Bachelor thesis
Computer Science

Radboud University

Exploring Variable Ranges in Machine Learned Models

Author:
Robin Tonen
s4486668

First supervisor/assessor:
B.P.J. Stienen, MSc
`b.stienen@science.ru.nl`

Second supervisor:
prof. dr. T.M. Heskes
`tomh@cs.ru.nl`

August 21, 2017

Abstract

Experiments, simulations and machine learned models can all be used as ways of classifying a set of data into a certain class. When the dimensionality of the input data starts getting larger, classification results like this become difficult for humans to comprehend; it's no longer possible to easily visualize how the various parameters contribute to the given answer. A novel approach for exploring and ranking the influence of input parameters of a Random Forest classifier is given. This method is successfully used to show slice thickness of an exclusion plot based on particle physics data.

Acknowledgements

Bob, thank you so much for providing me a chance to work with a subject so wholly outside of my comfort zone. Thank you for putting in the time and effort to try getting me at least the slightest bit educated about a complex and utterly fascinating field of science. Tom, please keep teaching. It's thanks to your courses that I've found the things that interest me. Data sciences are an amazing field, and I'd have never thought of it were it not for your inspirational lectures.

Contents

Chapter 1

Introduction

There's a clear hierarchy in the complexity and cost of gathering experimental data when it comes to High Energy Physics. An actual physical experiment, like the ATLAS detector at the CERN institute is an expensive project that takes a significant time to set up, run and interpret. Using a computer simulation to give an approximate prediction of such an experiment is significantly faster and cheaper, but still requires a large computing infrastructure[5]. Lastly, a prediction can be made using a Machine Learning classifier that has been trained on already produced or simulated data; taking no more than seconds on simple consumer hardware [4]. While predictions from these models can't be used to provide conclusive answers to physical questions on their own, predictions can be used to define parameters for more expensive simulations and experiments.

Using a model in this way can be difficult, as manual interpretation of results is required for deciding the next course of action. Of course we could simply brute-force our sampling of the entire parameter space; but even with the relative performance of a Machine Learned model this process scales terribly with parameter dimensionality.

Instead of brute-forcing, knowledge of the Machine Learned model can be used to provide information on parameter variance; Given a known classification, the extent to which parameters can be altered before model certainty is impacted can be gleaned from observing the model itself. This thesis presents a method for transforming a trained Random Forest Classifier to a new model suitable for such parameter variance exploration. First, explanations will be given on the basics of both machine learning and the physical theory we're hoping to explore. We'll discuss the specific machine learned model used for exploring the problem area. This model will then be rewritten to a new model that may be used more efficiently in some ways. This new model is then used to show validity and showcase example usage scenarios.

Chapter 2

Preliminaries

In this chapter, information is given about the concept of Machine Learning; Explaining the basics of how this technique is used in classification problems. Secondly, a very short introduction to a proposed theory in High Energy Physics will be presented. Finally we'll show an example of an existing Machine Learning model applied to investigating this proposed theory.

2.1 Machine Learning

Extracting meaningful information from large sets of data is a large and ever increasing field in computer science. A technique used in such data analysis is Machine Learning; a collection of several methods and techniques that allow a computer to learn about and provide insights on sets of data. A computer model is fed information in order for the model to learn properties and patterns in that data. This model can then be used to predict properties of new (similar) data [11]. Such models are often primarily or at least partly generated and modified by the computer during the process of learning about the dataset.

2.1.1 Classification

One of the areas Machine Learning can be applied to is that of labeling data. Two methods to do this are classification models and regression models; methods where knowledge of a given set of data is used to apply a label to newly presented data. The difference between the two methods lies between the resulting labels. The labels can either be said to be belonging to a series of discrete, distinct classes. Or the target label may be an element of a continuous series. In the former case, we refer to a classification model, the latter case is a regression model.The difference may best be illustrated with a pair of examples.

Classification	A model can be trained on a set of data detailing financial transactions. This data is distributed in two classes: A transaction is either a legitimate transaction, or a fraudulent transaction. New transactions fed to the model are labeled one of these two classes.
Regression	A model can be trained on weather data. Given information about a new day, it may predict the amount of rain expected to fall that day. Such a result is given on a continuous scale of, for example, millimeters.

Applying Machine Learning to these concepts is typically done by generating a *model* and presenting that model with a collection of data to train it, as illustrated in Figure 2.1. In some types of model these steps are one and the same. An example of such a model is the Random Forest model. In other types, model creation and training are distinct steps, for example a Neural Network. Finally, we can use the trained model to predict a label for a new point of data, illustrated in Figure 2.2. Variations on this can be made, including how much of the model is pre-determined, which parameters in the dataset are used for training and labeling, and whether information about classes is provided along with the training data. The case where class information is given is called supervised learning; unsupervised learning is where it's left up to the computer to distribute the data into classes on its own. In the remainder of this thesis we'll be dealing with supervised classification and classifiers.

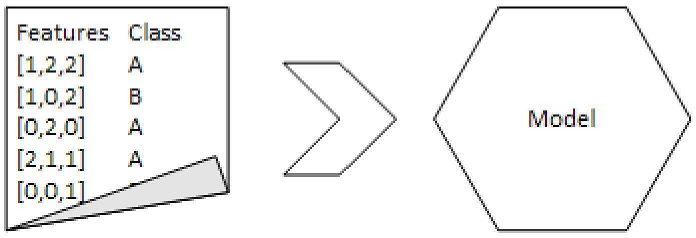

Figure 2.1: Supervised Classification training.

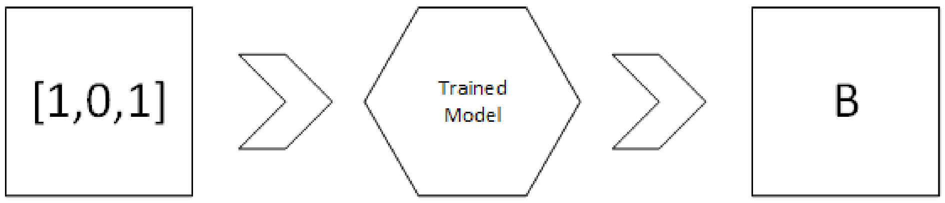

Figure 2.2: Classification using trained model.

2.1.2 Validity

The quality of a trained model is usually determined by presenting the model a set of testing data. Several measures can then be calculated from the results of this labeling. In general, for each class C we can divide the results in four groups:

True Positive (TP)	A point of data in the training set which belongs to class C, and has been correctly labeled as such by the model.
False Positive (FP)	A point of data in the training set which does *not* belong to class C, but has incorrectly been labeled as such by the model.
True Negative (TN)	A point of data in the training set which does *not* belong to class C, and has correctly not been labeled as such by the model.
False Negative (FN)	A point of data in the training set which belongs to class C, but has incorrectly not been labeled as such by the model.

There exist several measures, but the shared property is trying to maximize TN and TN rates, while minimising TP and FN rates. For example, we can calculate the accuracy of the model by taking the fraction of correctly classified points. Having access to the previously detailed numbers we may calculate it as such:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

Another pair of metrics we can calculate are the True Positive Rate (TPR) and False Positive Rate (FPR):

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

It is typical to repeatedly modify configuration variables of a model to attempt to obtain as high a classifier rating as possible. A way of visualizing such a rating is by means of an ROC curve [12], which uses the TPR and FPR. A higher rating will drift the curve towards the top left, while a pure 50/50 guess is represented by the diagonal. An example of such a curve is given in Figure 2.3.

We separate the known data used to generate the model into training and testing data to minimise the problem of overfitting. Were we to use the same

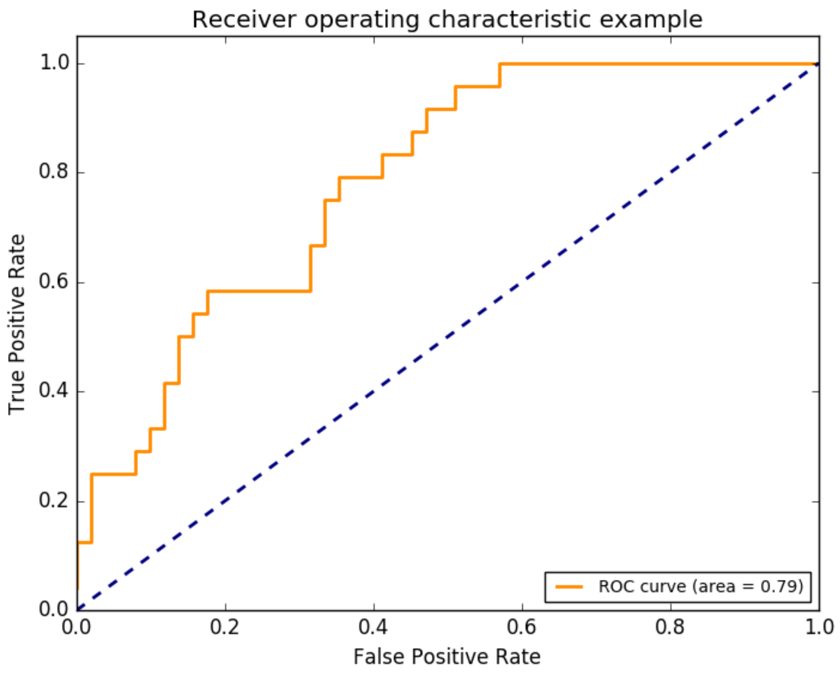

Figure 2.3: Example ROC Curve (©scikit-learn developers)

dataset for both training and testing, increasing classifier rating could very well mean a severe decrease in performance for new, unknown data. The model becomes too specifically tuned on noise in the original data, instead of the actual data characteristics that we want to train it on. Separating test and training data reduces but does not completely alleviate the problem of overfitting. An example of overfitting is given by the green line in Figure 2.4, where the black line would represent a classifier that will provide a better classification to new data points.

An important observation here is that Machine Learning operates on purely mathematical interpretation of the presented data, and has no understanding of the underlying problem or nature of the data. This is both a positive and a negative; on one side it's sometimes difficult for a computer to see a connection that a human with intimate knowledge of the problem area might easily identify. On the other hand, the computer operates without prior assumptions, and may find connections and patterns in the data completely invisible or overlooked by human observers. In the next section, we'll discuss the background and use of a classifier on a dataset.

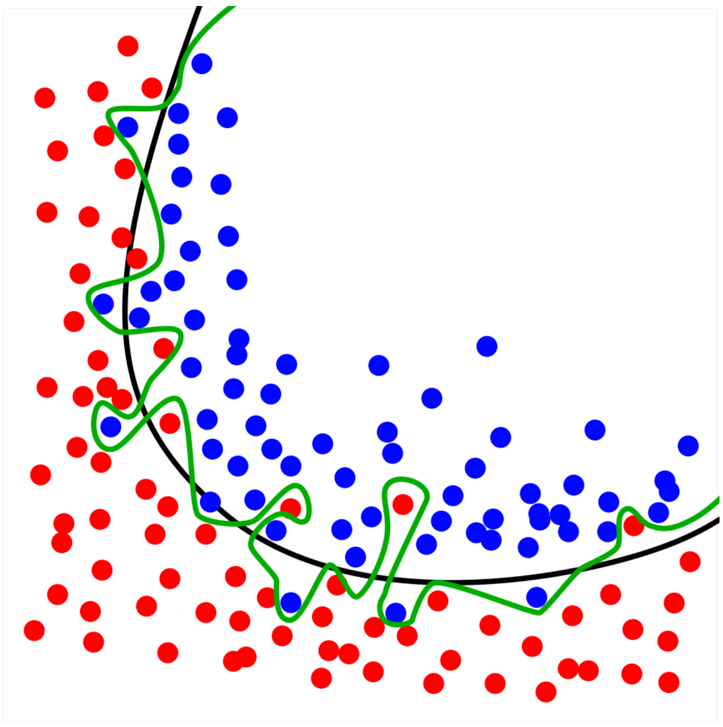

Figure 2.4: Overfitting example. Two classes (Red and blue), an overfitted classification (green, wavy line) and a more ideal classification (black, smoothed line). (ⓒCreative Commons, Chabacano)

2.2 Supersymmetry

This paper will not focus on physics. However, a brief outline is given of some relevant concepts. For a significantly more elaborate introduction to these concepts, I gladly refer to Bob Stienen's Master's thesis. [13]

2.2.1 Standard Model

An ongoing problem in physics is the search for an aptly called 'Theory of Everything'; a mathematical description of the universe and all its contents. As of now, no such model has been found that can pass experimental tests. There are however several models that explain at least a subset of physical phenomena and aspects of the universe. One such model, that does very well to explain matter and forces is the so called 'Standard Model'. This model explains matter as a composition of elementary particles, and the interaction between various particles by the exchange of force particles. A diagram showing the various elementary particles in the Standard Model is shown in Figure 2.5.

The standard model is very accurate in describing these parts of nature, and experiments show no significant deviation between its predictions and actual measurements. However, there are some known shortcomings to the model. Most importantly, it fails to explain several concepts in physics;

Dark Matter	The presence of more matter than just the optically visible matter has been theorized and hinted at by experiments. This matter is referred to as 'Dark Matter', alleging to its yet unobserved nature. No particle in the Standard Model is fulfills the requirements to be able to be this Dark Matter, either because such a particle cannot exist in sufficient quantities for extended periods of time, or because such a construct would already have been observable.
Gravity	Various physical models exist to explain gravity at a quantum scale. However, such models are incompatible with the Standard Model; which does not contain a particle that can adequately account for gravity.

These, and various other problems with the Standard Model call for a replacement or extensions of the model, in order to explain a greater portion of, if not the entirety of the universe and its contents.

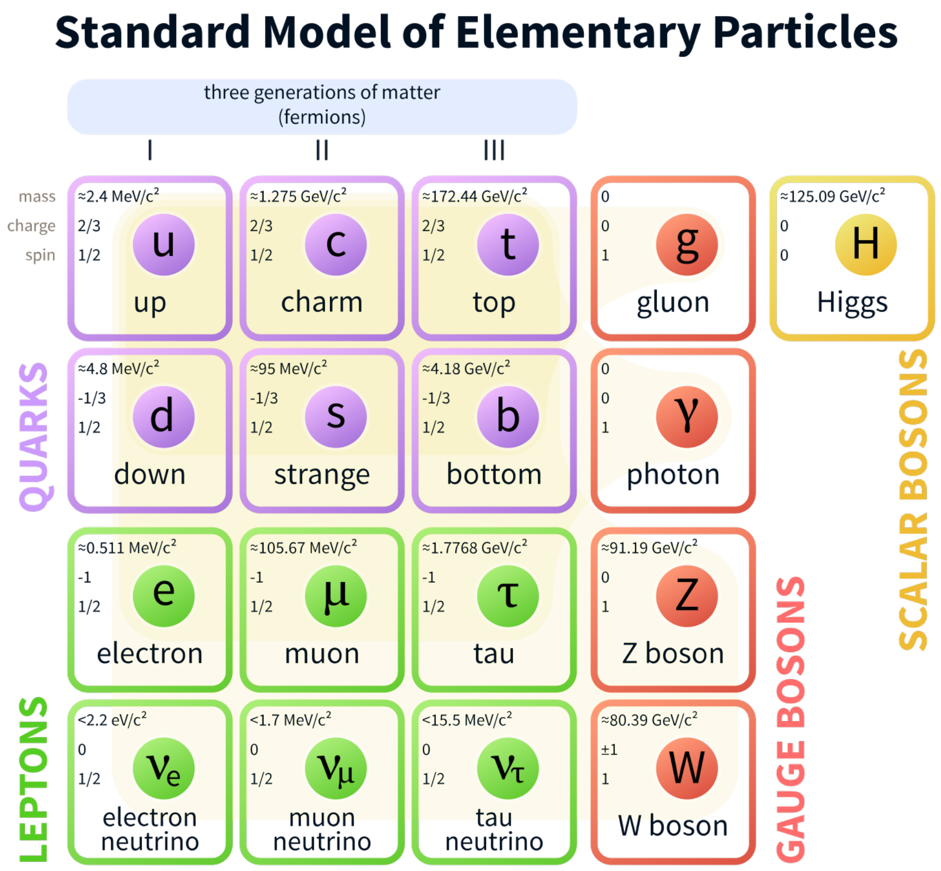

Figure 2.5: Elementary Particles in the Standard Model (🛈Creative Commons, Cush)

2.2.2 Supersymmetry

One such theorized extension is called Supersymmetry. In this proposed extension, the particles in the Standard Model account for only about half the existing particles, the other half are matching particles with identical parameters and quantum numbers; differing only in the property of spin with their observed counterpart. [10] Such a construction would explain some of the deficiencies in Standard Model, and would allow for the unification of known forces to a single unifying force. However, it relies on the assumption of existing particles that have not yet been observed. [8]

Exact symmetry like this isn't realized in nature, as such a situation would have meant the missing particles would have long since been detected by virtue of them having identical masses to already observed particles. This means that if such a symmetry exists, it is broken, where additional slight differences between particle properties and their Standard Model counterparts are introduced.

2.2.3 pMSSM

Introducing the possibility of such slight differences creates a problem of dimensionality. Broken supersymmetry introduces at least 105 free parameters. This configuration is called the Minimal Supersymmetric Standard Model (MSSM). This is significantly too much for any feasible search or exploration. However, several assumptions can be made while focussing on keeping the theory as general as possible, and making as little deviations from existing rules as possible. By making such assumptions, we can reduce the problem to the so called phenomenological MSSM (pMSSM), having only 19 free parameters.

Research into this model (and other models beyond the Standard Model) is aided by the Large Hadron Collider: the largest particle accelerator in the world. At the LHC, the ATLAS experiment is capable of searching for deviations from the known Standard Model in various processes.

As it stands, no significant deviations have been found, which means the experiments can currently only be used to set limits on the parameters of various Beyond Standard Model theories, including the pMSSM. In the most simple terms; experiments can determine parameter values that are excluded in the theory. Were the chosen parameters be the correct, a deviation would have already been detected in the experiment. In such a manner, LHC experiments can be used to provide exclusion limits for models such as the pMSSM. [6]

2.3 SUSY-AI

2.3.1 Computer Simulation

Physical experiments such as performed with ATLAS require both extensive resources and time to perform. Unguided search of exclusion limits with all possible configurations of particles in particle accelerator and detectors is prohibitively expensive and time consuming. Instead of physical experiments, a computer simulation can be run to provide at least an approximation of the results.

However, given the complexity of the physical model; computer simulation of the experiments is computationally extremely expensive. Too much so to be performed on readily available consumer hardware. The simulation infrastructure and the data used and produced by the simulations is not commonly publicly available. However, a large dataset was made available [6], and has been used for generating a Machine Learned model.

As detailed above, Machine Learning allows us to use a dataset such as the one produced by the simulation to train a model. This model can then predict the outcome of future simulations in a significantly less computationally expensive manner. These predictions can then be used to decide whether or not a full simulation and analysis is required.

A second goal of such a model is the possibility of finding a deeper analysis of the underlying data. As explained above, machine learned models have no understanding of the actual phenomena creating the data, and operate purely on mathematical properties found during analysis of the testing data. This may result in a previously unseen understanding of the data, based on patterns in the data located by the model's training. Such a model has been made with the SUSY-AI program. [4]

2.3.2 Further Analysis

The model generated by SUSY-AI is significantly less computationally expensive than the full simulation and can easily be run on consumer hardware. However, this does not remove the need for manual interpretation of the data produced by the classifier. Full exploration of the parameter space would require still brute forcing; an operation that which computational complexity scales exponential with the amount of variables. Even the reduced 19 parameters of pMSSM and the low runtime of the machine learned model mean a significant computational investment. This means that directing the search is a manual job, both in determining interesting points for full simulation, as well as for attempting to find patterns in the model.

Instead of brute forcing, it might be possible to adapt the generated model to ease the locating of limits to the various parameter spaces that lead to a certain classification. A method for this is researched in the next chapter, and applied to the data SUSY-AI is based on. [6]

Chapter 3

Research

We'll define and dissect the Random Forest classifier that Susy-AI uses, trying to transform the existing model into an equivalent one with different properties that may aid in some of the exploration such models can be used in. This chapter will detail the generation of such a model from a Random Forest classifier, show several example algorithms, and finally apply these algorithms to show possible usage in tandem with Susy-AI.

3.1 Dissecting Random Forest Classifiers

3.1.1 Model Explanation

A Random Forest classifier consists of a collection of Decision Tree classifiers [2]. Decision Trees are relatively simple classifiers; a data point is sent through a series of nodes, where each node tests the value of a single attribute, and splits the data to one of two lower nodes. This is repeated through a path of nodes until a leaf node is reached, which contains the label to be attached to the data point.

We can think of each node as dividing the remaining parameter space in two regions over the attribute it specifies. Training of the model is performed by repeatedly finding an attribute and value that best splits the data in two groups corresponding to the desired classes. An example of a tree with two nodes and three leaves is given in Figure 3.1. This tree uses two attributes to split the data into one of two classes.

For classification in the Random Forest classifier, a data point is fed through each Decision Tree classifier in the forest. The resulting labels from each Tree are aggregated to provide a certainty measure for the label returned by the Random Forest. The different trees are trained on different subsets of

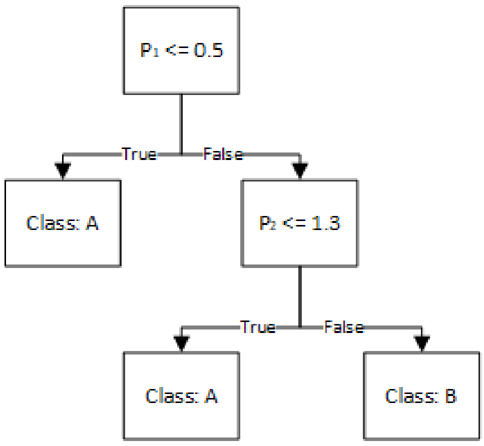

Figure 3.1: Decision Tree Classifier

the training data set, and on different subsets of the various dimensions of the attribute space. [2]

We can try and roughly but formally define a Random Forest in Backus-Naur Form [7], first specifying it as nothing more than a collection of Decision Trees:

$$RandomForest ::= \quad Trees \quad :[DecisionTree]$$

Where each Decision Tree is simply a pointer to a root node.

$$DecisionTree ::= \quad Root \quad :Node$$

Leaving for us to define what a node is, in this case it is either a leaf node; specifying what class this node belongs to. Or it is a node with a left and right descendant. In this case we need to define what attribute we're working on, and what value we're splitting on. By convention, we'll split $Dimension \leq Value$ to the left node, and $Dimension > Value$ to the right node. (This is equivalent to the scikit-learn implementation of such a classifier [3])

$$\begin{aligned} Node ::= \quad & |Leaf && :Class \\ & |Left && :Node, \\ & \ Right && :Node, \\ & \ Attribute && :Dimension, \\ & \ Value && :Number \end{aligned}$$

3.1.2 Different Perspective

As noted above, we can think of each node as dividing the remaining parameter space in two regions in a single dimension. This means that, over the total parameter space, a Decision Tree classifier can be seen as a collection of non-overlapping regions with complete cover of the parameter space; each region representing a label to be given as a resulting classification for every point located in that region.

Because the parameter space is identical for each classifier, and each classifier's regions do not overlap, a single point in the parameter space always falls within exactly X regions, where X is the number of Decision Tree classifiers in the Random Forest. Instead of representing the Random Forest as a collection of nodes, we can represent it as a collection of regions.

Similarly to our effort in formally specifying a Random Forest classifier, we can specify our Region Model replacement, starting with specifying the model as a collection of Regions:

$$\begin{array}{rll} RegionModel ::= & \mathit{Regions} & :[\mathit{Region}] \end{array}$$

And then specifying each region as an ordered list of boundaries, and a class that this region belongs to.

$$\begin{array}{rll} \mathit{Region} ::= & \mathit{Boundaries} & :[\mathit{DimensionalBound}], \\ & Class & :Class \end{array}$$

Finally specifying the boundaries as having a start and an end. To match with the previously made choice of left and right nodes; The Start value is taken as an exclusive boundary, while the End value is taken as an inclusive boundary.

$$\begin{array}{rll} \mathit{DimensionBound} ::= & \mathit{Start} & :Number \\ & \mathit{End} & :Number \end{array}$$

3.1.3 Consequences

Representing the model in this way has a few negative consequences; it is significantly more space-inefficient than the node representation, and classifying a single data point by means of checking region overlap is much more computationally intensive than in the node representation. There is however one major benefit of this representation; regions can be sorted in accordance to a dimension. A summary of complexity consequences is given in Appendix A.1

Once sorted, we can traverse region boundaries one by one, subtracting and adding the region count for each class. In this way we can efficiently find the exact boundary where the Random Forest classifier will cross a given certainty threshold.

3.2 Creating Region Model

3.2.1 Algorithm

Having already formalized the input and output types, we can specify the function to be made as a function $F : RandomForest \rightarrow RegionModel$. We can write the following pseudocode as Algorithm 1.

Algorithm 1 Generating Region model from Random Forest model

```
function GenerateRegions(RandomForest)
    $Regions \leftarrow \emptyset$
    for all Tree in RandomForest do
        $Working \leftarrow \emptyset$
        $Node \leftarrow Tree.Root$
        for all Dimension in Tree.Dimensions do
            $Working \leftarrow Working + (-\infty, \infty)$
        end for
        function Recurse(Working, Node)
            if Node = Leaf then
                $Regions \leftarrow Regions + (Working, Node.Class)$
            else
                Recurse(Working[Node.Attribute].End = Node.Value, Node.Left)
                Recurse(Working[Node.Attribute].Start = Node.Value, Node.Right)
            end if
        end function
    end for
    return Regions
end function
```

We'll start with an empty set of Regions, which we'll fill with the regions obtained from the various trees. For each tree, we initialize a working set, giving boundaries from $-\infty$ to ∞ for each dimension. Then, we recursively iterate the nodes, starting at the root of the tree.

If we hit a leaf node, we can add a region to the set of regions, using the dimension limits of the working set, and marking the class as the one belonging to the leaf node. On any other node, we need to recursively iterate both the left and right nodes. We'll modify the working set by respectively changing the end or start attribute of the correct dimension to the value given by the node. In Figure 3.2 we can see the resulting regions generated from the tree in Figure 3.1.

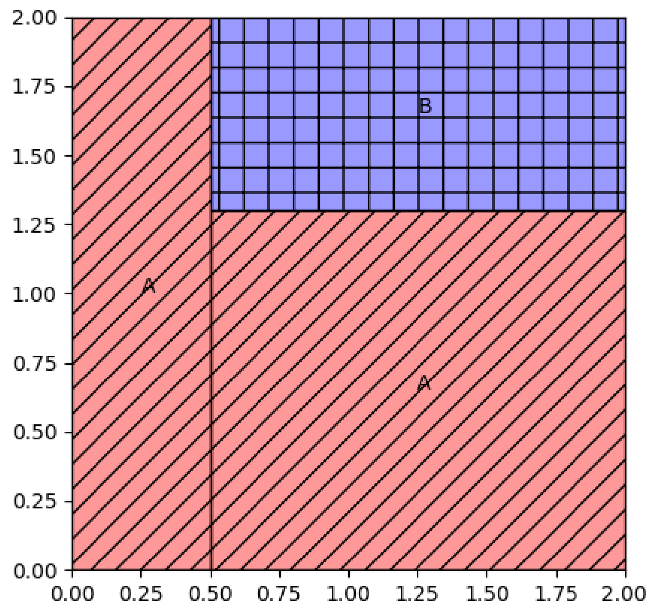

Figure 3.2: Region Model of the tree in Figure 3.1

3.3 Using the Model

3.3.1 Classification

The generated Region Model may be used in several ways. We can for instance use it as a drop in replacement for the Random Forest Classifier by simply checking what regions a given input falls in. We start with all the boundaries, and remove them if the given parameter falls outside of the region. Note that this operation is likely significantly slower than classification via the Random Forest classifier, and is shown only to provide context of equivalence in classification. Classification is detailed in Algorithm 2.

Algorithm 2 Classification using Region Model

```
function CLASSIFY(Input, RegionModel)
    $Boundaries \leftarrow RegionModel.Boundaries$
    for all Parameter in Input do

        for all Boundary in Boundaries do
            if Boundary[Parameter].Begin $\geq$ Parameter.Value or Boundary[Parameter].End $<$ Parameter.Value then
                $Boundaries \leftarrow Boundaries - Boundary$
            end if
        end for
    end for
    return COUNT($Boundaries.Class$)
end function
```

3.3.2 Boundary Checking

Far more interesting is to use the generated Region Model to to boundary checking. An example is given in Figure 3.3. In this example model with two trees, we're interested in the boundaries for the feature represented by the x-axis. The arrows represent the amount we can vary this feature while still being at least 50% certain of our class prediction being 'blue'. For such an algorithm, we'll use the fact that sorting can be done in efficient time (More discussion on complexity is done in Appendix A.1). We'll create the algorithm for finding the boundary in a single dimension. Our first task is to collect the relevant regions. The code to this is almost identical to Algorithm 2. However, we exclude the dimension that we want to find the limits for. At the end, we disassemble the region's boundaries, taking only the ones in the dimension we're interested in.

An example of applying Algorithm 3 for finding the relevant boundaries is shown by Figure 3.3, where for a two-dimensional mock region model with two trees and an input $(5, 5)$, we have selected every relevant region in the dimension represented by the X-Axis. The rest of the regions are already hidden.

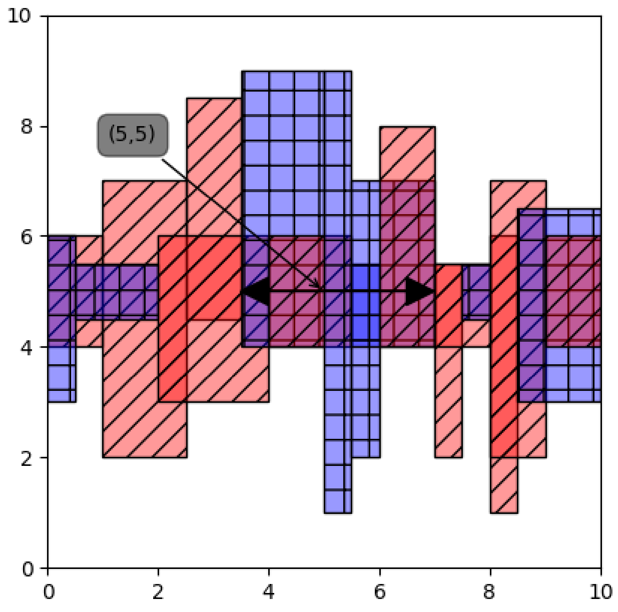

Figure 3.3: Example mock region model, including boundaries for class represented by the blue regions at point (5,5). Only the regions in the relevant x-axis are shown.

Algorithm 3 Selecting relevant Boundaries for a given dimension and input

```
function RelevantBoundaries(Input, RegionModel, Dimension)
    $Regions \leftarrow RegionModel.Boundaries$
    for all Parameter in Input \ Dimension do

        for all Region in Regions do
            if $Region[Parameter].Begin \geq Parameter.Value or Region[Parameter].End < Parameter.Value$ then
                $Regions \leftarrow Regions \setminus Region$
            end if
        end for
    end for
    return $\text{Sort}(Regions.Begin[Dimension] \cup Regions.End[Dimension])$
end function
```

Since we now have a sorted list of the relevant boundaries, we can very efficiently traverse this list in either direction. Each boundary crossed removes one from the count of a class, while adding one to the count of another class. This property can be used to find the limit to which we can vary the requested dimension in the input data while maintaining a given certainty in the returned class.

Algorithm 4 Finding the lower limit for a given input and dimension.

```
function LOWERLIMIT(Input, RegionModel, Dimension, DesiredClass, Threshold)
    $Stack \leftarrow$ RELEVANTBOUNDARIES($Input, RegionModel, Dimension$) $\leq Input[Dimension]$
    $Counter =$ COUNT($RegionModel$)$[DesiredClass]$
    while $Counter \geq Threshold$ do
        $Boundary \leftarrow Stack.Pop()$
        if $Boundary = End$ then
            if $Boundary.Class = DesiredClass$ then
                $Counter \leftarrow Counter + 1$
            end if
        else
            if $Boundary.Class = DesiredClass$ then
                $Counter \leftarrow Counter - 1$
            end if
        end if
    end while
    return Boundary.Value
end function
```

Algorithm 4 is given for finding the lower limit, but an equivalent algorithm can be easily created for the upper limit. Alternatively, constructions can be made that keep track of every class, rather than just a single one. Most of the constructions of this type can be performed in $\Theta(N)$, since they only require a single pass of already sorted data.

3.4 Slice Thickness in SUSY-AI data

3.4.1 Problem Definition

Visualizing data from experiments, simulations and model predictions is an effective means of showing results and patterns in the data. However, while visualizing data on two, three or with some constructs slightly more dimensions can be easily performed in a variety of manners; there are few if no ways of presenting data in as much as the 19 dimensions that pMSSM requires.

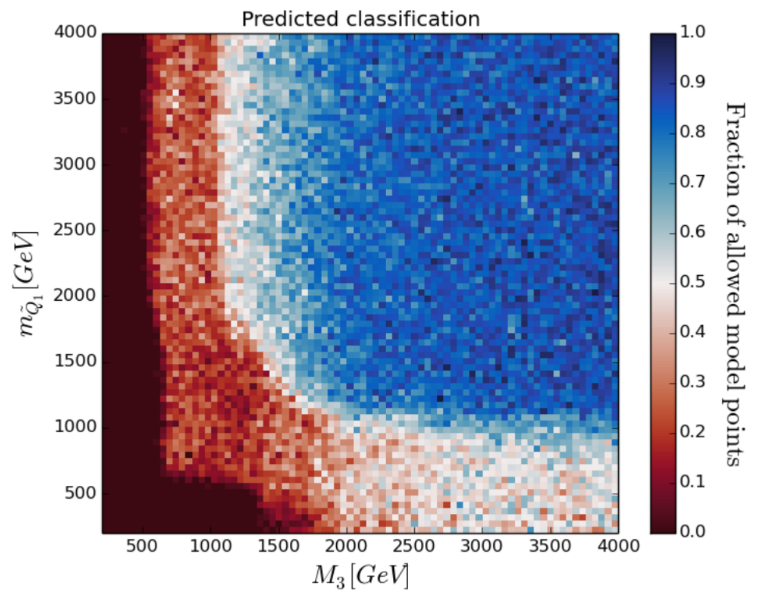

Figure 3.4: Example slice of pMSSM data. (Taken from [13])

As such, plots are often given in slices, where only two parameters are tweaked, and the remaining 17 are either taken as a static value or simply projected unto the shown plane. Such a slice is easily visualized, but deceptive in that it says absolutely nothing about the close proximity in any other dimension; There is no telling how representative the data actually is outside of the static parameters given. An example of such a representation is given in Figure 3.4, showing the percentage of points with all measured combinations of the remaining 17 parameters on a single point in the two remaining dimensions.

We can use the Region Model to augment such a plot with more information, to give at least an idea of how representative the slice is for the surrounding data. Instead of colouring the various cells of the plot with the certainty of the prediction, we will colour them with a measure of the lowest distance in the remaining 17 dimensions before a change in predicted class is observed.

In this way, we can give a measure of slice thickness across the plot; points with a high value mean the slice is representational in this point for a large variation in values along the other dimensions, while points with a low value mean the slice is less representational for at least one dimension in that point.

3.4.2 Code

An example can be given that produces a flat projection of two dimensions, leaving the other dimensions for our region model. The two dimensions will be referred to as x- and y-dimensions, reflecting their locations in the plot. It's important to note that this example works with a given plot size and precision. We'll input X and Y ranges as lists of values. We'll also need a base value for the remaining dimensions.

Algorithm 5 Creating a slice representation.

```
function SLICE(Input, RegionModel, $X_{range}$, $Y_{range}$)
    $Results \leftarrow \emptyset$
    for all X in $X_{range}$ do
        for all Y in $Y_{range}$ do
            $Point \leftarrow input[x \leftarrow X, y \leftarrow Y]$
            for all Feature in $input \setminus \{x, y\}$ do
                $Lower \leftarrow Feature - LowerLimit(Point, RegionModel, Feature)$
                $Upper \leftarrow UpperLimit(Point, RegionModel, Feature) - Feature$
                $Width \leftarrow Minimum(Lower, Upper)$
                $Results[X, Y] \leftarrow Minimum(Results[X, Y], Width)$
            end for
        end for
    end for
    return Results
end function
```

Algorithm 5 details this operation, but in essence we simply check the upper and lower limit of each feature not used as a grid dimension, and extract the minimum value out of that.

3.4.3 Example Results

In Figure 3.5 we can see that using the region model in this fashion produces results that can be considered sane. Boundaries of the existing Figure 3.4 are evident in the new Figure 3.5 as well, and such a verification can be even more clearly shown by using the region model to produce these boundaries

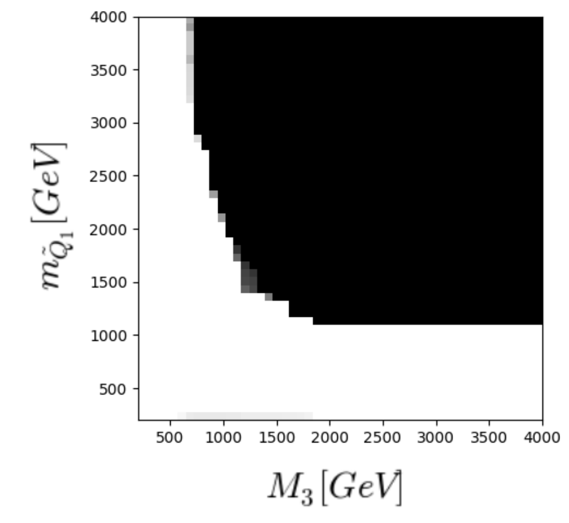

Figure 3.5: Example slice with feature width information. Whiter areas are wider in all dimensions, darker areas are thinner in at least one dimension.

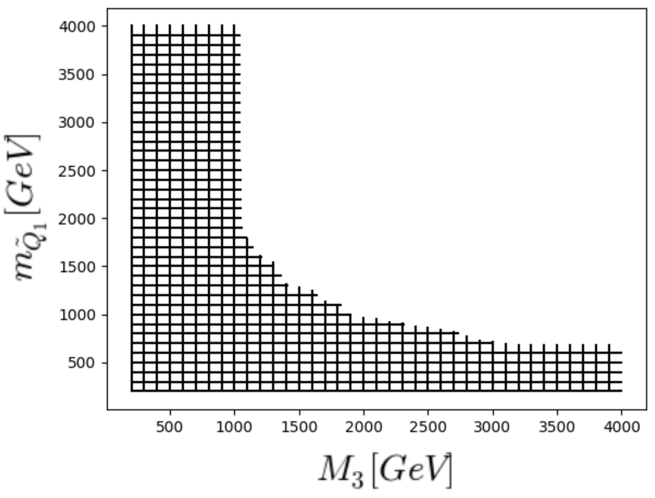

Figure 3.6: Boundaries using region model. Lines show the area of the flat slice with at least 50% certainty in predicting exclusion.

itself, as we have done in Figure 3.6. To produce these boundaries, we repeatedly run a boundary finding algorithm such as Algorithm 4, choosing points along the edges of the desired plot as the input point.

It can be observed that slice thickness is lower near some of the edges, this is perfectly sensible, as thickness can be expected to lower as we approach a point where the predicted class changes. The example used shows high values for all other points. This suggests that, at least for these dimensions and these input values, no unexpected holes in regions are found.

Sadly, much of this method remains brute-forced, and the solution; while faster than complete brute-forcing, is still a time-intensive process. Running a single iteration of the boundary-search such as 4 is a process that takes a matter of minutes on consumer hardware. Depending on search space size and desired resolution, the algorithm needs to be run hundreds if not thousands of times. This combines to a runtime of several hours on consumer hardware to produce a single plot such as Figure 3.5.

Chapter 4

Conclusions

It is possible to transform a Random Forest classifier into a different model that encompasses the same data in a different format. This format can be used for several applications. Using the region model format for applications that can also be solved by the existing Random Forest classifier shows identical results to said classifier. Example applications that cannot be efficiently solved by means of the Random Forest classifier, and make use of the region model's property of being sortable show that such a model can be used to provide answers to meta-questions on the ML classifier and its problem domain.

The model is however highly inefficient in tasks where the classifier itself would be appropriate, and many tasks where it replaces pure brute-forcing with the classifier still retain a (smaller) element of repetitive application and brute-forcing. Such questions remain a large investment of both computational and time resources regardless of method, especially keeping in mind the added cost of creating and storing the region model. While for select applications the region model might be usable and appropriate, it is not widely effective.

Bibliography

[1] Christian Bessiere, Emmanuel Hebrard, and Barry O'Sullivan. Minimising decision tree size as combinatorial optimisation. *Lecture Notes in Computer Science*, 2009.

[2] Leo Breiman and Adele Cutler. Random forests. `https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm`, 2001.

[3] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, et al. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[4] S. Caron, J.S. Kim, K. Rolbiecki, et al. The bsm-ai project: Susy-ai – generalizing lhc limits on supersymmetry with machine learning. *Eur. Phys. J. C*, 2017.

[5] The ATLAS Collaboration, G. Aad, B. Abbott, et al. The atlas simulation infrastructure. *Eur. Phys. J. C*, 2010.

[6] The ATLAS Collaboration, G. Aad, B. Abbott, et al. Summary of the atlas experiment's sensitivity to supersymmetry after lhc run 1 — interpreted in the phenomenological mssm. *Journal of High Energy Physics*, 2015(10), 2015.

[7] D. Crocker and P. Overell. Augmented bnf for syntax specifications: Abnf. STD 68, RFC Editor, January 2008. `http://www.rfc-editor.org/rfc/rfc5234.txt`.

[8] H.E. Haber and G.L. Kane. The search for supersymmetry: Probing physics beyond the standard model. *Physics Reports*, 1985.

[9] J Kent Martin and DS Hirschberg. On the complexity of learning decision trees. In *International Symposium on Artificial Intelligence and Mathematics*, 1996.

[10] Stephen P. Martin. A supersymmetry primer. Master's thesis, University of Michigan, 1997.

[11] Tom M. Mitchell. The discipline of machine learning. `http://www.cs.cmu.edu/~tom/pubs/MachineLearning.pdf`, 2006.

[12] Andrew P.Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 1997.

[13] Bob Stienen. Generalising lhc exclusion limits using machine learning. Master's thesis, Radboud University Nijmegen, November 2016.

Appendix A

Appendix

A.1 Complexity

While generation of a Random Forest remains a decently time-intensive and complex matter [9], actually doing a prediction by means of a trained model is quick and computationally cheap. Each tree in the forest has a worst-case time complexity in $\mathcal{O}(H)$ where H is the height of the tree. We can reason the total complexity of the forest as being in $\mathcal{O}(NH)$, with N being the number of trees. Depending on implementation the algorithm can be ran in parallel for N.

For space complexity, a decision tree has a number of nodes in $\mathcal{O}(2^{H+1} - 1)$, methods exist to try and minimize the size of a tree, ensuring a minimal amount of nodes [1]. For a Random Forest classifier we of course need N trees. This means that Random Forests can be quite space-costly classifiers. However, time-complexity of running a prediction is very low. This is a very handy property of random forests, which sadly the region model doesn't copy.

A.1.1 Model Generation

Generating the Region model is theoretically a time efficient solution. Our algorithm recursively passes each node exactly once and performs an $\mathcal{O}(1)$ operation. This gives us a total complexity of $\mathcal{O}(N)$ in the size of the Random Forest. We have however established that Random Forests are rather big, so effective computational cost is still pretty big simply due to model size.

The resulting model is, like the model it was generated from, relatively sizeable for a Machine Learned model. Each decision tree has at most $\mathcal{O}(2^H)$ leaf nodes. Each leaf node produces a region, with the size of the data representing this number dependent on the number of features. If we take the random forest as a whole this gives us a resulting region model size of $\mathcal{O}(FN2^H)$ on the number of trees (N), the number of features (F) and the height of the trees(H)

A.1.2 Model Usage

Basic operations on the region model are more complex than the equivalent operation on the random forest. To perform the classification algorithm, we need to consider each region and check whether the input falls withing the boundaries of that region. While this is an operation in $\mathcal{O}(RF)$, with F the number of features, and R the number of regions; not the number of trees like N in the Random Forest. As explained in the last paragraph, the number of regions scales exponentially with tree size. While the theoretical time complexity is similar to that of the random forest, the size of the numbers involved is significantly larger.

This is the primary problem of the region model. The random forest by its very nature allows model traversal to skip the majority of nodes, as usually only a single path to a leaf node is required. With the region model, almost every operation requires checking each region. Due to the number of regions being very large, this simply isn't efficient.

The real advantage lies in the regions being sortable, and as such we can replace brute forcing, which requires many iterations of an algorithm using the random forest with only a single iteration of an algorithm using the region model. While that single iteration might be costly, it may be more efficient than repeatedly performing a cheaper algorithm.