledge my honor that this represents my own work in accordance with university regulations.

Abstract

Machine learning algorithms have the potential to significantly enhance decision making processes in highly impactful fields, ranging from health care to criminal justice sentencing. Owing of the significance and intricate nature of the decisions that are often made in these fields, it is important for the experts using these algorithms to have a concrete understanding of how they work, and why they make the predictions they do. Unfortunately, some of the algorithms with the highest degrees of accuracy, offer zero levels of interpretability. Such models are referred to as being 'blackbox' in the literature. Recently there has been much interest in finding mechanisms to interpret these blackbox models, as it is generally believed that there is much value in being able to do so. In this paper, we explore the potential that decision trees have in enhancing blackbox model interpretability. To do so, we implement an evaluate an algorithm called Model Extraction — proposed by researchers at Stanford University — which aims to construct interpretable approximations of blackbox models. We develop heuristics and implement methods to evaluate the potential that Model Extraction has in enhancing blackbox model interpretability. Subsequently, we propose a variant of the algorithm which appears to perform better than the original algorithm, when implemented and evaluated with real world breast cancer data.

Acknowledgements

Thank you to the countless number of people who made my Princeton experience all that it was.

Firstly, and most importantly, thank you to my parents for making all of this possible, and for being my biggest cheerleaders on a daily basis. Thank you to my brother for putting up with my terrible long distance communication, and for consistently being there regardless. Thank you to my advisor, Professor Braverman, for your support, patience and advice as I undertook this project.

Thank you to all of the friends I've made at Princeton, many of whom now feel like family as well. To Soham, for helping me structure my thesis, for being the best unofficial thesis fairy, and for being one of the biggest sources of emotional support through my writing process. To Soumya, for inspiring me to maintain my intellectual curiosity on a daily basis, and for being my sounding board, and a stabilizer. To Lindsay, for never getting tired of making fun of me, and for delivering me hugs, love, and entertainment, as I worked. To Waqa, for making the latter half of my Princeton experience so much of what it was, and for teaching me how to be the best version of myself. To Ava, for showing me how to value other people, and for helping me realize the things I care about most.

To Stefanos and Kobby, for making me love attending your classes and for reminding me how much I enjoy learning. To all of my other room mates, to my squash team, and to any one I've had a meaningful interaction with on this campus, thank you for making the last four years all that they were. Leaving this place will be my most heartfelt goodbye.

Contents

Chapter 1

Introduction

1.1 Overarching Motivation

Machine learning (ML) algorithms have the potential to significantly enhance decision making processes in highly impactful fields, ranging from health care to criminal justice sentencing. In the health care sector, ML algorithms are increasingly being used to help determine whether or not a patient has a certain disease, given a set of conditions and symptoms. Recently, for example, there has been much interest in using ML techniques to help predict whether or not a woman is at risk of breast cancer, since "early [cancer] detection based on phenotype and genotype features can greatly increase the chances for successful treatment" [10]. Likewise on social media, there has been increasing interest in using Facebook and Twitter to predict whether an individual is at risk of depression or other mental illnesses. Effectively using ML in these types of fields can offer great value. In the latter example, for instance, ML algorithms could substantially help early stage diagnoses of mental illnesses. Though many illnesses are curable, unfortunately, a large proportion of affected patients never receive treatment – either because they lack access to care or feel constrained by stigmas associated with mental health. Using social media to identify and reach out

to users at risk of mental illness could substantially increase the number of patients who receive diagnoses and thus adequate treatment. Diagnoses of this sort would be especially valuable in lower income communities, where people face more barriers to seeking care.[6]

Because of the sheer significance and intricate nature of the decisions being made in the situations outlined above, in order to effectively use ML algorithms in these decision making processes, one ought to have a thorough understanding of the algorithm and how it works. Specifically, it is important for experts to be able to validate the predictions made by these models, to ensure that the logic behind the prediction is rational and fair. Unfortunately, however, some of the models that offer the highest degrees of accuracy in the above domains offer nearly zero levels of interpretability. This means that we cannot understand why the algorithm makes the predictions it does. Models of this nature are referred to as being 'blackbox'. For a given input, these models only offer an output and no explanation or reasoning as to how the algorithm obtained that output. Some of the most popular blackbox models include neural networks and random forests.

Failure to understand how blackbox algorithms work and naive reliance on their results can lead to undesirable consequences. A prime example of this is the US Police Department's use of COMPAS. COMPAS is a recidivism prediction algorithm that assigns convicts risk scores based on their background. These risk scores were often used to determine an offender's bail sentence. However, sometime after its implementation, a deeper analysis of COMPAS' predictions revealed that the algorithm was inherently biased when making 'repeat offender' predictions. Though the algorithm had a fair degree of accuracy overall, when analyzing its errors it was revealed that the algorithm consistently predicted black convicts as more likely to be repeat offenders than they actually were, and white convicts to be less likely [11]. Had experts had a more thorough understanding of this algorithm before deploying it, they may

have been able to see the disproportionate importance the algorithm gave to race related features, and thus detect the problem before it influenced several bail sentence decisions.

1.2 Thesis Specific Motivation

Through the literature, there seem to be two primary ways in which researchers are tackling the problem of improving blackbox model interpretability.

1. *By creating interpretable approximations of the blackbox models*[12]

   This approach entails using an interpretable model (examples include support vector machines, decision sets, decision trees etc.) to help us understand the behavior of the blackbox model. The general aim of this approach is to construct an interpretable model that most effectively mimics the behavior and output of the blackbox model. Thus this approach is not concerned with maximizing the accuracy of the interpretable approximation with respect to a raw data set; rather, it is concerned with maximizing the accuracy of the approximation with respect to the blackbox model's output.

2. *By direct inspection and experimentation with the blackbox model itself*

   These approaches generally attempt to interpret singular predictions, rather than attempt to understand the blackbox model's behavior as a whole or on a more global scale. This is usually done by perturbing a test input's features in some way, and measuring how these perturbations affect the blackbox model's output.

In this paper, we focus on the first approach to improving blackbox model intepretability. Specifically, we are interested in evaluating the potential that decision trees have in helping us interpret blackbox models. A decision tree, like the one shown in figure 1.1 below, is an interpretable predictive model that can be used for

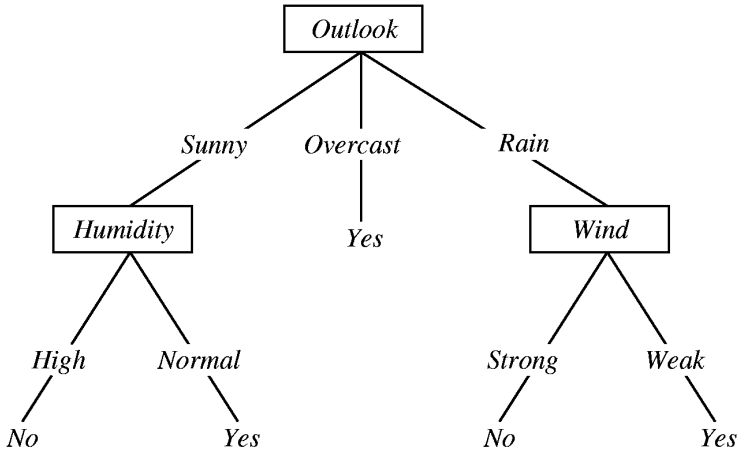


Figure 1.1: A decision tree that determines whether or not to conduct an event, given the weather conditions. The input attributes are the weather characteristics and the output value is a binary prediction corresponding to whether or not to conduct the activity.

classification or regression tasks. To obtain a prediction for a given input, one starts at the root node and follows the edges that correspond to the input's feature values. Eventually we reach a leaf node that contains the predicted value. Standard decision trees are commonly referred to as 'CART' (Classification And Regression Trees) trees in the literature. For the remainder of this paper, we will also refer to these trees as CART trees.

Unlike many other interpretable ML models, which are only good at capturing linear behavior, decision trees can effectively capture non-linear behavior too. Many blackbox models — such neural networks and random forests — are specifically used when the data has a non-linear structure, since these models work especially well in these scenarios. Because of this, we believe that decision trees could be particularly interesting models to explore in the context of aiding blackbox model interpretability.

To explore the potential that decision trees can have in aiding blackbox model interpretability, we first implement and evaluate an algorithm presented by Osbert Bastani, Carolyn Kim, and Hamsa Bastani (from Stanford University) called Model Extraction [5]. We explain the details of this algorithim in Chapter 3. In a nutshell,

Model Extraction aims to help us interpret a blackbox model by creating a decision tree that is supposed to mimic the behavior and logic of that blackbox model. We will often refer to such trees as decision trees that 'approximate' a blackbox model. Beyond simply presenting the algorithm, Bastani et al's paper, entitled *Interpretability via Model Extraction*, advocates two primary ideas:

1. The decision trees constructed by the Model Extraction algorithm perform better than standard CART trees in terms of both a) accuracy on test data and b) accuracy relative to the model they attempt to approximate

2. The decision trees produced by their algorithm can help us interpret blackbox models

Through analysis and experimentation with real world breast cancer data, this paper evaluates both of the ideas presented in *Interpretability via Model Extraction*. We subsequently present a variant of the Model Extraction algorithm, which improves upon the original algorithm. This study is performed to better understand how much potential a) the Model Extraction approach and b) decisions trees in general have in helping us improve blackbox model interpretability.

Following the above analyses, in the extensions section of this paper, we briefly consider the second approach to interpreting blackbox models – which is via direct inspection. We present an approach that we believe may help us understand which features a neural network places the most significance on when determining an output for a given input.

Chapter 2

Related Work

Reserach in this domain seems to be fairly new, with the earliest directly relevant papers dating back to 2016. Likewise, tools to aid blackbox model interpretability — such as Local Interpretable Model-Agnostic Explanations [4] — were also only made public in 2016. Many researchers in this field are primarily interested in finding interpretation methods that are model agnostic. This means that the interpretation mechanism only requires access to the input and output from the blackbox model to work, and does not require any understanding of nor place any constraints on the blackbox model's internal structure. Through the literature, it appears that approaches to interpreting blackbox models can be split into three categories, each of which are described below.

2.1 Local Approximations

Local approximations of blackbox models focus on providing interpretability at a local level. This means that for a single data point, these methods aim to help us understand why the blackbox model gave the data point its respective prediction value.

2.1.1 Local Interpretable Model-Agnostic Explanations (LIME)

LIME[4], a tool developed by Marco Tulio Ribeiro, Sameer Singh and Carlos Guestrin, helps us understand blackbox model predictions at a local level. This tool works particularly well when the input domain consists of text or images. The algorithm's developers believe that it is easier, and arguably more effective, to provide interpretations/approximations of the blackbox model on a local level, where explanations behind the prediction of specific instances can be provided (rather than having to provide interpretations on a global level). Its approach is as follows:

- The instance is split up into well definable and interpretable regions. Thus if our input was text, these regions would consist of specific words or phrases within the input
- By removing some of these regions from the original instance, multiple variants of the instance are created
- These variants are fed into the model, and their output is recorded
- Each variant is given a similarity rank to the original instance
- The above two factors are used to create a simple (often linear) weighted model, which portrays which regions/features are most significant in determining the original instance's prediction

2.1.2 Rivelo

Rivelo[9] is a visual analytics interface developed by Paolo Tamagnini, Josua Krause, Aritra Dasgupata, and Enrico Bertini, that helps the average user to understand the predictions made by blackbox models. Rivelo captures the behavior of blackbox models at a local level, since for each data point, the interface provides a short explanation suggesting why the data point received the label it did. By calculating

the frequency of the features that appear across all of the instance explanations, Rivelo also provides insight into the model's behavior on a global level.

Rivelo creates its instance explanations by first determining which features are most significant in that instance's prediction. Let f be the model we are trying to interpret. Since the classifier is binary, the instance label is determined by comparing $f(\mathbf{x})$ to a given threshold, $theta \in [0, 1]$. The most significant feature, in this algorithm, is deemed to be the one whose removal causes the greatest change in $|f(\mathbf{x})|$, in the direction of the threshold. Significant features are continually removed until the value of $f(\mathbf{x})$ is on the opposite side of the threshold. This set of features, S is the smallest feature set whose removal would cause the binary prediction value to change. Thus the features in S are used to construct the instance level explanation for f.

This approach is model agnostic. However, it is limited because it only works for binary classifiers that have binary input feature sets, and is more efficient when these feature sets are relatively sparse. Since many text classifiers use a bag of words representation though, this approach has several applications in very relevant domains.

2.2 Global Approximations

Unlike local approximations, global approximations do not help us understand which features were most important in prediction of a singular data point, or why a specific data point was given its respective prediction. Rather, these approaches aim to help us understand the blackbox model's behavior and decision making rationale at a higher level.

2.2.1 Blackbox Explanations Through Transparent Approximations

In *Interpretable Explorable Approximations of Blackbox Models*[8], Himabindu Lakkaraju, Ece Kamar, Rich Caruana and Jure Leskovec, Propose the framework: Blackbox Explanations Through Transparent Approximations (BETA). Unlike Rivelo and LIME, BETA does not locally approximate the blackbox model and determine which features are most discriminatory in specific instances; rather, it seeks to create a global approximation of the model. The result is a set of 2-level decision sets, where the first level acts as a *neighborhood descriptor*. The primary purpose of the first level, therefore, is to split the feature space into non-overlapping regions. The second level is a conditional if-then statement, which captures the primary logic/feature attributes that are used to discriminate between classes in the specific neighborhood. An example of the 2- level decision sets produced by this approach is shown below This

Figure 2.1: Visualization of decision tree that globally approximates the blackbox model [8]

approach determines the optimal and most representative list of decision sets (denoted as $\mathcal{R}$) by optimizing an objective function. The objective function encapsulates three primary components:

1. **Fidelity** - This is the accuracy of $\mathcal{R}$ with respect to the original blackbox model. We want to maximize fidelity.

2. **Unambiguity** - This is a cumulative measure of how many neighborhood descriptors each data point satisfies (denoted by overlap). Cover is a measure of whether or not each data point fits at least one neighborhood descriptor. Ideally we would want cover to equal overlap

3. **Interpretability** Often simpler models that do not have very complicated logic paths are more intepretable for users. Thus this metric measures the number of features that are referenced throughout the decision sets (the same feature can be counted more than once), and tries to contain this number, so that the logic remains easy to follow. This metric also measures the total number of neighborhood descriptors within the decision set, and tries to moderate that size

The authors of this approach claim that during user studies, users claimed that this provided them more insight into the blackbox model behavior than LIME did. Further testing is still required however to understand the effectiveness of this approch more. One limitation is that this approach can only be used for classification problems.

2.3 Understanding How Models Are Derived From Training Data

This approach to enhancing interpretability differs significantly from the previous few. Instead of attempting to understand and explain the rationale behind a blackbox model's predictions, this approach focuses on trying to understand how the blackbox model learns from and relies on its training data. Specifically, this approach helps

us gain insight into which training points are most influential in the prediction of a given test point.

2.3.1 Influence Functions

Pang Wei Koh and Percy Liang in *Understanding Black-box Predictions via Influence Functions* [7] suggest using Influence Functions to understand how a model depends on, and is derived from, its training data. Unlike the previous approaches, this approach does not attempt to find the features most responsible for a given test prediction; rather, it attempts to find the training data points that were most responsible for the test prediction. To approximate the effect of a training data point on a test prediction, the following approach is taken:

1. **Upweighting a training point -** Koh and Liang proclaim that understanding the effect of a training data point on a prediction is equivalent to removing the data point from the training set, and observing how it influences the prediction. Repeatedly removing points and re-training the model can be expensive, however, thus Koh and Liang estimate the effect of removing a data point using influence functions. In the general case, the model parameters, $\theta \in \Theta$ are determined by:

$$argmin_{\theta \in \Theta} \frac{1}{n} \sum_{i}^{n} L(z_i, \theta)$$

where L is the model's loss function, and z_i is a training data point. Removing a point, the author's claim, is equivalent to up-weighting it by $\epsilon = \frac{-1}{n}$, giving us:

$$\hat{\theta}_{\epsilon,z} = argmin_{\theta \in \Theta} \frac{1}{n} \sum_{i}^{n} L(z_i, \theta) + \epsilon L(z, \theta)$$

2. **Understanding its influence -** To understand the influence ($\mathcal{I}$) of upweight-

ing z, on z_{test}, we need to compute:

$$\mathcal{I}_{\epsilon,L}(z, z_{test}) = \frac{dL(z_{test}, \hat{\theta}_{\epsilon,z})}{d\epsilon}|_{\epsilon=0}$$

In their paper Koh and Liang explain that this is equivalent to computing the expression below, where H is the hessian of L.

$$\mathcal{I}_{\epsilon,L}(z, z_{test}) = - \bigtriangledown_\theta L(z_{test}, \hat{\theta})^T H_{\hat{\theta}}^{-1} \bigtriangledown_\theta L(z, \hat{\theta})$$

Since computing and inverting H across all training points is computationally inefficient, Koh and Liang use various optimization methods to approximate this calculation.

This approach is extremely promising as it allows us to understand how models use their training data to make predictions. In the case of interpretable models, this approach reaffirms what we know about how these models learn and make their predictions with respect to their training data, and in the blackbox domain it provides us with new insight. For example, in one case study, this approach highlighted that while training points which have the lowest euclidean distance to a test point have the most predictive influence in support vector machines (SVMs). In convolutional neural networks (CNN) this is not the case. In the latter there was much less correlation between euclidean distance and influence on a prediction, since training data from different classes was often helpful in determining the classes of test instances. Their paper further illustrates how this approach can be used to fix mislabelled examples, deal with domain mismatches, and lastly, detect vulnerabilities in the models. Hackers can exploit models by creating adversarial data points (that are, at a cursory glance, visually indistinguishable to the original data points) which cause the algorithm to flip its prediction in certain cases. Though this approach offers promise, it is limited

in that it can only assess the effects of singular training data points on predictions, and cannot assess effects of groups of data points. [Furthermore, this approach would have to be used in conjunction with others to interpret blackbox models, as this mechanism does not give us direct insight into the rationale used by blackbox models to make specific predictions in the way that the aforementioned approaches do.] To do: rephrase -terribly worded

Chapter 3

Model Extraction: Algorithm Overview

In *Interpretability via Model Extraction*, Bastani et al. present Model Extraction: an algorithm that attempts to create a decision tree which mimics a blackbox model's behavior[5]. Note that this algorithm is model agnostic – this means that it can be used to interpret any blackbox model, since all it needs access to is the blackbox model's input and output values, and does not require the model to have any specific internal structure. The author's take this route because they seem to believe that if the constructed decision tree's accuracy relative to the blackbox model is high, then the constructed decision tree should mimic the blackbox model. More formally stated:

> Let $f : X \to Y$ be the blackbox model. Let T be a decision tree that approximates f. If T is a sufficiently good approximation of f (i.e if T has a high relative accuracy to f) then T should mimic the logic and rationale of f when making predictions. The accuracy of T relative to f is computed by: $\frac{1}{|X_{test}|} \sum_{x \in X_{test}} 1_{T(x)=f(x)}$[5].

If we assume that this notion is true, the problem reduces to finding mechanisms to

construct the decision tree, T, such that it has a high accuracy relative to f. To do this, Bastani et al. propose an active learning approach. Before we explain their approach, we provide a brief overview of a fundamental idea in statistical learning theory to contextualize why Bastani et al. might have chosen the approach they did.

The Fundamental Theory of Statistical Learning: PAC Learnability

Probably Approximately Correct (PAC) learning offers us a framework through which we can analyze whether or not a given problem is machine learnable (Leslie Valiant, 1984). In simple words, it helps us answer the question: Given this set of constraints, can we feasibly come up with a machine learning approach to effectively solve this problem?

Let $\mathcal{H}$ be a hypothesis class. A hypothesis class can be any general class or family of functions. The PAC learning theorem tells us that if you have m training samples for a given problem, where m satisfies:

$$m \geq \frac{log|H| + log(1/\delta)}{\epsilon}$$

then with probability δ you can develop an ML algorithm that has an accuracy of $1-\epsilon$, for the task at hand. δ and ϵ can be any values between 0 and 1. We see that the larger our hypothesis class is (the more complicated the class of functions that we are working with), the more samples we need in order to effectively learn an algorithm. We also see that in theory, if m is large enough, we can achieve infinitesimally small values of ϵ with high values of δ.

Considering Model Extraction in Context of PAC Learnability

If we consider this in the context of the Model Extraction algorithm, the hypothesis class we are working with is the class of all decision trees that have a maximum of

n nodes (where n is any number we choose to define), and where the features at the nodes can take on any one of d values (where d is the number of features/dimensions of our original data set). Since our hypothesis class is finite and constrained, we can see that if we have a sufficient number of training data points, then it is, in theory, possible to construct a decision tree, T, that almost exactly mimics f.

We know that f is trained on a specific data set (we will refer to this as the original data set going forth). Let us assume this data set is of size S. To construct T, we could train the decision tree with the input values from the original data set as the tree's input values, and the corresponding labels that the blackbox model assigns to these inputs as the output values. The problem with this approach, however, is that we are constrained by S. If the number of training samples we have is not large enough, we cannot create a T that is a sufficiently good approximation of f. In the Model Extraction algorithm, Bastani et. al fit a distribution over the original data set's input space. We believe that they do this to overcome the training size limitation. By fitting a distribution, one can sample an arbitrarily large number of input values from the distribution, and not be constrained by the size of the original training set.

Model Extraction Algorithm Outline [5]

1. Train a blackbox model on the original data set

2. Fit a distribution to the original data's input space using expectation maximization. The distribution that Bastani et al. use and advocate is a Truncated Axis Aligned Multivariate Gaussian Mixture Model (TAAMGMM). A brief explanation of this distribution is provided below this section. Note that from now on we will refer to the process of constructing a decision tree via sampling from a fitted distribution as 'active learning'.

3. Sample N points from this distribution, and calculate their corresponding labels using the neural network. Let N be a relatively large number.

4. Train a decision tree on the above $N(x, y)$ data points. This decision tree serves as the interpretable approximation of the blackbox model

5. Use the tree to make predictions on test data, or inspect its structure to gain insight on the blackbox model's structure

An axis aligned multivariate Gaussian mixture model is a multivariate Gaussian mixture model, whose different dimensions are independent. A truncated version of this distribution simply constrains each of the dimension values so that they are within the minimum and maximum values that that feature takes on in the original data set. Since a Gaussian distribution can take on any values from negative to positive infinity, truncating the distribution eradicates the small probability of sampling input values from the distribution, whose features may have unreasonably large or small values.

Chapter 4

Methodology

We present the methodology in two sections. The first set of methods is aimed at helping us evaluate our first thesis question:

> *Do the decision trees constructed by the Model Extraction algorithm perform better than standard CART trees in terms of both a) accuracy on test data and b) accuracy relative to the model that they attempt to approximate?*

Similarly, the second set of methods is aimed at helping us evaluate the second question:

> *Can the decision trees produced by the Model Extraction algorithm help us interpret blackbox models?*

4.1 Methods, Section I

4.1.1 Data Used

To evaluate the Model Extraction algorithm, we implement and test the algorithm — as well as other models, and variants of the algorithm — on a real world breast

cancer data set. This data was originally obtained from the Clinical Sciences Center in Wisconsin, Madison [1]. The data contains information on 512 breast masses wherein each breast mass is classified as benign or malignant. Every data point has 30 features, each of which depict some attribute of the breast mass. All of the input feature values are real numbers, and output values are binary. The data set is fairly balanced as it includes information about 357 benign masses and 212 malignant ones. More information about the data and the features can be found using the data set link in the bibliography. While training and testing the models below, we split up the data set so that 80% of the values are used for training purposes and 20% are used for testing.

4.1.2 Basic CART Approach

Before we implement the Model Extraction algorithm, we implement a simple CART decision tree and measure how well it performs on the test data set. This gives us a baseline from which we can evaluate the performance of all the other models/algorithms we deploy. We limit their depth of the decision trees to six, and compare how the performance varies across decision trees with depths between 2 and 6. Since the eventual aim of this exploration is to improve interpretability, we are primarily interested in the performance of trees of depths 2, 3 and 4; although larger trees may have higher accuracies, they are prone to overfitting and their larger depth makes them harder to interpret. All of the models we train and distributions we fit are done using the Scikit-Learn software [2].

4.1.3 Neural Network Approach

We then train various blackbox models with our data. After a preliminary round, we find that neural networks outperform other blackbox models, such as random forests, on this data set. We consequently carry on the rest of our investigation using neural

networks[2].

4.1.4 Model Extraction Approach

We implement the model extraction algorithm that we outlined in chapter 3.

Decision Tree Construction without Sampling

As discussed, the purpose of the Model Extraction algorithm is to create decision trees that approximate blackbox models. Before we use Bastani et al's active learning approach to construct these trees, we will simply construct a tree using the X values from the original data set as training input, and the corresponding labels that the neural networks assigns these values as the training output. The purpose of doing this is to measure the value add that fitting and sampling from a distribution offers.

Decision Tree Construction with Sampling

We then implement the Model Extraction algorithm as outlined in chapter 3. To fit the TAAMGMM distribution to the input space, we use expectation maximization[2][5]. Next, we sample 10,000 data points from it and obtain the neural network assigned labels for each of these data points. We experimented with a variety of sample sizes before deciding on 10,000. This number was chosen as it seemed to maximize decision tree performance, while still allowing for computational efficiency. Sample sizes beyond 10,000 did not notably enhance accuracy, and made the algorithms take much longer to run. After constructing the decision trees with this training sample, we evaluate and test its performance on the original test data set.

4.1.5 Variants of Model Extraction Algorithm

Distribution without Axis Aligned Constraints

We hypothesized that the dimension-independence assumption in the axis aligned

models would be too strong, especially for our data set, since many features of biological cells are closely correlated. Thus we explored distributions that allow dependence between the various dimensions. Relaxing the axis aligned assumption, however, meant that we could no longer truncate the sampled feature values in the way we earlier did. This is because when we truncate the distribution at the minimum and maximum feature values (from the original data set), we fundamentally change the distribution slightly. When our dimensions are independent, we can do this to each dimension without worrying about the consequence it may have on other dimensions; however, when our dimensions have dependencies between them, truncation is not as easy. We hypothesize that this trade off might be worth it, and thus fit a multivariate Gaussian mixture model to the input space, instead of fitting the TAAMGMM espoused in the original Model Extraction algorithm.

Furthermore, through experimentation, we found that having high numbers of components in our Gaussian mixture model did not necessarily improve the distribution fit, thus we hypothesized that a simple multivariate Gaussian might work just as well and have similar levels of accuracy. We also thought that the simplicity of this model, in comparison to the Gaussian mixture models, could be helpful for our purpose as it could help protect against overfitting. Thus we experiment with this distribution and fit it to the input space.

4.1.6 Methods, Section II

In the second part of our analysis, we aim to explore the extent to which the decision trees constructed by the Model Extraction algorithm — and the variants of the algorithm — can enhance our ability to interpret blackbox models. Furthermore, we want to gain insight into which of the methods presented in Section I are most effective in helping us enhance interpretability. Since this is a very open question and there does not exist quantifiable metrics or standards in the literature through which we can

directly evaluate this question, we must construct heuristics/quantifiable metrics to facilitate our analysis. Below we present and explain the heuristics that we developed to evaluate how much the constructed decision trees can enhance blackbox model interpretability.

4.2 Consistency

In order for the decision trees produced by the Model Extraction algorithm to facilitate our understanding of blackbox models, they must be consistent. This means that when the algorithm is run multiple times, it must produce relatively similar decision trees.

If the decision trees produced by the algorithm on each iteration differ considerably, we would be presented with multiple potential logic paths, each of which take us from input to output in their own way. It would be difficult to discriminate between these paths and to determine which ones most accurately mimic the neural network's internal logic in any meaningful way. This is simply because we have a lack of information with which we could make this assessment. One could argue that accuracy is a potential metric that could be used to make this assessment. While we could choose the decision tree that has the highest accuracy relative to the neural network, and proclaim its logical pathway as being the one that most closely mimics the neural network's internal logic, if all of the produced decision trees have very similar accuracy (which they often do), then this metric is insufficient to make any meaningful discrimination. This is especially an issue when we have decisions trees that have notably different logical pathways, but very similar accuracy. Consequently, in order for the Model Extraction algorithm to facilitate our understanding of how blackbox models work, as a first step, it must produce decision trees that are relatively consistent and have relatively similar logical pathways.

How to most effectively measure decision tree similarity is an open question, for which much research is being done. Consequently, to measure how consistent the produced trees are, we develop heuristics and quantifiable metrics with which we can make this analysis. Below we present two possible ways of measuring consistency.

Root Consistency

A naive but simple approach to measuring consistency involves considering the decision tree roots. The feature at the root is often the feature that the tree deems as having the most significance. Therefore, if a large sample of the constructed decision trees have the same root, it indicates that the trees have at least a baseline level of consistency. If the data we are working with is numeric, beyond just looking at the root feature, it is instructive to look at the feature value on which the split occurs - we will refer to this as the root threshold going forward. We measure the root features and root thresholds produced across 100 decision trees and analyze the results.

Defining A Decision Tree Similarity Heuristic

The evident limitation with the above approach is that it only considers the root nodes. To get a more holistic picture of how similar two trees are, we need to consider the nodes further down in the tree as well. Below is a heuristic we developed that attempts to overcome this limitation and offer an alternative, and more quantifiable measure of similarity.

Outline of Tree Similarity Heuristic

1. Let d be the number of features in our data set. For each decision tree, create a d-dimensional vector. We will populate each index, d_i so that it contains a measure of how much importance that decision tree gives to feature i.

2. For each feature, calculate the number of training samples that use that feature

in their classification. To illustrate this, let us consider a given training data point x. P is the path between the root node and the leaf node into which x is classified. If feature i exists in path P, we consider point x to 'use' i in its classification.

3. Performing this process across all training data points tells us how many training samples each feature helps classify. This gives us a measure of how important this feature is to the decision tree, and how much weight the tree places on that feature. Going forth, we will refer to the vectors constructed in this manner as the vectors that 'represent' the respective decision trees that they were constructed from.

To measure the difference between two trees X and Y, represented by vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, we can use the following formula:

$$\sum_{i=1}^{d} |x_i - y_i|$$

The above expression, which is the L1 norm of $\boldsymbol{x} - \boldsymbol{y}$, measures the difference between the importance that tree X and tree Y places on each feature i.
Since we are more interested in finding a measure of similarity between multiple trees (say 100, or so), it would be more instructive to use the following metric:

$$\frac{1}{n} \sum_{j=1}^{n} \sum_{i=1}^{d} |X_{ji} - \hat{x_i}|$$

where n represents the number of decision trees we are comparing; X_j represents a given decision tree; X_{ji} represents the number of samples that feature i is used to classify in tree j; and $\hat{x_i}$ represents the median value of x_i across the n vectors.
Both metrics use the L1 norm instead of the L2 norm, as the former penalizes outliers less harshly, and it is fairly likely that we will see some large and extreme values for

$X_i - \hat{x_i}$.

4.2.1 Applying the Similarity Heuristic to Gauge Feature Importance

An interesting application of the decision tree similarity heuristic is that we can use it to gain insight into which features are deemed as being most important across a large number of our constructed decision trees. This is useful because whilst it is easy to visualize and interpret a singular decision tree, it is hard to interpret collections of decision trees and to understand, on average, which features are deemed as being the most important across a collection of decision trees.

To find the most important features across n decision trees, we construct vector representations of each of the n trees, in the same manner as we did in the section above. Since each of these vectors is d dimensional, aggregating the n vectors into a 2d matrix gives us an n by d matrix. Finding the column-wise mean of this 2d matrix gives us a d dimensional vector, whose values, d_i represent the average number of training samples that feature i is used to classify. The indices corresponding to the k largest value in the array would represent the k most important features across the n decision trees.

4.3 Mistake Analysis

As has been discussed, simply looking at the relative accuracy between the constructed decision trees and neural networks gives us limited insight about the extent to which the decision tree's logic path mimics the that of the neural works'. Further insight can be gained by analyzing the mistakes made by the two models. Specifically it is instructive to see what portion of the mistakes made by the neural network are also made by the constructed decision trees. This lets us see whether or not the

models are making the same errors, or different ones. Intuitively speaking, if the constructed decision tree is making the same errors that the neural network is, it is more likely that the tree is mimicking the neural network's logic. If the tree is making entirely different mistakes, however, then even if the two models have similar accuracy, it is likely that the internal logic and rationale being used by the two models differs.

Chapter 5

Results

5.1 Results, Section I

5.1.1 CART Decision Tree

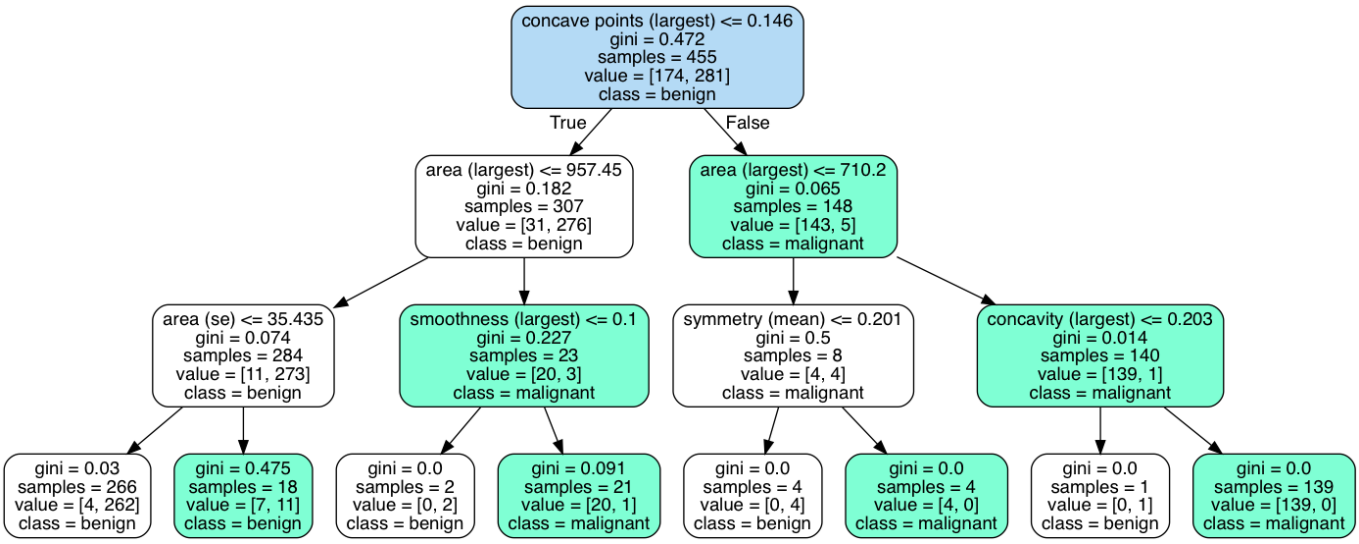
Table 5.1: Accuracies obtained by the decision trees (over 1000 trials) at different depths. Raw accuracy refers to the accuracy obtained on the breast cancer test set, and relative accuracy refers to the decision tree's accuracy relative to the neural networks

Tree Depth	Raw Accuracy	Relative Accuracies
2	0.922	0.925
3	0.926	0.931
4	0.931	0.935
5	0.931	0.934
6	0.930	0.933

As the table shows, increasing the depth of the tree beyond 4 does not have any significant benefit on both raw and relative accuracy. Thus the optimal tree seems to be the tree of depth four, which has a raw accuracy of 0.931, and a relative accuracy of 0.935.

Visualization of CART Decision Tree

Figure 5.1: The decision tree of depth 3 constructed from the original dataset. This tree was created using the Graphviz software.



5.1.2 Neural Network

The average accuracy obtained by the neural network over 1000 trials was **0.972**. The neural network we implemented used a rectified linear unit function as its activation function, which is defined as $f(x) = max(0, x)$. It had 2 hidden layers, each of which had 5 nodes.

As we can see, the neural network's accuracy was approximately 3% greater than the standard decision tree's, making the neural network a desirable model to use in this case. The model extraction algorithm is particularly useful in scenarios like these, where we want to leverage the accuracy of the blackbox model, yet still want to have some insight and understanding of how the model we are using works.

5.1.3 Original and Modified Model Extraction Algorithms

Table 5.2: Raw and relative accuracies obtained by the constructed decision trees over 100 trials. Each of these trees was constructed using either the original model extraction algorithm, or one of its variants. The only difference between the trees is the distribution from which they were constructed. We apply and experiment with six different distributions.

Distribution Model	**Accuracy Relative to NN**	**Raw Accuracy**
depth = 2		
Multivariate Gaussian	0.921	0.914
Axis Aligned Multivariate Gaussian	0.820	0.812
Truncated Axis Aligned Multivariate Gaussian	0.684	0.688
Multivariate Gaussian Mixture	0.930	0.923
Axis Aligned Multivariate Gaussian Mixture	0.892	0.885
Truncated Axis Aligned Multivariate Gaussian Mixture	0.896	0.895
No distribution	0.926	0.924
depth = 3		
Multivariate Gaussian	0.940	0.936

(Continued on next page)

Table 5.2: (continued)

Distribution Model	**Accuracy Relative to NN**	**Raw Accuracy**
Axis Aligned Multivariate Gaussian	0.875	0.873
Truncated Axis Aligned Multivariate Gaussian	0.735	0.733
Multivariate Gaussian Mixture	0.939	0.933
Axis Aligned Multivariate Gaussian Mixture	0.900	0.898
Truncated Axis Aligned Multivariate Gaussian Mixture	0.907	0.906
No distribution	0.939	0.935
depth = 4		
Multivariate Gaussian	0.947	0.940
Axis Aligned Multivariate Gaussian	0.889	0.884
Truncated Axis Aligned Multivariate Gaussian	0.783	0.781
Multivariate Gaussian Mixture	0.949	0.942
Axis Aligned Multivariate Gaussian Mixture	0.907	0.906
Truncated Axis Aligned Multivariate Gaussian Mixture	0.909	0.907
No distribution	0.940	0.935
depth = 5		

(Continued on next page)

Table 5.2: (continued)

Distribution Model	**Accuracy Relative to NN**	**Raw Accuracy**
Multivariate Gaussian	0.954	0.948
Axis Aligned Multivariate Gaussian	0.908	0.902
Truncated Axis Aligned Multivariate Gaussian	0.811	0.808
Multivariate Gaussian Mixture	0.955	0.948
Axis Aligned Multivariate Gaussian Mixture	0.909	0.907
Truncated Axis Aligned Multivariate Gaussian Mixture	0.914	0.912
No distribution	0.940	0.934

As we can see from the table above, for each of the depths from two to five, the decision tree built using sampled data from the TAAMGMM distribution actually performed worse on a test data set than the simple CART decision tree that was trained on the original data set. The decision tree constructed from the TAAMGMM distribution also has a lower relative accuracy as compared to the neural network than the original CART tree does. Therefore, we have little reason to believe that the TAAMGMM constructed tree approximates the neural network better than a standard CART tree does. For example, at depth three, the CART tree has a raw and relative accuracy of 0.926 and 0.931 respectively; whereas the TAAMGMM constructed decision trees has accuracies of 0.906 and 0.907 respectively.

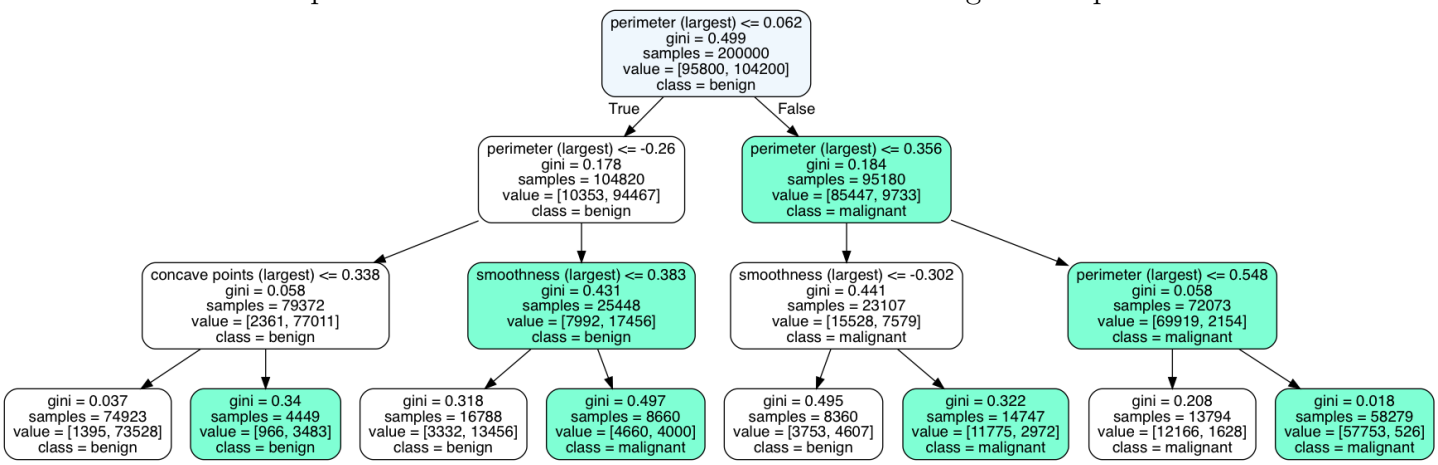
Interestingly, performing truncation revealed that approximately 85% of sampled data points in the non-truncated models had at least one dimension value that was

out of range (where the range for d_i is defined as being $\max_{i \in S}\{d_i\} - \min_{j \in S}\{d_i\}$ and i & j are any two data points within the original data set S). Considering how many sampled data points were out of range, one would think that truncation would help the distribution to mimic the original data set considerably more closely, and thus improve results, however truncation does not seem to be offering much value in this scenario.

Though the TAAMGMM performed relatively poorly, the decision tree constructed from the multivariate Gaussian performed considerably well. It significantly outperformed by the TAAMGMM across all depths, suggesting that the dimensional independence assumption was too strong, and invalid for this data set. The multivariate gausssian performed slightly better than the original decision tree at depths three, four and five. At depth four, while the CART decision tree had raw and relative accuracies of 0.931 and 0.935 respectively; the multivariate Gaussian constructed tree had accuracies of 0.934 and 0.94 respectively. However, since the tree of depth five is very prone to overfitting and less interpretable than the smaller depth trees, we will not focus on trees with depths greater than four for the remainder of the analysis.

Visualization of Decision Tree Constructed from Multivariate Gaussian Distribution

Figure 5.2: Visualization 2: Decision Tree of depth 3 constructed with X values from Multivariate Gaussian Distribution and Y values from the corresponding Neural Network output. This visualization was also constructed using the Graphviz software.



Although the decision trees in figures 5.1 and 5.2 offer similar levels of accuracy on a test data set (the former is 0.931 and the latter is 0.936) we can see that both models have notably different reasoning behind their predictions. For example, we see that model 2 places a considerable amount of importance on the breast mass' largest perimeter value, since it appears in both the first and second levels of the tree. The first visualization, however, does not consider this feature even once. This disparity further motivates section II of our methods and results section, where we construct heuristics to help us understand which interpretable approximations are more effective/reliable blackbox model approximations.

5.2 Results, Section II

5.2.1 Consistency Analysis

Root Feature Analysis

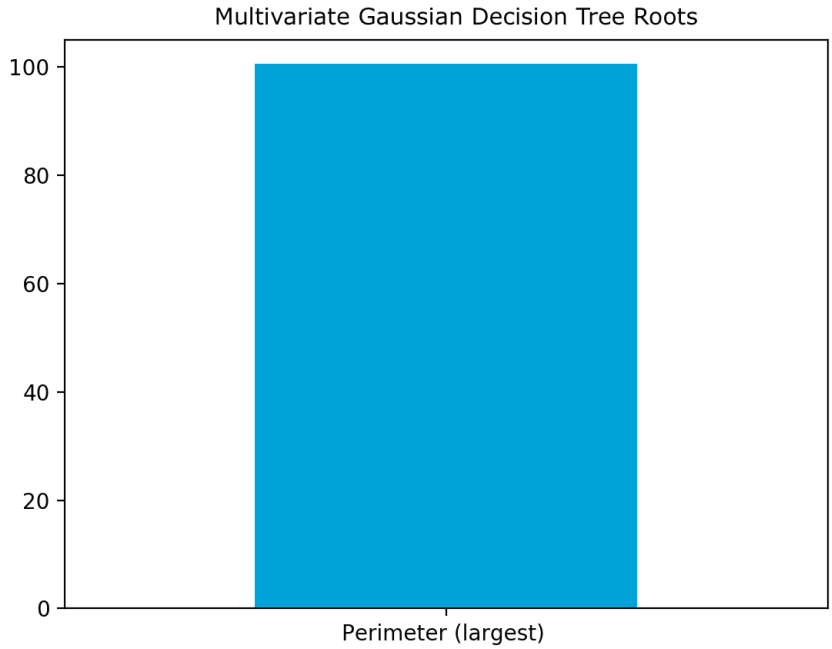


Figure 5.3: A histogram showing the roots of 100 decision trees, each of of which were trained using 10,000 sampled data points from the multivariate Gaussian distribution as input, and the corresponding neural network assigned labels as output

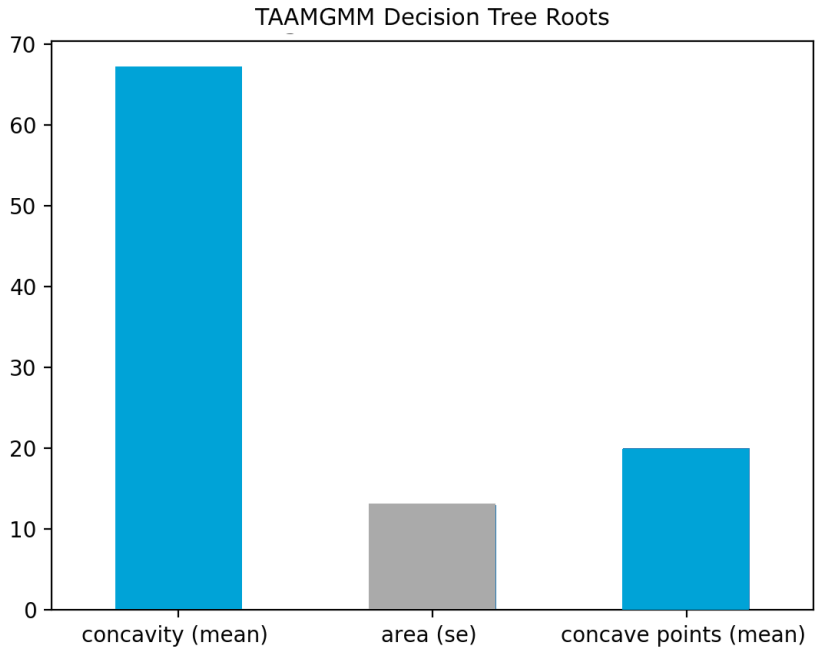


Figure 5.4: A histogram showing the roots of 100 decision trees, each of of which were trained using 10,000 sampled data points from the TAAMGMM distribution as input, and the corresponding neural network assigned labels as output

As we can see, the decision trees constructed from the multivariate Gaussian distribution had considerably more consistent roots than the decision trees constructed from the TAAMGMM distribution. The root feature is the one that the model attributes most importance to; thus seeing that every one of the 100 decision trees constructed from the multivariate Gaussian distribution has the same root offers us assurance that all of the trees had at least a baseline level of similarity, as they all agreed upon what the most important/discriminatory feature was. On the other hand, the decisions trees constructed by the TAAMGMM distribution had three different roots, with the most prominent root appearing only just under 70% of the time. Interestingly, none of these three roots include the feature 'Perimeter (Largest)', which was the sole root in the former case. It is important to keep in mind however, that this approach offers us limited insight into decision tree similarity. This is because two trees that espouse the exact same logic could have different roots. Trees A and B below offer an example of this, where both trees represent the exact same logical pathways, but have

completely different structures. Furthermore, in this analysis, we only consider the root feature. To gain a more wholesome measure of decision tree similarity we need to consider the nodes deeper down in the tree as well.

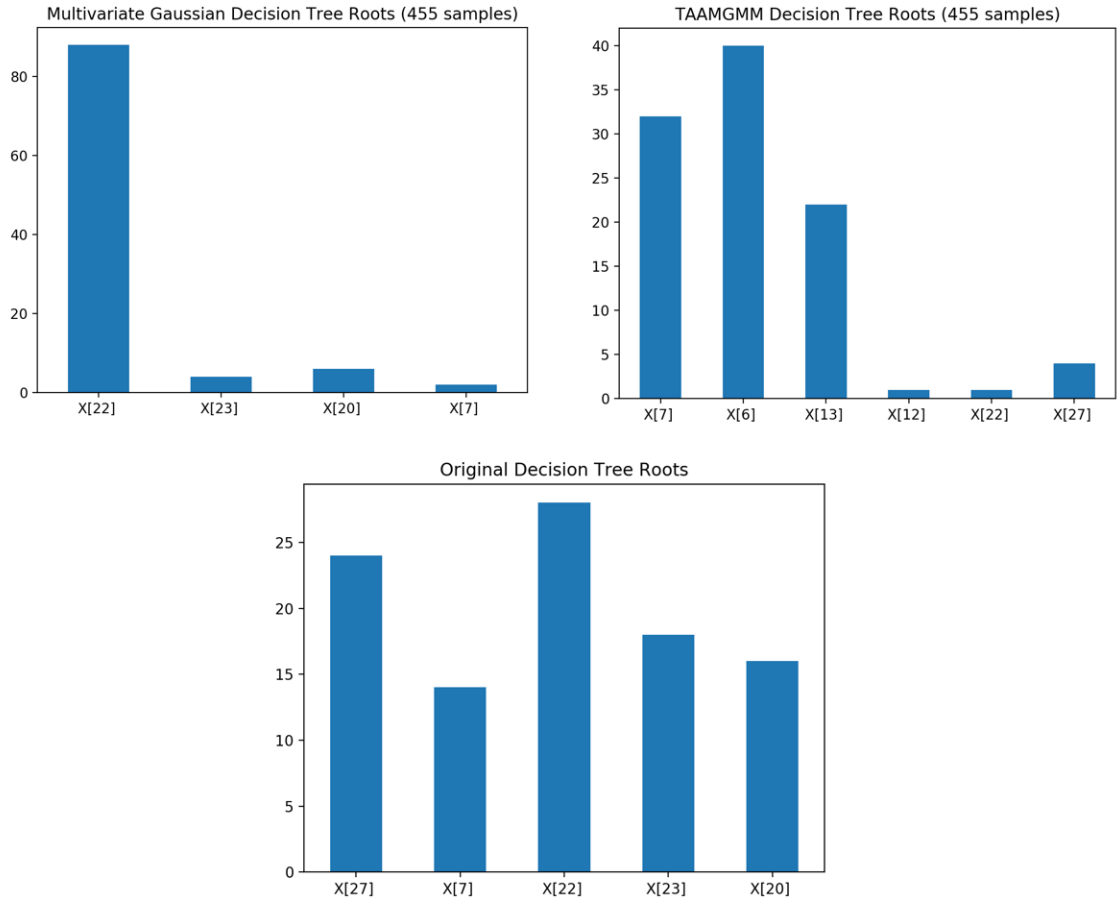


Figure 5.5: Histograms showing the roots of 100 decision trees, where the top left trees were trained on input from a multivariate Gaussian distribution, the top right trees were trained on input from a TAAMGMM distribution, and the bottom center trees were trained on the original breast cancer data set. Each of the trees were trained on 455 samples.

By comparing figure 6.3 with figures 6.2 and 6.1, we can see that training our decision trees with a higher number of samples generally improves the root consistency. This gives the active learning approach a comparative advantage (over just using ordinary decision trees) in scenarios where we have a relatively small sample size, but want to use decision trees to gain logical insights/enhance interpretability.

Root Threshold Analysis

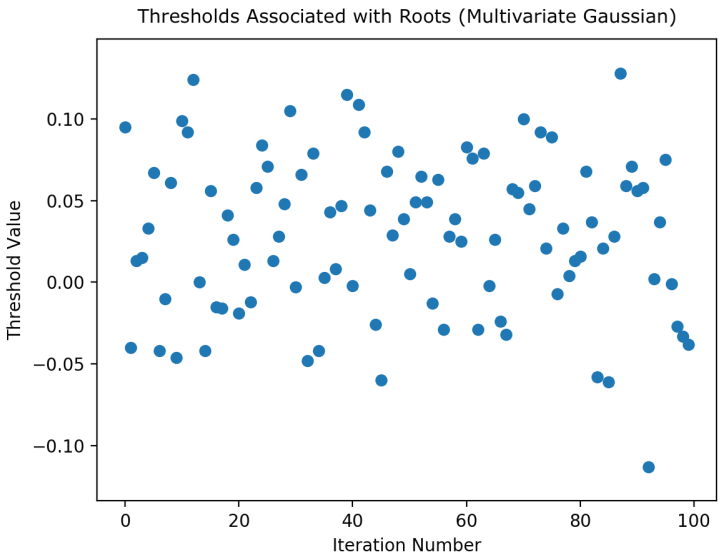


Figure 5.6: A scatter plot showing the root threshold values for 100 decision trees, that were constructed using input values from the multivariate Gaussian distribution

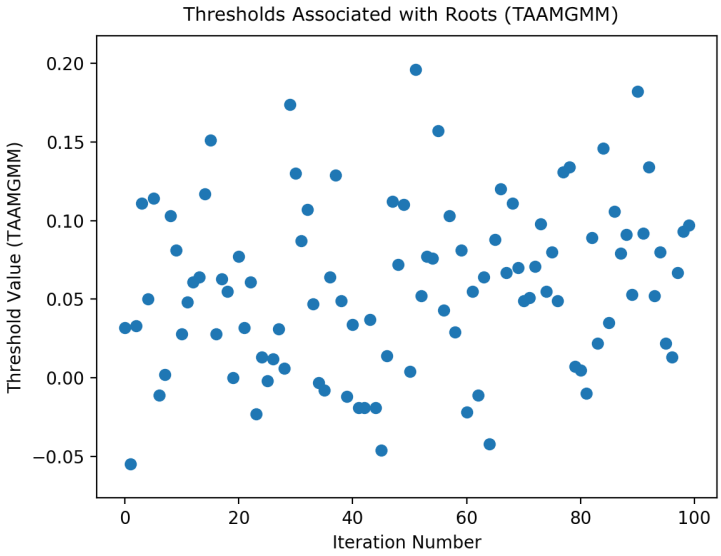


Figure 5.7: A scatter plot showing the root threshold values for 100 decision trees, that were constructed using input values from the TAAMGMM distribution

Figure 6.4 highlights that not only were the root features consistent in the trees that were constructed from the multivariate Gaussian distribution, but the root thresholds were relatively consistent too. Since the data is normalized, all features have a stan-

dard deviation of 1. We can see that the majority of root threshold values lie within -0.05, and 0.10, which is just one-tenth of a standard deviation away from the mean. Considering that this result is obtained across 100 decision trees, and that each of the decision trees are constructed on different input data that is randomly sampled from a distribution, this consistency is fairly notable. In Figure 6.4, we can see that the threshold values lie between -0.05 and 0.2 - so within two-tenths of a standard deviation of the mean. Because however, the root features are less consistent among the trees constructed from the TAAMGMM distribution, the threshold consistency analysis is less instructive, because the thresholds do not all correspond to the same feature.

5.2.2 Decision Tree Similarity Heuristic

Table 5.3: A similarity measure of the decision trees constructed from the TAAMGMM distribution, and multivariate Gaussian distribution respectively. The first column shows the value computed across 100 decision trees, and the second column shows the value computed across 200 decision trees. Each tree was constructed using 10,000 samples from each distribution, and their corresponding neural network labels

Distribution	**Similarity Measure (100)**	**Mistake Similarity**
TAAMGMM	9658	0.51
Multivariate Gaussian (MG)	7650	0.58

Before we discuss the results in figure 5.3, we will present the upper and lower bound of $\frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{d}|X_{ji}-\hat{x_i}|$ to contextualize the results. The lower bound of this expression is 0; this occurs when the vectors $x_j\ \forall\ j$ are equal. The upper bound of this equation is $s \cdot h$, where s is the number of training samples used to construct each decision tree (which we control and keep constant across all decision trees) and h is the depth of the decision tree. The bound is $s \cdot h$ because at each tree level, the

sum of the samples contained within the nodes at that level can be at most s. If a particular feature appears twice in a certain root $\rightarrow$ leaf node path, we do not double count samples associated with that feature. Therefore $\sum_{i=1}^{d} x_{ji}$ is maximized for a given x_j if each root $\rightarrow$ leaf path in the tree contains distinct nodes (note that the nodes within a path should be distinct, but two different paths can have overlapping nodes). Since the value of $\hat{x_i}$ is always positive, we know that:

$$\sum_{i=1}^{d} |X_{ji} - \hat{x_i}| \leq \sum_{i=1}^{d} X_{ji} \leq s \cdot h$$

$$\frac{1}{n} \sum_{j=1}^{n} \sum_{i=1}^{d} s \cdot h| \leq s \cdot h$$

We can see that on average, the similarity between the trees produced by the mutlivariate Gaussian distribution is higher than the trees produced by the TAAMGMM distribution. This reinforces the idea that fitting a multivariate Gaussian distribution to the input space can be more effective than fitting a TAAMGMM distribution - especially with data sets like the one we are working with, where the different dimensions of the original data set are correlated. A limitation of this heuristic however, is that the similarity values can be fairly volatile. This is because the training samples from which you construct each tree can vary considerably. The volatility was more pronounced when the trees were constructed from a smaller number of training samples, thus we stuck to 10,000.

5.2.3 Applying the Similarity Heuristic to Gauge Feature Importance

Using the method outlined in 4.2.1, we get the three most important features across 100 decision trees constructed using the TAAMGMM distribution, and 100 decision trees constructed using the multivariate Gaussian distribution. The three most im-

portant features in classifying an input in the former case were:

1) Breast mass concavity (mean value)

2) Breast mass area (standard error value)

3) Breast mass image concave points (mean value)

The three most important features in the latter case were:

1) Breast mass concavity (standard error)

2) Breast mass image concave points (mean value)

3) Breast mass perimeter (largest value)

It is interesting that the different tree constructions only seem to have one overlapping feature in their '3 most important' feature sets.

5.2.4 Mistake Analysis

Table 5.4: The term 'Mistake Similarity' is a fraction that represents what proportion of the mistakes made by the neural network by were also made by the various models that attempted to approximate the neural network. The presented fraction is computed over an average of 100 trials.

Distribution	**Mistake Similarity**
TAAMGMM	0.51
Multivariate Gaussian (MG)	0.58
Neural Network (MG samples)	0.92

As the table above depicts, over 100 trials, approximately half of the mistakes made by the neural network were also made by the decision tree (which was constructed by the TAAMGMM distribution). The MG constructed tree had a slightly higher mistake similarity value, however this difference is not too significant.

As controls, we also trained neural networks on the same data that we used to

construct the decision trees. For the first neural network, we used the data sampled from the multivariate Gaussian as input, and the corresponding neural network labels as output. The same was done for the second neural network, only this had input data from the TAAMGMM distribution. These controls show that not all of the variation in the mistakes by the neural networks and decision trees comes from the structure of the models themselves; rather, some variation simply comes for the fact that we are using different data to train the decision trees and neural networks respectively.

Whilst the mistake similarity values might initially seem low, it is important to note that our testing data set is relatively small (only contains 114 samples), and the accuracy of our neural network is fairly high, so on average the neural network was only making approximately three mistakes per iteration. Thus on average, the NN would only make one or two mistakes that the constructed decision tree would not. Considering this, in conjunction with that fact that the MG and TAAMGMM trained neural networks did not make exactly the same mistakes either, makes the low mistake similarity values seem like less of a red flag. Given the circumstances, the mistake analysis results do not shed much light on the question of whether or not the [actively] constructed decision trees mimic the logic of the neural networks that they are supposed to approximately. In the future it would be valuable to perform this mistake analysis on a larger testing data set, so that more insightful conclusions can be drawn.

Table 5.5: Accuracies obtained by the decision trees (over 1000 trials) at different depths. Raw accuracy refers to the accuracy obtained on the breast cancer test set, and relative accuracy refers to the decision tree's accuracy relative to the neural networks

Model	**Raw Accuracy**	**Relative Accuracy to NN**
CART Tree	0.92	0.93
Neural Network	0.97	1
TAAMGMM Tree	0.90	0.91
Multivariate Gaussian Tree	0.94	0.94

Chapter 6

Discussion

6.1 Does the decision tree produced by the model extraction algorithm outperform a CART decision tree in terms of raw and relative accuracy?

For the breast cancer data set that we worked with, we found that this was not the case. In fact, the model extraction algorithm espoused by Bastani et al. using a TAAMGMM distribution performed notably worse than a CART decision tree did, both in terms of accuracy on the test set and accuracy relative to the blackbox model. A more extensive response to this question is already discussed in section 5.1.3.

6.2 Further Work, Section I

Using different data sets

To build upon this work it would be extremely valuable to repeat this process on other data sets, and compare how the various trees (constructed from the different

distributions/variants of the model extraction algorithm) perform. Performing this analysis would give us a much richer understanding of the merits and limitations associated with the model extraction algorithm. It would also give us a more holistic understanding of which distributions are most effective to use and apply when working with different input spaces.

Applying different distributions

Since all of the distributions we used were variants of a Gaussian distribution, they are all continuous distributions. Thus these distributions can only be used to approximate continuous input data spaces, that contain only real numbers. Fitting any Gaussian distribution on a binary input space, for example, would not make sense, as the values we sample from this distribution could be anywhere $\in (-\infty, \infty)$. No values apart from 1 or 0 however, carry any real meaning or weight in a binary distribution. To confirm this, we tried using Bastani et al's algorithm, and our own variants of it, on discretized text input. As expected, it performed fairly poorly. In order to extend this approach, and make it applicable to binary or categorical input spaces, we need experiment with and fit discrete distributions to our input space. For example, in the case of binary input data, it could be very effective to fit a multivariate bernoulli distribution [3] to the input space, and see how well the decision tree constructed using input values from this distribution performs.

6.3 To what extent do the constructed decision trees enhance blackbox model interpretability?

The decision trees constructed by the original model extraction algorithm, and its variants, effectively enhance our ability to interpret blackbox models if:

1. The constructed decision trees closely mimic the behavior and logic deployed by the blackbox model that the tree attempts to approximate

2. The decision trees produced by the model extraction algorithm are consistent across multiple iterations of the algorithm

If the above conditions hold, then the constructed decision trees could be very valuable in helping us understand and interpret blackbox models. This is because to understand how the blackbox model works, and why it makes the predictions that it does, we simply need to understand how the decision tree works, and why it makes the predictions it does. This can be fairly easily done by examining the structure and the logic pathways within the constructed decision tree. If the trees produced by the model extraction approach are fairly consistent, then we can examine any tree produced by the algorithm to bolster our understanding of the blackbox model (if however, the trees were not consistent, it would be much harder to know which of the trees produced by the model extraction approach we could or couldn't use to help us interpret the blackbox model). Evaluating the extent to which the constructed decision trees can enhance our ability to interpret blackbox models then, comes down to evaluating the two statement listed above. The metrics with which we can evaluate the first claim are relative accuracy and mistake similarity. In our analysis, we found that the best constructed tree — which was produced by fitting a multivariate Gaussian to the input space — had a relative accuracy to the neural network of 0.94. This

relative accuracy is fairly high, so we could interpret it to suggest that the constructed decision tree approximates the blackbox model fairly closely. However, it is important to note that the accuracies achieved by various other models on the original data set were quite high as well (for example, an ordinary CART tree could get an accuracy of 0.92), which makes the relative accuracy of 0.94 seems less impressive.

The mistake analysis showed that on average just over half of the mistakes made by the neural network were also made by the multivariate Gaussian constructed decision tree. Since however, the number of mistakes made by the neural network was only three on average, we cannot infer or decipher too much from this statistic since the size of the neural network mistake set it so small. To more effectively perform mistake analysis, and understand whether or not our interpretable approximations are making the same mistakes as the blackbox models, ideally the number of mistakes made by the blackbox models would be slightly higher.

The root feature, root threshold and decision tree similarity heuristics can are helpful in evaluating. Given that the roots of the decision trees constructed from the multivariate Gaussian distribution were consistent across 100 trials; that the threshold values were within one-tenth of a standard deviation from each other; and that the decision tree similarity value was 7650 (on a scale from 0 to 30,000), it appears that the decision trees constructed from the multivariate Gaussian distributions had a fair amount of consistency. The trees constructed by the TAAMGMM distribution however did not have as high levels of consistency. This makes the trees constructed using the multivariate Gaussian distribution more valuable in helping us interpret the respective blackbox models.

Given the relative accuracy and consistency of the trees constructed using the multivariate Gaussian distribution, there seems like there could be merit in using the model extraction algorithm (with a multivariate Gaussian fitted to the input space) to help us understand and interpret blackbox models. A key way in which we can

use our variant of the model extraction algorithm and our decision tree similarity heuristic to help us interpret blackbox models is by finding which features seem most important across large numbers of decision trees, as we do in section 6.3. Because this heuristic considers a sizable number of decision trees (all of which are intended to be approximations of the neural network), we can say that the significant features highlighted by this approach are more reliable than any measure of feature significance that only considers a single constructed decision tree.

While all of the above results suggest either that there is merit in using our variant of the model extraction approach to aid interpretability or that more research needs to be done to assess its potential, there exists a compelling argument to suggest that this approach may not be effective in helping us enhance blackbox model interpretability. This argument is highlighted in paper entitled *Understanding Black-box Predictions via Influence Functions* by Pang Wei Koh and Percy Liang[7]. We present their argument below.

Influence functions (which are more extensively explained in Related Work, Section 2.3) can help us understand which training data points are most responsible for a given test point's prediction. In a more general sense, subsequently, influence functions can help us understand how models learn from their training data, and how they rely on/use their training data points when making predictions. These functions, therefore have great potential in helping us understand how the internal logic of one model, varies from the internal logic of another. When using influence functions to understand how support vector machines (SVMs) used their training data points in making predictions, in comparison to how neural networks used their training points to make predictions, Koh and Liang found interesting results. It appeared that in SVMs, the training data points with smallest euclidean distance to the test point were most influential in the data point's prediction; however, in neural networks, there was much less correlation between euclidean distance and influence on prediction. In fact,

with neural networks, it appears that some of the training points that were most influential in determining a prediction, came from classes entirely different to the that of the test points. The graphic below, obtained from Koh and Liang's paper itself, encapsulates this notion.

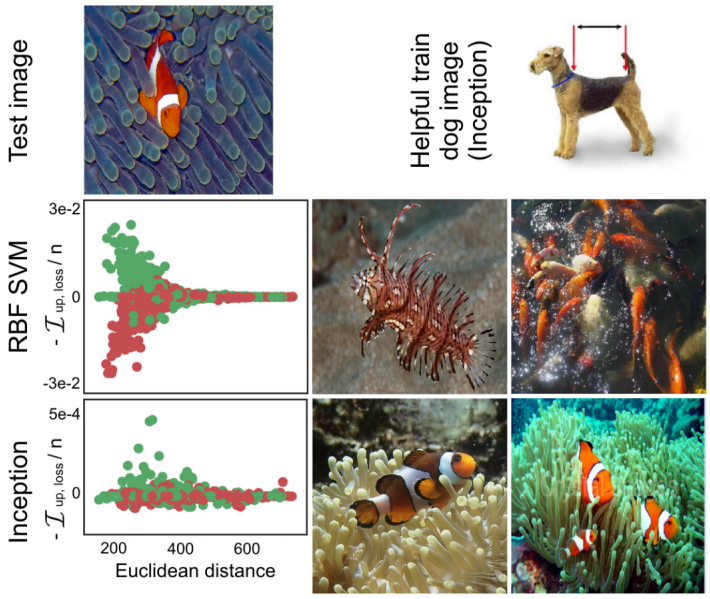


Figure 6.1: In the two graphs (bottom left) the green dots are fish, and the red dots are dogs. The bottom right pictures shows the two training points (for each model) that were most helpful in determining the test point's prediction. This figure is obtained from *Understanding Blackbox Models via Influence Functions* [7]

We can see that in SVMs, the fish with the lowest euclidean distance from the test point had the most (positive) impact on the influence function value. 'Inception' is a state of the art neural network for image classification. As we can see, the relationship between influence and euclidean distance is much less pronounced here. The figure also highlights that in determining this prediction, the dog image did have considerable influence in the neural network, whereas it had very little in the SVM. These findings highlight the idea that although different models might reach similar conclusions, and have similar levels of accuracies, their internal structure and the decision making rationale behind the models' predictions might vary substantially. Thus, even if we construct a decision tree whose accuracy and performance relative

to the neural network it attempts to approximate is very high, we still do not have enough evidence or any sufficient assurance that that the logic used by the decision tree parallels the logic deployed by the neural network. Consequently, it is hard to say whether or not the constructed decision trees can offer us significant insight into how to the neural network works, as we do not know how the neural network uses its training data, in comparison to how the decision trees use their training data.

6.4 Further Work, Section II

As in the previous section, to better evaluate the extent to which the constructed decision trees can enhance blackbox model interpretability, it would be valuable to try the original model extraction algorithm and the variants we proposed on different data sets. Specifically, it would be helpful to work with data sets where the number of mistakes made by the blackbox model is slightly larger, so that we can conduct a more effective mistake analysis. Most significantly, however, it could be very valuable to use the influence functions propagated by Koh and Liang to compare how decision trees and neural networks use their training data. This approach could generally also help us determine which machine learning models have the most similarities with respect to their internal structure and decision making rationale. This knowledge could allow us to more effectively guide our efforts in creating interpretable approximations of blackbox models

Chapter 7

Extensions

As an extension to the project, we began to explore mechanisms to enhance blackbox model interpretability via direct inspection and examination of the blackbox model. These approaches, unlike the previous ones we've mentioned and have worked with, are usually not model agnostic.

The literature shows that one way in which researchers have attempted to enhance blackbox model interpretability via direct inspection, is by creating tools/algorithms that allow you to gain insight into which features are most significant in determining a prediction at a local level. So for example, given a singular test data point i, these tools will tell you which features were most significant in determining data point i's prediction. Two tools perform this particularly well. The first is LIME; a tool created by Marco Tulio Ribeiro, Sameer Singh and Carlos Guestrin in 2016[4] (a more detailed explanation of LIME can be found in Related Work, section 2.1). LIME works particularly well with text and image input, as it highlights the regions of an image or text string that are most significant in determining the input's prediction. LIME however, doesn't work well with other types of input.

Rivelo [9], the second tool, works effectively when the input and output are both binary. Essentially, for each data point, this algorithm determines the smallest subset

of features whose values need to be "flipped" in order to flip the prediction value. For each data point, this subset is deemed to include the features that are most significant for the data points prediction. Though both of these analyses are performed at a local level, aggregating results can potentially give us insight into how the model acts on a global level.

Since there does not exist a tool/algorithm that measures local feature significance when the input values are real numbers; we explore mechanisms to perform/achieve this. To contain the problem, however, we limit ourselves to the real valued input - binary output scenario. Inspired by Rivelo[9], we devise an algorithm that has potential to measure feature importance at a local level in the real value - binary scenario.

Algorithm Outline

1. Let A be the set of features that are most important in determining the output for a given input. Initially, A will be an empty set

2. For a given test point,x, calculate the significance, $s(x_i)$ of each feature, x_i. $s(x_i)$ can be calculated using any one of the two metrics below. Further experimentation and analysis needs to be done to discriminate between which metric is more effective

$$s_1(x_i) = |\frac{\delta f(x)}{\delta x_i} \cdot \frac{x_i}{f(x)}|$$

$$s_2(x_i) = |\frac{\delta(logit(f(x))}{\delta x_i}|$$

$$s_2(x_i) = |\frac{\delta log(f(x)/1 - f(x))}{\delta x_i}|$$

3. Take the feature with the largest $s(x_i)$ value (we consider it to be the most significant feature). Add this feature to A and set its value to the mean value of that feature.

4. Iteratively perform this process until the value of the prediction flips

5. The features that are in set A upon the completion of this process are the features that are considered most significant in determining the predicted value for the given data point

6. To get a gauge of which features are most strongly correlated with each other, we could also note which features $s(x_i)$ value changes the most after setting a certain feature's value to the mean. An increase a feature $j's$ $s(x_j)$ value after setting feature i's value to the mean suggests that feature i and j are correlated; whereas a decrease in feature $j's$ $s(x_j)$ value after setting feature i's value to the mean suggests that there was an interaction effect between the features.

Algorithm Explanation

The first feature significance metric, $s_1(x_i)$, measures the percentage change in $f(x)$ divided by the percentage change in x_i. If we re-write the expression in the following way, it is easier to see this:

$$s_1(x_i) = |\frac{\delta f(x)}{f(x)} / \frac{\delta x_i}{x_i}|$$

We measure the percentage change in $f(x)$ and the percentage change in x_i because intuitively speaking, a change in $f(x)$ or x_i from 0.01 to 0.02 is more significant than a change from 0.45 to 0.46. Considering raw values does not however capture this, but considering percentages does. In $s_2(x)$, we build upon our first metric by adding the logit function into our measure of significance. The logit function, defined as $logit(x) = log(\frac{x}{1-x})$ has very high values for x values near 0 and 1, but lower values for x values in between that range. Therefore, this measure not only treats changes in $f(x)$ from 0.01 to 0.02 as being more significant than changes from 0.45 to 0.46; but also it treats changes from 0.96 to 0.97 as being more significant as well (the first metric however only treats changes near zero as being more significant). In our case, when we have binary output values, we want both changes near 0 and near 1

to be considered more heavily than changes in other parts of the interval. This is because a probability increase of x being 1, from 0.98 to 0.99, results in a probability decrease of x being 0 from 0.02 to 0.01. Since all probability changes near one have complimentary probability changes near zero, we want to consider both changes near and near 1 as having more significance when constructing our feature significance metric. Note, although there does not exist much literature about how to compute feature importance at a local level, there does exist literature about how to perform neural network feature selection on a global level. We used insights from global feature selection methods to guide our approach. Specifically, in *Feature Selection with Neural Networks* [13], we see that first derivatives are commonly used in computing feature importance.

After computing the most significant feature, we set its value to the mean (and continue to do this process until the prediction value flips). This, however, means that if a given data point has all of its feature values near the mean, then we will not be able to 'flip' its prediction using this approach. Thus this approach can only be used to understand which features are most significant for the prediction of data points which have feature values far away from the mean. While this might initially seem like a limitation of this mechanism; often the data points which belong to 'out-of-the-norm' classes have feature values that stray away from the average feature value. Understanding which features results in the 'out-of-the-norm' prediction is often more interesting and desirable than understanding what features result in prediction of the norm. Thus this approach still has potential and value to it.

Though we managed to devise the algorithm, owing to time constraints, we did not have time to fully implement and evaluate it. As further work, we would implement this algorithm and test it with a neural network built on TensorFlow.

Chapter 8

Conclusion

In this research, we implement the Model Extraction algorithm proposed by Bastani et al. and develop heuristics to evaluate the algorithm. Through our analysis, we aim to evaluate two primary ideas that are advocated in Bastani et al.'s paper

1. The decision trees constructed by the Model Extraction algorithm perform better than standard CART trees in terms of both a) accuracy on test data and b) accuracy relative to the model they attempt to approximate

2. The decision trees produced by their algorithm can help us interpret blackbox models

By implementing and testing Bastani et al.'s Model Extraction algorithm on a breast cancer related data set, we find that the constructed decision trees produced by the original Model Extraction algorithm actually perform worse than a standard CART tree in terms of both: accuracy on the original test data, and accuracy relative to the blackbox model. For example, for a decision tree of depth three, the CART tree's accuracy on the test data set is 0.926, and its accuracy relative to the neural network is 0.931. The decision tree produced by Bastani et al.'s Model Extraction algorithm however has an accuracy of 0.906 on the test data, and an accuracy of 0.907 relative to the neural network.

While the original algorithm performs worse, a variant of the algorithm in which we fit a multivariate Gaussian distribution to the input space — instead of a TAAMGMM distribution — performs much better than the original algorithm, and performs marginally better than the baseline CART tree at tree depths of four and five. At depth four, while the CART decision tree had raw and relative accuracies of 0.931 and 0.935 respectively; the multivariate Gaussian constructed decision tree had a raw and relative accuracy of 0.934 and 0.94 respectively.

The decision trees produced by our variant of the Model Extraction algorithm also appeared to be more consistent than the trees produced by the original Model Extraction algorithm with respect to all: root feature consistency, root threshold consistency, and the decision tree similarity heuristic we developed. While the original and our variant of the Model Extraction algorithm definitely needs to be further tested before making any claims, it appears like our proposed variant might have more promise than the original Model Extraction algorithm in helping us construct high performing decision trees and interpreting blackbox models.

Although we concluded that there does seem to be merit in using these constructed decision trees to enhance blackbox model interpretability, we highlighted that a major obstacle to evaluating this question more holistically and thoroughly is that we do not know whether decision trees learn from and 'use' their training data points in similar ways to how neural networks learn from and use their training data points. If it is the case that decision trees learn from and use their training data points in ways entirely different to neural networks, then the Model Extraction approach (both the original one and our proposed variant) becomes less useful in helping us enhance blackbox model interpretability. Therefore, to better evaluate the potential that the decision trees produced by our variant of the Model Extraction algorithm have in enhancing blackbox model interpretability, it could be very valuable to better understand how decision trees learn from and rely on their training data points, in comparison to how

other blackbox models do so.

As mentioned in the introduction, creating and improving mechanisms to enhance blackbox model interpretability has much value; especially today, when blackbox models are increasingly being used to influence significant decisions in fields ranging from health care to criminal justice. The ability to interpret and understand blackbox model behavior would allow these models to be so much more effective in the health care sector, as professionals would be far more comfortable in using and relying on these algorithms when they understand how they work, and can validate their predictions. Moreover, understanding how the models make predictions in the health care domain may offer us valuable scientific insight into the nature and causes of specific diseases.

Bibliography

[1] Dr. William H. Wolberg, W. Nick Street, and Olvi L. Mangasarian. {*UCI*} *Machine Learning Repository - Breast Cancer Wisconsin (Diagnostic) Data Set*. 1992. URL: `https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%5C%28Diagnostic%5C%29`.

[2] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[3] Bin Dai, Shilin Ding, and Grace Wahba. *Multivariate Bernoulli distribution*. arXiv:1206.1874, 2013.

[4] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Introduction to Local Interpretable Model-Agnostic Explanations (LIME)". In: (2016).

[5] Osbert Bastani, Carolyn Kim, and Hamsa Bastani. *Interpretability via Model Extraction*. arXiv:1706.09773, 2017.

[6] Angelica Chen. *Prediction of Depression and Suicidality from Social Media Activity Using Deep Neural Networks*. 2017.

[7] Pang Wei Koh and Percy Liang. *Understanding Black-box Predictions via Influence Functions*. arXiv:1703.04730, 2017.

[8] Himabindu Lakkaraju et al. *Interpretable Explorable Approximations of Black Box Models*. arXiv:1707.01154, 2017.

[9] Paolo Tamagnini et al. *Interpreting Black-Box Classifiers Using Instance-Level Visual Explanations*. HILDA'17 Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics Article No. 6, 2017.

[10] Mengjie Yu. *Breast cancer prediction using machine learning algorithm*. 2017. URL: `http://hdl.handle.net/2152/62802`.

[11] Julia Dressel and Hany Farid. *The accuracy, fairness, and limits of predicting recidivism*. Science Advances 17 Jan 2018: Vol. 4, 2018.

[12] Xuan Liu, Xiaoguang Wang, and Stan Matwini. *Interpretable Deep Convolutional Neural Networks via Meta-learning*. arXiv:1802.00560v1, 2018.

[13] Philippe Leray and Patrick Gallinari. *Feature Selection with Neural Networks*.