MASTER

Instance-level decision visualization of Random Forest models

Collaris, D.A.C.

Award date:
2018

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
Visualization Research Group

Instance-level decision visualization of Random Forest models

Master Thesis

D.A.C. Collaris

Supervisors:
Prof. dr. ir. J.J. van Wijk *(TU/e)*
Prof. dr. M. Pechenizkiy *(TU/e)*
L. Vink BSc *(Achmea)*

Eindhoven, January 2018

Abstract

Machine Learning models get better and better at various tasks. However, along with these improvements, the complexity of these models also rapidly increases. This negatively affects the comprehensibility of these models. In this thesis, we defend the case for comprehensibility of a specific predictive model, called a Random Forest.

To this end, we show a comprehensive overview of currently available interpretation methods: feature analysis, simpler model derivation and structural visualization. We analyze the effectiveness of the most promising methods, as well as various visualization techniques in order to effectively convey which choices a model makes.

In our endeavors, we found that explanations of a single prediction (*instance-level*) are underrepresented in literature, as opposed to global explanations. We explore the possibilities of local model explanations and develop a local rule extraction technique that is, to the best of our knowledge, a novel approach in obtaining model insights.

We combined various visualization techniques into two dashboards that facilitate a dialogue between users and a given Random Forest model. The first dashboard is centered around features, giving a per-feature explanation of its contribution to the prediction. The second dashboard takes the possible target classes as a starting point, and uses model simplification to present a set of rules that describe the choices the model made for the prediction of those classes.

These dashboards were evaluated through a case study at Achmea (one of the leading providers of insurances in the Netherlands) within the context of insurance fraud detection. We use the standardized System Usability Scale (SUS) survey to quantify the satisfaction of the fraud team at Achmea, and show that these dashboards can aid them with their day-to-day tasks.

Acknowledgements

First of all, I would like to thank Prof. dr. ir. Jack van Wijk for his guidance, encouragement and enthusiasm during the project. Even though the project focused less on visualization as initially anticipated, we always had interesting discussion about the project, which even caused him to forget the time and miss a meeting once.

Next, a special gratitude goes out to Leon Vink, for picking me up every week and having interesting discussions about Data Science, Achmea, and leadership in the car. His help was essential for me to get started quickly and find my way within the company. Also a big thanks to Achmea for giving me the opportunity for this graduation project, and providing me with anything I required.

Furthermore, I would like to thank Prof. dr. Mykola Pechenizkiy for his inspiring inaugural lecture. His enthusiasm helped me to deal with some major setbacks during the project, and encouraged me to press on. This also inspired me to pursue a PhD as a continuation of this thesis.

With a special mention to Senna, Joost and Yorick for taking the time to plow through my document and provide me with feedback during the final moments of the project.

And finally, last but by no means least, I wish to express my gratitude towards my girlfriend Daphne, who kept supporting me throughout the project. She most certainly heard every line written in this report twice and was always prepared to listen to another theory or problem.

Thank you all for your encouragement!

Contents

Chapter 1

Introduction

The retrieval and storage of data is getting cheaper and more ubiquitous. Companies are accumulating more data than ever before, sometimes even exceeding their capacity to extract value from it [87]. To deal with this issue, new technologies are emerging to make sense of these massive amounts of data. We can use predictive modeling in order to find meaningful patterns in data sets that could have otherwise never been found due to the sheer size. Predictive modeling is defined as the process of developing a mathematical tool or model that generates an accurate prediction [62, 66].

For example, if it is possible to model the characteristics of fraudulent users within a huge collection of user data, then it would also be possible to automatically report high-risk subjects to the stakeholders. They, in turn, will be able to verify the validity of this claim and take subsequent action. Finding such a model, however, is a non-trivial task: the data are only a few samples within the full feature space, and there is often no clear distinction between the predicted class and noise.

One particularly useful tool data scientists use is Supervised Machine Learning. This is the collective name for algorithms that generalize a mapping between test data input and a feature to be predicted, through analyzing a set of training data with ground truth. As this generalization extends beyond the training input to encountered cases (called inductive bias [82]), the learned model can be used as a predictive model.

These algorithms can help to deepen our understanding of our data, but as Freitas [35] points out, the vast majority of work to date concerns classification models that are evaluated using only predictive accuracy [50, 54, 103]. This has led to the creation of many so-called black box Machine Learning models: models that are so complex that it is not straightforward to reason about their inner workings directly. Examples of these include ensemble models [88] and neural networks [79]. The predictive accuracy of these models may be high, but this is achieved at the cost of comprehensibility.

In the 90's this problem was recognized by the scientific community [59, 94, 28, 29], yet this still did not have a lot of impact on the limited focus on comprehensibility. Only lately the field seems to have regained interest in comprehensibility aspects [41, 36, 73, 35, 61, 62, 43].

Comprehensibility can be viewed in two different ways: globally, where we consider the behavior of the entire model, or locally, where we only consider a subset of the behavior of the model. A local explanation is called *instance-level* when it only concerns the prediction of a model for a single instance. This can be very beneficial in cases like fraud detection [18], illness diagnosis [62, 15] or bankruptcy prediction [117], where the consequences of a bad decision are significant, and the problem is difficult to predict. In such cases an explanation per instance can be crucial for analysts to make a judgment on the model prediction. In Chapter 4 we elaborate on the work so far on making models more interpretable.

One of the models that is particularly difficult to comprehend is a Random Forest model. The algorithm was first devised by Ho [48] in 1995 and later popularized and trademarked by Breiman [11] in 2001. As of today, it still proves to be one of the most accurate models for classification [74, 43, 7], however it cannot be considered very interpretable. A Random Forest model is comprised of decision trees that are generally considered interpretable individually [35], however, the sheer size of multiple trees combined still makes it difficult to reason about and understand the model.

1.1 Thesis objectives

In this thesis, we address the following research question:

> How can we make the instance-level decision making process of a Random Forest comprehensible?

In order to answer this main question, we need to answer the following subquestions:

1. How do we define comprehensibility?
2. What preexisting methods are applicable to Random Forest methods?
3. How effective are these methods?
4. Can we improve upon the state-of-the-art regarding Random Forest visualization?
5. Which visualizations should be part of a dashboard that facilitates analysis of a Random Forest model?

This project is carried out in collaboration with Achmea BV, a leading insurance company in the Netherlands. They provided the data set and a Random Forest model trained for detecting sick leave insurance fraud. Their use case is to analyze why the model has classified a specific case as fraudulent. The questions and requirements from Achmea are described in more detail in Chapter 3. Although this use case is similar to the main motivation for our research, we aim at a general solution that should be applicable to any Random Forest model and provide a comprehensible explanation of its predictions.

1.2 Thesis scope

This thesis concerns the fields of Machine Learning and Visualization. There is a massive amount of research that touches on these subjects. To keep this project manageable, we mention some constraints on the thesis scope.

First, within Machine Learning, we constrain ourselves to Random Forests only. As a few model-agnostic approaches already exist [63, 61, 62, 110, 98, 108, 38, 55], we hope to achieve more accurate results by using Random Forest specific methods.

We also constrained our solution to binary problems only. This means that the model only outputs one of two different classes. To the best of our knowledge, the majority of data sets contain only two classes. Having more than two classes could pose further complications for the visualization. Additionally, most higher-class data sets can be reduced to binary problem sets.

1.3 Thesis structure

The remainder of this thesis is structured as follows: in Chapter 2 we provide preliminary knowledge about the problem domain. This section also answers *subquestion 1*. Next, the problem is analyzed in detail in Chapter 3, with specific focus on the questions, user tasks and requirements posed by Achmea. A taxonomy of related work is presented in Chapter 4, showing which state-of-the-art techniques are currently available to gain insights into Random Forests. This section answers *subquestion 2*. The visualization of these techniques is described in depth in Chapter 5, and the effectiveness of these methods, corresponding with *subquestion 3*, is evaluated in Chapter 6. In that chapter we also show a novel method for model explanation called local rule extraction (*subquestion 4*). Next, we use the visualization techniques and insights described in the previous chapters to create two dashboards, which are showcased in Chapter 7 (*subquestion 5*). These dashboards are evaluated through a case study at Achmea in Chapter 8. Finally, achievements, limitations and future work are provided in Chapter 9.

Chapter 2

Preliminaries

2.1 Comprehensibility

As Freitas [35] points out, the vast majority of works to date concerning classification models only use predictive accuracy as the evaluation criterion [50, 54, 103]. Often used evaluation criteria are predictive accuracy, precision/recall, Area Under ROC-curve (AUR) [94] and Out-Of-Bag (OOB) error. This shows the primary goal of most current predictive models: to represent patterns within a data set as accurately as possible.

This has resulted in the creation of many so-called black box Machine Learning models: models that are so complex that it is hard to understand their inner workings. They only provide a classification result; nothing more about the process is being exposed.

Model complexity can be high due to the nature of the model, like for instance neural networks [79], Support Vector Machines (SVM) or other kernel-based learning methods [20]. But the sheer size of a model can also limit comprehensibility. Ensembles [88] are a popular method where multiple simpler models are combined to gain predictive accuracy. Although the individual models are often simple and easy to interpret, the complexity of the entire ensemble can be orders of magnitude higher than the sum of the parts. For instance, decision trees are generally considered to be very interpretable [35]. However, it becomes impossible to generate useful insights from the model when the tree becomes excessively large [32, 36, 2].

Note that various alternative terms are used to describe comprehensibility: amongst others interpretability, understandability and explainabilty. In this chapter we will use terminology that the original papers used in their work.

2.1.1 Focus in literature

In early Machine Learning research, the distinct lack of focus on comprehensibility was recognized and addressed. Already in 1994, Kodratoff wrote the "Comprehensibility Manifesto" [59] in which he pleads that the community should recognize comprehensibility as an important research field and incorporate it more in Machine Learning model development. He provides empirical evidence that when Machine Learning approaches are applied in industry, the interpretability, though ill-defined, has been a decisive factor for choosing a model.

One of the first attempts to improve the comprehensibility of complex models was by Domingos [28, 29] in 1997. He states that the output of a Machine Learning model can only be considered useful knowledge if it is accurate, stable, as well as comprehensible. In short, accuracy is the ability of the model to make a correct prediction, stability means that small changes in the training set input do not result in large changes in the final predictions, and comprehensibility is the ability to explain the choices that a model made. These terms are explained more in depth later on in Sections 2.3.1, 2.3.4 and 2.1 respectively. To achieve this, he proposes to approximate a complex model by a much simpler model with comparable performance. Around the same time, Provost et al. wrote a paper

that criticizes the common practice of evaluating models on predictive accuracy only [94].

Despite their efforts, the research on comprehensible Machine Learning models lagged behind. It was not until recently that the field slowly started to regain interest.

2.1.2 Relevance

It is important to note that there are many situations in which using a black box model makes sense. For problems such as object recognition or natural language processing, it is less relevant to know which steps the algorithm has made to achieve its prediction, as long as the predictive accuracy is high.

However, there are also cases in which comprehensibility is very beneficial. For fraud detection, the only thing a model is be able to provide is a conjecture, as usually no rules exist that perfectly distinguish a fraudulent case from a non-fraudulent one. We refer to this as a low *quality* prediction. A professional will need to investigate the case to verify the suspicion by the model. In such a case the explanation how the model achieved its prediction could be just as valuable as the original prediction. Other examples include illness diagnosis [28] (e.g., for diabetes or pneumonia risk [62, 15]) or bankruptcy prediction [117].

2.1.3 Benefits

In general, when risk factors like financial liability or health risk are involved, or the quality of the prediction is inadequate or unknown, then there is a need for better explanations how the model made a prediction. This matches well with the benefits that comprehensibility can leverage [73, 61]:

1. **Data understanding and Causality** Model explanations can be used to help analysts to understand and observe the data they are working with by uncovering complex patterns or hidden interactions. For example, a simple regression model might reveal a strong association between thalidomide use and birth defects or smoking and lung cancer [73, 115].

2. **Trust and Accountability** Explanations can also ensure trust in a model that makes critical decisions about human beings [98]. For cases like fraud detection, illness diagnosis or bankruptcy prediction, the decisions have a direct associated risk. Especially if the predictability of the class is inadequate or unknown, we need explanations to verify the correct behavior of a model.

3. **Model diagnostics** For Machine Learning model development it is also very beneficial to understand why a model makes a wrong decision. By analyzing the model, the developer could extract the more salient features in the data and optimize the model for this. Additionally, by looking critically at the edge cases, wrong behavior can be fixed [58]. As Nguyen et al., Szegedy et al. [86, 109] show, we cannot always rely on the classification result that a model provides. There may exist *adversarial* instances: some slightly altered instance from the training set can lead the model to believe the instance is a completely different and often incorrect class. Without an explanation from the model, it is very challenging to verify such behavior.

4. **Fair and Ethical decision making** Consider a model that is used for predicting the probability of an individual relapsing in criminal behavior [17]. In such a scenario there is no way of knowing for sure whether the model discriminates based on race or gender. Explanations could give us the ability to evaluate the model and ensure ethical decision making.

   Recently a European regulation [97] has been passed which prevents a company from using Machine Learning insights to make decisions about customers directly without providing clear reasoning. This means that automatic fraud detection is not allowed; it is required that an employee considers each case and determines whether the classification was reasonable.

2.1.4 Definition

Many authors agree that interpretability is an ill-defined concept [59, 73, 113]. Lipton [73] states that the interpretability of various models exhibits a quasi-scientific character. It is not a monolithic concept, but in fact reflects several distinct ideas. In his work he presents a taxonomy of various perspectives on the concept.

Some papers explain interpretability as the ability to grasp *how the models work* [76]. Lipton [73] refers to this as *transparency*. Humans should be able to follow every calculation made by the model within "reasonable" time (*simulatability*). Every input, parameter, and calculation should admit an intuitive explanation (*decomposability*, e.g., the decision in a node of a decision tree can be displayed textually), and the algorithm itself should be clear (*algorithmic transparency*, e.g., we can understand the shape of the error surface). Other papers refer to interpretability as post-hoc: *what else can the model tell me?* This does not explain how the model works in its entirety but still gives intuition how certain functionalities are derived. Lipton [73] identifies three methods of post-hoc interpretability: natural language explanations, visualizations, and explanations by example.

In this report, we define model interpretability as

> the ability to explain the choices a model made that make up a single classification.

This definition has a lot of similarities with the *decomposability* concept as defined by Lipton [73].

2.2 Training data

2.2.1 Glossary of common terms

In Machine Learning and data science often multiple terms are used for the same concept. To aid the reader to understand common concepts, we present a glossary of common terms (adapted from [60]).

- **Feature** *(also: variable)* A quantity describing an instance. An attribute has a domain defined by the attribute type, which denotes the values that can be taken by an attribute. The following domain types are common:
  - **Categorical** A finite number of discrete values. The type *nominal categorical* denotes that there is no ordering between the values, such as last names and colors. The type *ordinal categorical* denotes that there is an ordering, such as in an attribute taking on the values low, medium, or high.
  - **Continuous** *(also: quantitative or numerical)* Commonly, subset of real numbers, where there is a measurable difference between the possible values. Integers are usually treated as continuous in practical problems.
  - **Target** *(also: dependent variable)* Special feature of a record providing the records ground truth. The model is trained to predict the target feature for instances where it is not known.
- **Class (also: class label)** The value of a target feature.
- **Record (also: item)** A list of features describing an instance.
- **Instance (also: case)** A single object of the world from which a model will be learned, or on which a model will be used (e.g., for prediction).

2.2.2 Formalization

Given all possible values, or *domain*, of class feature $\mathcal{Y}$, let $X = (x_1, x_2, ..., x_D)$ be a D-dimensional input and $y \in \mathcal{Y}$ be an output class label. Each instance record for a test or training data set with n records and features $F = (f_0, f_1, ..., f_D)$ can be described by composing X with the given ground truth y as: $\{(X_i, y_i) \mid i = 1..n\}$. For categorical feature f_i, the domain is described by D_{f_i}. We use this notation throughout the rest of this thesis.

2.3 Supervised Machine Learning Models

Supervised Machine Learning is the collective name for algorithms that generalize a mapping between *test* data input and a particular set of output classes. They derive this mapping by analyzing a set of *training* data with ground truth. The produced generalization extends beyond the training input to classify newly encountered records correctly. This is often referred to as inductive bias [82]. This mapping is referred to as a *predictive model* and classification is the process of predicting a class label for a given data instance.

2.3.1 Glossary of common terms

In Machine Learning and data science often multiple terms are used for the same concept. We present a glossary of common terms, adapted from Kohavi and Provost [60].

- **(Predictive) Accuracy** The rate of correct predictions made by the model over a data set (cf. coverage). Accuracy is usually estimated by using an independent test set that was not used during the learning process (*validation*).
- **Classification** The process of assigning a class label to a data record using a predictive model. In the case the set contains only two class labels, the process is referred to as *binary classification*.
- **Regression** The process of assigning a continuous value to a data record using a predictive model.
- **Precision** The fraction of returned instances by the model that belong to the target class.
- **Recall** The fraction of all instances belonging to the target class that are returned by the model.

2.3.2 Black box vs. White box

In literature, authors often distinguish between two types of models: *black box* and *white box*. This difference comes down to how a model is approached. Black box models are often complex, and their inner workings are difficult to understand. White box models, on the other hand, are simple enough that their functionality can be intuitively explained.

Even though the process of training a black box model is fully understood, the structure of the trained model gives us little information about the function being approximated. Complex models often boast high predictive accuracy, but this is achieved at the cost of comprehensibility of the model.

Examples of typical black box models include neural networks and ensemble models. As we mentioned in Section 2.1, these models are difficult to understand. Hence, authors approach these models by only considering the input and output (see Figure 2.1).

In contrast to black box models, transparent models exist that, when of reasonable size, can be easily explained to humans [113]. We refer to these models as *white box models*. Due to their simple structure these models often have a lower predictive accuracy compared to black box models. A good example would be logistic regression, which has a very trivial input-output relationship. There is only a fixed number of weights equal to the number of features that explain the feature's contribution to the output. The complexity of such a model does not change given different amounts of training data,

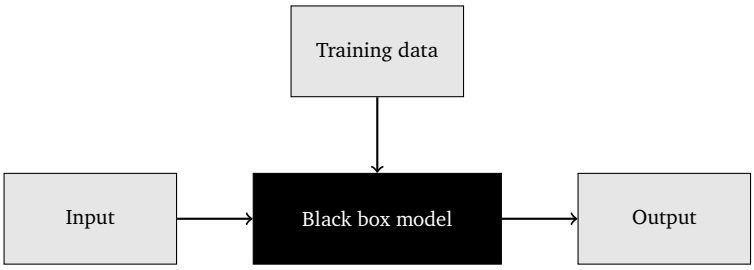


Figure 2.1: Black box model.

and thus remains interpretable. The downside is that a linear model is unable to capture complex behavior such as feature interactions.

Not all literature agrees upon the clear distinction of black box vs. white box models. As early as 1997, methods to interpret ANNs were proposed [6]. The author argues that, since a method for explaining the method exists, we cannot refer to ANNs as black boxes. This suggests it is more of a dark-grey box. Similarly, decision trees (which we discuss in Section 2.3.4) can be considered white-box as their structure is easy to interpret. However, if the tree gets sufficiently large, it becomes difficult to reason about [113]. In such a case, the term light-grey box is more appropriate. We should thus not consider the black- or white box nature of a model as a strict boundary, but more of a gradually changing property depending on the complexity and size of the model, nature of the training data and knowledge of the user.

2.3.3 Global vs. Local

Besides categorizing models on a black-white box scale, we can also approach them by considering their locality. A model can be approached either *globally* or *locally*.

A global viewpoint means we consider what the model does as a whole. What is the predictive accuracy of the model? What features are important? How complex is the model? Most literature focuses only on the global aspects of a model, but such a global approach can conceal relevant information about its behavior. If two types of input are processed in different ways, then a global insight will show the average of both ways and obscure the real behavior.

In contrast, a local viewpoint means we only look at a smaller portion of the model. For instance, we can consider only the decisions the diabetes model makes for patients of a particular age. Various decisions of the model are be applicable to this group, and can thus be disregarded.

Typically, we consider how the model behaves given one particular instance input: referred to as *instance-level*. How well was the model able to generalize a prediction for that instance? Which features were important for that particular classification? Only recently authors have started to focus more on the instance-level decision making process of models [110, 98, 63, 61, 62, 91].

2.3.4 Decision tree

Many different Machine Learning models exist. One of the most widely used [83, 119] and easy to understand (and thus white box) models [35] is the decision tree. It consists of *internal nodes* that contain a test based on a specific feature on the input data and directed *edges* that dictate the order in which these tests are executed. *Leaf nodes* contain the final prediction class.

A decision tree can be naturally visualized using a node-link diagram. An example is shown in Figure 2.2. The tree is concerned with the decision whether to play golf or not, indicated by the class labels *Play* and *Don't Play*. The decisions are based on the weather features *Outlook* (factor), *Humidity* (numerical) and *Wind* (boolean).

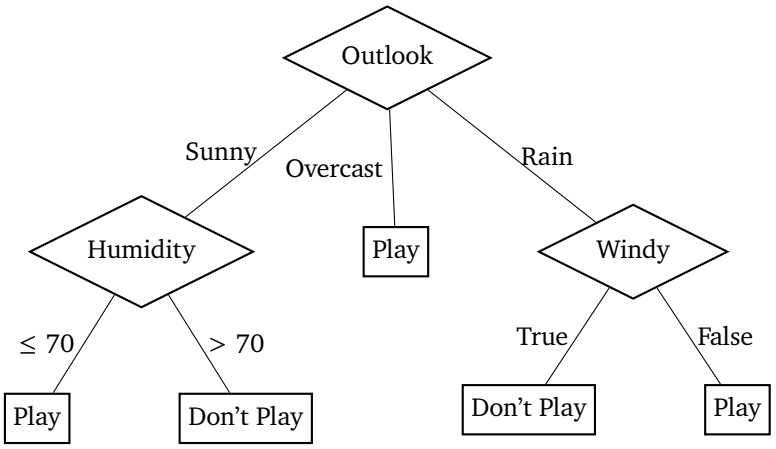


Figure 2.2: Classical example of a decision tree for the 'Play Golf' problem.

Classification of a single instance happens by following the path from the root down the tree by choosing the direction based on the test in the node. The first record in Table 2.1 is classified by following the left branch for *Outlook* = "*Sunny*", then the right branch for *Humidity* > *70*, by which we end up in a leaf node with class label *Don't Play*. Other features of the instance are ignored. Likewise, the fourth record applies the tests *Outlook* = "*Rain*" and *Windy* = *false* to return class label *Play*.

Table 2.1: Training data set for the "Play Golf" problem.

Outlook	**Temperature**	**Humidity**	**Windy**	**Play Golf?**
Sunny	85	85	False	Don't Play
Sunny	80	90	True	Don't Play
Overcast	83	78	False	Play
Rain	70	96	False	Play
Rain	68	80	False	Play
Rain	65	70	True	Don't Play
Overcast	64	65	True	Play
Sunny	72	95	False	Don't Play
Sunny	69	70	False	Play
Rain	75	80	False	Play
Sunny	75	70	True	Play
Overcast	72	90	True	Play
Overcast	81	75	False	Play
Rain	71	80	True	Don't Play

Training the model is a top-down process where the training data set is recursively partitioned into ever smaller subsets. These subsets are determined by choosing a feature constraint at each step which is best able to split the set of records according to a certain metric, called the *splitting criterion*. These metrics measure the homogeneity of the target feature within the training data. The data set is split based on various feature split points. Next, the metric returns a score for each subset. The feature split point which yields a subset with the highest score will become the definite split point. Finally, the algorithm recurses over both resulting child subsets and starts the process over.

The most ubiquitous splitting criteria used for training decision trees are listed below.

- **Information gain** This metric is based on the concept of *information entropy* [101]. It measures the disorder or uncertainty in a system. Shannon [101] states we can gain more information from low-probability events relative to high-probability events. Formally, entropy H is described by

$$H(S) = -\sum_{i=1}^{n} p_i \log_2(p_i) \tag{2.1}$$

  where n is the number of classes, and p_i refers to the probability of the event given class $\mathcal{Y}_i$. For decision tree classification, p_i is the fraction of records that have class $\mathcal{Y}_i$ within the subset S. There is a clear negative correlation between the probability and entropy.

  Information gain is defined as the difference between the entropy before the split and after the split on feature f. We do this by calculating the entropy of the parent, and next subtracting the weighted average of the entropy of all children. Formally, information gain is described as

$$G(S, f) = H(S) - H(S \mid f) \tag{2.2}$$

  As an example, information gain is computed for the first split in the decision tree in Figure 2.2 on the feature *Outlook* in Appendix B.

- **Gini impurity** is very similar to information gain. It measures how often an arbitrary record from the training data set would be classified incorrectly if it were randomly labeled according to the distribution of labels in the subset. Gini impurity I_{Gini} is defined as

$$\begin{aligned} I_{Gini}(S) &= \sum_{i=1}^{n} p_i(1 - p_i) \\ &= \sum_{i=1}^{n} (p_i - {p_i}^2) \\ &= \sum_{i=1}^{n} p_i - \sum_{i=1}^{n} {p_i}^2 \\ &= 1 - \sum_{i=1}^{n} {p_i}^2 \end{aligned} \tag{2.3}$$

  where p_i is again the fraction of records that have class i within the subset S.

The most well-known implementations of decision trees are the following:

- **ID3** or Iterative Dichotomizer was the first decision tree implementations for classification, developed by Quinlan [95]. It uses information gain to determine splits.
- **C4.5** is the successor to ID3. C4.5 [96] supports both categorical and continuous features, it handles incomplete data points and includes a pruning technique that prevents overfitting.
- **CART**, or Classification And Regression Trees is very similar to C4.5, as it supports all the improvements over ID3. It differs as it uses Gini impurity rather than information gain to determine splits. In addition, it always constructs binary trees, whereas C4.5 allows for more than two children per node [120].

Decision trees do have a number of limitations. One of the biggest problems is that they easily *overfit* the training data. As the model gets more complex, the error on the training data seems to decrease further and further. Yet, when the model is validated on a separate test set, a disparity between error rates emerges. An example is shown in Figure 2.3 in which after polynomial degree three, the error stops its decline for the test set. After this point, the model is trained too specific for the training data, and does not generalize well to different data sets.

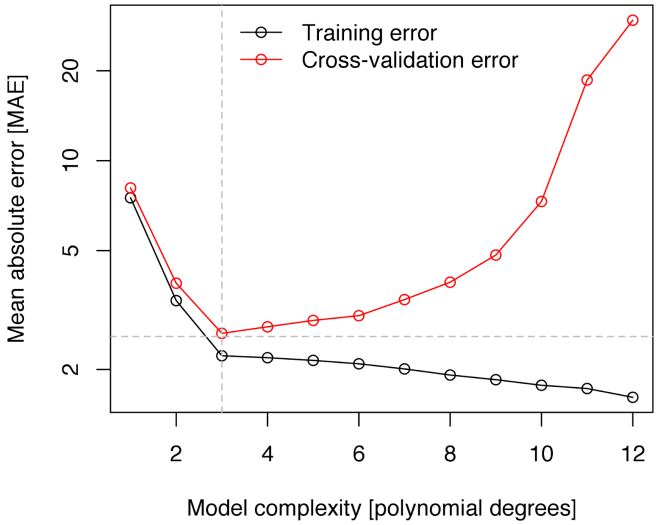


Figure 2.3: Example of test and training data disparity over training iterations when overfitting. Taken from: https://www.r-bloggers.com/evaluating-model-performance-a-practical-example-of-the-effects-of-overfitting-and-data-size-on-prediction/

Decision trees are also inherently unstable [69, 53]. This means that small changes in the training set input can result in large changes in the final predictions. In addition the construction algorithm is *greedy*, as it makes locally optimal decisions per node. This implies that it is very unlikely to return a globally optimal solution. This is not unexpected, as the construction of optimal decision trees has been shown to be NP-complete, by Hyafil and Rivest [51, 85] as early as 1976.

Formally we describe a decision tree model T as a binary tree (CART). A leaf node $N_{leaf} = (y)$ contains only a class label $y \in \mathcal{Y}$. Each internal node $N = (N_L, N_R, C)$ has a left child N_L, right child N_R and associated constraint C. For continuous features, C is of the form $C = (f_i \leq s) \vee (s_1 \geq f_i \geq s_2) \vee (f_i > s)$ where s constitutes the split point for the node. For categorical features, $C = (f_i \in Q)$, where $Q \subset D_{f_i}$. For the special case where $|Q| = 1$, the constraint is expressed as $C = (f_i = Q)$. We can define a subset S_C to contain records which all conform to constraint C as $\{(X_i, y_i) \mid i = 1..n, \forall_i(C(X_i))\}$. N_{root} is a special case denoting the root of the tree.

2.3.5 Decision rule

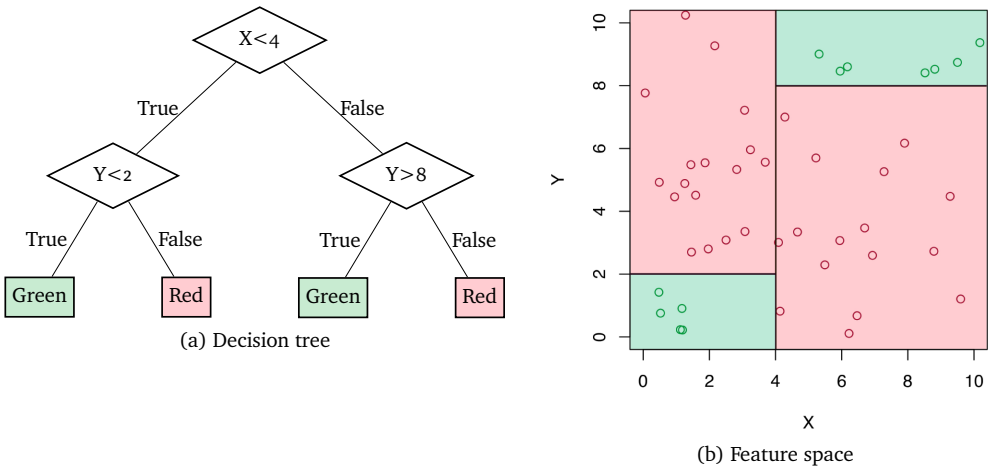
With the introduction of C4.5, Quinlan [95] also introduced the c4.5rules tool, which, instead of representing the model as a tree, outputs a set of *decision rules*. A decision rule R is a composition of one or more constraints C that correspond to a path from a root node N_{root} down to a leaf node N_{leaf}.

An example of decision rules for the "Play Golf" decision tree in Figure 2.2 would look like

$$\begin{aligned} Outlook = \text{"}Sunny\text{"} \ \wedge \ Humidity > 70 &\Rightarrow Don't\ Play \\ Outlook = \text{"}Rain\text{"} \ \wedge \ Windy &\Rightarrow Don't\ Play \\ \\ Outlook = \text{"}Sunny\text{"} \ \wedge \ Humidity \leq 70 &\Rightarrow Play \\ Outlook = \text{"}Rain\text{"} \ \wedge \ \neg Windy &\Rightarrow Play \\ Outlook = \text{"}Overcast\text{"} &\Rightarrow Play \end{aligned} \tag{2.4}$$

Decision trees divide the feature space in partitions. This is illustrated in Figure 2.4b. Each of these divider lines correspond to a decision made in the node of a tree. For instance, the middle vertical line at $X = 4$ corresponds to the decision in the root of the tree.

The separate areas created by the partitions represent decision rules. For instance, the green area in the bottom left corner can be described as the rule $(X < 4) \wedge (Y < 2) => Green$.



(a) Decision tree

(b) Feature space

Figure 2.4: Areas created by decisions in a decision tree. Each area corresponds to an extracted decision rule.

A set of decision rules can be used for classification by checking the applicability of the left hand side of every rule. The rules that apply will give a prediction, which is the final prediction by the rule set. In case the rules are extracted from a decision tree, only one of the rules will match. This corresponds with the notion that none of the feature space areas divided by the tree overlap, like in Figure 2.4b. However, when decision rules are extracted from multiple trees (like from a Random Forest introduced in Section 2.3.6), it is possible for areas to overlap. In such a case multiple rules can be applicable to an instance, and the prediction is determined by voting amongst all applicable rules.

2.3.6 Ensemble models

In order to further improve the predictive accuracy of classifiers, new techniques have been proposed to combine the result of a set of individually trained models. By far the most popular approaches to combine models into an ensemble are Bagging [10] and Boosting [37, 99].

Bagging *(short for: Bootstrap Aggregating)* Bagging works by uniformly sampling N' records from the training data set *with replacement*, where N is the size of the original training data set, and $N' < N$. This is often referred to as a *bootstrap* sample [30]. For every bootstrap sample a new classifier can be learned, which has its own unique training set. The final classification is determined by majority vote among the multitude of classifiers.

Boosting Instead of individual classifiers, Boosting tries to create a *series* of classifiers. The training set used for a model in the series is based on the performance of earlier classifiers. When a record of the test data is incorrectly classified by a model, the likelihood of it being used as training data for the next models increases. Thus, Boosting attempts to create new classifiers that produce higher accuracy for the records that earlier models performed poor upon.

Both methods have been shown to outperform single classifiers [2, 5, 27, 21], although the accuracy of Boosting seems a lot less consistent [78, 88, 26]. The authors show that the accuracy of Boosting can be much higher than a single classifier. However, for a few data sets in their experiment the accuracy was equal or even slightly worse.

It has been shown theoretically [65, 42] and empirically [89, 90, 45, 88] that the best performance of an ensemble can be achieved when the classifiers disagree as much as possible for edge cases. This way there is always a model that can cover for the mistakes of others. The authors also show that this method is most effective for unstable classifiers. This implies that the models that comprise the ensemble will all be different, which naturally implies that the models will disagree.

Random Forest

Tree ensembles are ensembles comprised of decision trees. They were first devised by Ho [48] in 1995. One of the most powerful tree ensembles to date are Boosted Trees by Friedman [38, 26]. Yet, tree ensembles gained most popularity by the introduction of the widely popular Tree Bagging method by Breiman [11] called Random Forests. Along with co-author Cutler he even trademarked the name in 2006 [12]. As decision trees are inherently unstable [69], they are a good fit for ensemble modeling.

In addition to Bagging, Random Forests also apply a method referred to as *random subspace projection* [49]. This process selects a random subset of features at each candidate split. This prevents overfitting on features that are very strong predictors for the target class.

An example of a Random Forest trained on the Play Golf training data set is shown in Figure 2.5. For the seventh record in the training data in Table 2.1 the outputs of the individual trees differ. The first two trees classify the instance as *Play*, whereas the other tree predicts *Don't Play*. The output of the Random Forest as a whole is *Play* (by majority vote) with a classification score of 0.667. The different sets of features that appear in each tree indicate that random subspace projection has been applied. As the decision trees are of the CART variety, they are all binary trees.

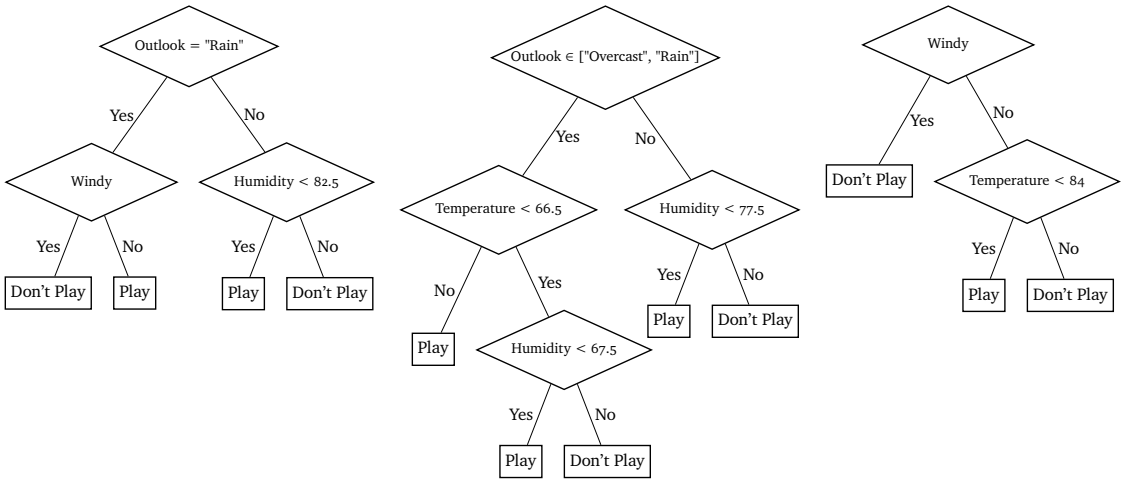


Figure 2.5: Random Forest model trained on the 'Play Golf' data set in Table 2.1.

The training procedure for Random Forests naturally allows for some useful metrics to be computed along the training process without much added complexity or need for an additional validation data set:

OOB error Out-of-bag error is the mean prediction error on each training sample X_i using only the trees that did not have X_i in their bootstrap sample [53]. Most Random Forest implementations collect this statistic as part of the training process. It has been shown that the OOB error can slightly underestimates the true accuracy of a model [81]. This bias very slight, and only becomes problematic when the number of features exceeds the number of records. For our purposes, it remains a useful tool for model validation.

Feature importance Similar to OOB error, Random Forests can also be used to rank features according to their relevance in classification. This is often referred to as *feature importance*. A basic implementation of this technique was introduced in the original Random Forest paper by Breiman [11].

To measure the importance of feature f, the values of that feature can be randomly permuted to render it meaningless. When the OOB error is recomputed, we can measure the difference in OOB error as a result of the permutations. The average of all differences for feature f, normalized by the standard deviation, forms the final importance ranking [11]. It can be seen as the decrease in accuracy of the model when the feature would not have been used. This gives a global indication of the importance of a feature. It has been shown that this method has a slight bias towards categorical features with a high number of class labels [24, 107].

To this date, the predictive accuracy of Random Forests is still competitive [74]. They are able to cope with overfitting due to the power of averaging [121, 4]. However, as the ensembles often comprise hundreds of trees, it can be very difficult to interpret the model directly. This complexity is not strictly required, as many authors have shown that simplification through reducing the number of decisions can lead to similar or even better results [29, 28, 14, 105, 106, 2, 41, 32, 122, 4].

Chapter 3

Problem analysis

This project is carried out in collaboration with Achmea BV. Achmea is one of the leading providers of insurances in the Netherlands. It is the result of a merger that took place on January 1, 1995 between Zilveren Kruis and Avéro Centraal Beheer Groep (AVCB). Nowadays, Achmea BV is the governing body of the majority of insurance brands in the Netherlands, including Centraal Beheer, FBTO, Interpolis and Zilveren Kruis. In this project, we focused on the following case.

Companies are obliged by the Dutch Civil Code to continue paying 70% of the employees salary during the first 104 weeks of illness. For smaller companies with unexpected illness amongst employees, this can be difficult to cover for. This is why Achmea provides an insurance specifically for this situation: *sick leave insurance*. This insurance is exclusively targeted at employers. With this insurance they are able to cover the costs of prolonged absence of employees as a result of an accident or illness.

A problem Achmea is dealing with is fraud. For the aforementioned insurance, it sometimes happens that employers tamper with the facts and provide false information in order to get coverage for uninsured employees, or receive compensation while no employees were actually sick. Achmea even mentions the existence of malicious groups that create companies with the sole purpose of exploiting this service to their advantage.

In order to support the fraud detection process, a predictive model was created to track down fraudulent cases. While this model has been proven successful, it still requires investigation of every predicted high-risk case to determine why it is considered fraud. The solution presented in this thesis could further improve the fraud detection process by exposing a reasoning for fraud classification by the model.

There are three phases in the fraud detection process, as illustrated in Figure 3.1. The first phase refers to the process before the model is introduced. The second phase, which is the current situation, refers to when the predictive model is used strictly for ranking insurance policies. The third phase refers to the aim of this project, where the model is not only used for a ranking but also provides an explanation for individual classification results.

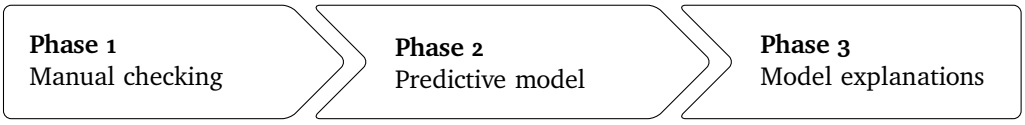


Figure 3.1: Three phases in the fraud detection process.

We continue by defining the Data Flow and Process Model for phase one. Next, we introduce phase two in Section 3.4 and show the impact on the process. We conclude this chapter by introducing the requirements necessary to move to phase three by defining questions, user tasks and requirements.

3.1 Stakeholders

The main stakeholder for the project is the fraud team; they will eventually use and benefit from the proposed solution. It is important to take the knowledge of the stakeholder into account [118]. The fraud team has a lot of domain knowledge about insurance, fraud, the systems they work with, and all aspects of the data within these systems. However, their knowledge of the predictive model is limited at best. We ensure that the questions require no domain knowledge to be understood or answered.

3.2 Data flow

A team of fraud professionals is permanently assigned to investigate possible fraud cases for sick leave insurance. We refer to this team as the fraud team. The data flow of their investigation process is modeled in Figure 3.2.

They mainly get leads from their colleagues at the sick leave insurance department. These colleagues are the primary contact between Achmea and the client. Every different type of department (Claim, Care, First-line and Medical service) has a checklist of around ten questions that are indicative of fraudulent cases. This includes questions like "Does the employee have a high salary relative to their activities within the company, or age?" and "Is the client difficult to reach by telephone?". The more questions are answered positively, the more suspicious the client. All customer contacts are logged in an Oracle system called Siebel CRM.

When operators are suspicious of an employer, they forward the case to the fraud team. They use various sources of information to investigate the case. This takes them on average about an hour per case. As soon as they too find the case relevant (i.e., a significant amount of money is insured) and suspicious enough, a larger investigation starts. Such an investigation can take a lot longer, sometimes even months of lead time. This is because enough evidence needs to be collected in order to lawfully proof the employer knowingly committed fraud.

When fraud is proven it is added to the FACTS fraud database system. From there, the case propagates to various systems, amongst which the external registration system called EVR. This system is used to share information about high-risk fraud cases between insurers in the Netherlands.

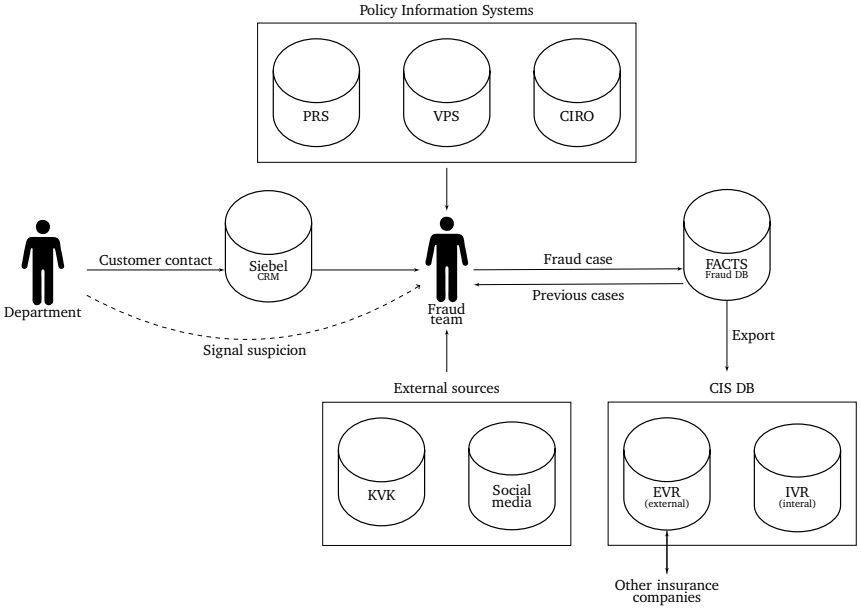


Figure 3.2: Data flow model for the Achmea fraud team. Cylinders represent different databases.

3.3 Process model

The process of investigating a fraud case varies: it differs per member of the team and per case being investigated. We capture the generalized behavior in the process model in Figure 3.3.

The fraud team first checks if the case is relevant. A case is deemed irrelevant when for instance the amount of insured salary is lower than the cost for Achmea to investigate the case, but also if the insurance policy is older than three years. Even if the team can lawfully proof fraud, after three years the limitation period expires and the client will not be liable anymore. It sometimes also happens that a case is reported where the policy has already been proven fraudulent, or which is already under investigation. In phase two, the prediction by introduced model forms another indication of relevance. If a case is irrelevant, the fraud team abandons the case to investigate the next one.

After the relevance check, a loop is entered in which all features are checked for suspicious values. This is where the model deviates from the real process. During operation, the fraud team does not systematically check all features of the data, but selects a few features that they consider relevant for that case. Which features they check varies widely per case and per team member. The process model reflects this by not being specific about which features are checked, but includes a choice whether there are features left to check instead.

The action to investigate a feature is left unspecific on purpose. This action includes checking the feature value and comparing it with either the fraud teams knowledge of what is considered fraud, but also to compare it with other cases. Additionally, this action includes comparing the feature value with other features in the data. The investigation process varies too much to be able to model it more precisely.

After each feature investigation, a decision is made on whether the entire case can be considered suspicious. If this is the case, the fraud team can impose prosecution, otherwise, more features are checked. If there are no features left to check (i.e., the team is sufficiently sure the case is not fraudulent) the process is ended.

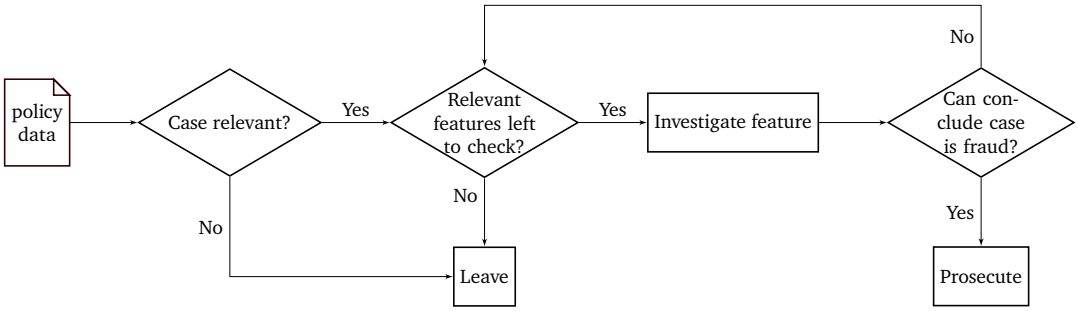


Figure 3.3: Process model for investigating a single insurance policy for fraud.

3.4 Automated fraud detection

The Business Intelligence (BI) Analytics team from Achmea created a predictive model to aid the fraud team with identifying fraudulent cases. This correlates with the second phase in Figure 3.1. The model is able to predict whether a case could be fraudulent based on information from three different systems the fraud teams use.

The model improves the fraud detection process by supporting the *Case relevant?* and *Can conclude case is fraud?* decision in the process model of Figure 3.3. For every case a score is available on the certainty of the model of the instance being fraud. This score can help the fraud team to determine the case relevance quicker. If the list of insurance policies is ranked based on this score, they can consider the top ranked cases to investigate. When a case is investigated, they can also use the score as a guideline to decide more quickly if a case is suspicious or not, corresponding to the *Can conclude case is fraud?* decision in the process model.

Even though the system has only recently been deployed, a number of fraudulent cases have already been uncovered by the system. Currently the analytics team provides a list of hundred highly suspicious instances according to the model upon request. The fraud team will manually investigate every case.

3.4.1 Data set

The data set used to train the predictive model combines data from the VPS, PRS and Siebel CRM systems, as well as some manually sourced external data from the KVK. It has been created by the analytics team specifically for this purpose. It contains 40,000 records that correspond to a sick leave insurance policy of a company. This means that no information is available on the specific employee that is absent. The large portion of insurance policies (30%) only have one sick employee listed. For the policies with multiple employees, information specific to the employee is aggregated. For instance, the duration of the illness is captured in two features: the average (*Feature 25*) and total duration (*Feature 24*) of all employees.

The data set contains 49 different features, of which 8 are categorical and 41 continuous. The class feature *Feature 1* is an indication whether the insurance policy was found to be fraudulent after extensive investigation. This feature is used to train the categorization to be made by the model. Of all records, only 129 records are proven to be fraud. A complete overview of the features and their meaning can be found in Appendix A.

Just as we defined in Section 1.2, this case is a binary classification problem, which means that insurance policies can be either fraud or not. This is indicated by the class labels *Wel fraude* for fraud, and *Geen fraude* for non-fraud.

To analyze the data we projected the data set into two dimensions using T-Distributed Stochastic Neighbor Embedding (t-SNE) [114, 77], as shown in Figure 3.4. It is clear that there are no apparent clusters in the fraud cases (red), and they seem relatively spread out throughout the non-fraud cases (black). This indicates that the problem of detecting these fraud cases is very challenging.

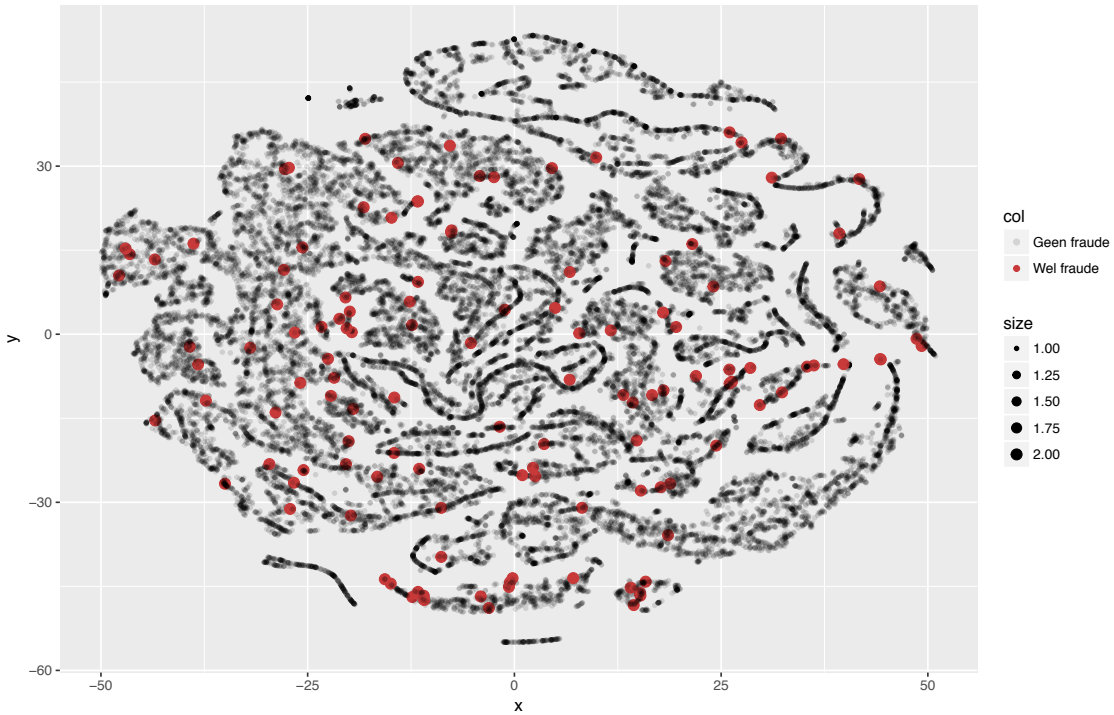


Figure 3.4: t-SNE projection of the Achmea data set.

Contrary to most benchmark data sets commonly available in the Machine Learning field, real data sets often contain many issues. It is important to be aware of these issues, as they might have a significant impact on the final visualization. We mention a few unfavorable characteristics of the data that have to be accounted for:

Imbalance The data set is heavily imbalanced. As we mentioned, of the 40,000 records, only 129 are proven cases of fraud: a mere 0.3%. This imbalance exists partially due to the low occurrence of fraud in general (5% as estimated by Achmea), but also because of the intensive manual labor required to collect enough lawful evidence to prove a fraudulent case. The analytics team is currently investigating using suspicion of fraud as a fraud indicator, but this requires large changes to the model and is outside the scope of this project.

Most predictive models are not able to deal with this imbalance. In order to cope with imbalance, a subset is required which is sufficiently balanced for the model to learn from. Common approaches are to sample the scarce class instances more than once (*Oversampling*) or pick a random subsample from the non-scarce class instances (*Undersampling*). With such a small number of fraud cases, only undersampling is feasible. This greatly limits the amount of data we can use for training of a single model.

Missing values The data contains many records with missing feature values. This is because the data is added and updated by many peers in the organization and the manual nature of data collection. As the amount of data on fraud cases is low, it is not feasible to remove the records with missing values, as too few records would remain.

The analytics team decided to manually *impute* all values, which means they substituted real values for the missing ones. This could lead to a trained model that is not able to distinguish between missing values and values that are close to the imputed value. Especially for continuous features this can be problematic. Consider the situation where the fact that a value is missing itself is a good predictor, and we impute with the value zero, then the model might learn that low values of a feature are a good predictor instead.

Unclear semantics There is also a semantic issue concerning class distinction. The cases that are considered fraud are proven to be fraudulent, but the remainder of records are not necessarily non-fraudulent, just not proven to be fraud. There is still a possibility that the remainder of records contains a fraudulent policy, even though the chance of a non-fraud case being fraudulent is relatively low.

Missing test data A final challenge is that all data made available was used as training data for the model. This means that no test data is available for verifying the behavior of the model, or any other simplified models we create. The only metric that is available is the OOB error, which we described earlier in Section 2.3.6. However, this metric only shows the error of the Random Forest and cannot be used for simplified models.

3.4.2 Fraud detection model

The model that Achmea provided is an ensemble model. More specifically it is a Bagging ensemble of 100 different Random Forest models. Each of the individual forests are comprised of 500 different CART decision trees.

Achmea chose to combine multiple Random Forest models in order to cope with the heavily imbalanced data set. Each Random Forest is trained on a subset with the same 100 fraud records and 100 randomly sampled non-fraud records (*undersampling*). Contrary to a single Random Forest (which was not able to generalize the fraud detection problem at all) the 100 models combined achieve a reasonable error rate of 27.7%. Next, we evaluate the Random Forest using some readily available methods.

In Figure 3.5 the OOB error rate is plotted against the number of trees. After 200 trees the performance of the model hardly increases. The analytics team decided to go with 500 trees in their model to be sure, even though the additional 200 trees hardly provide any benefit to the classification.

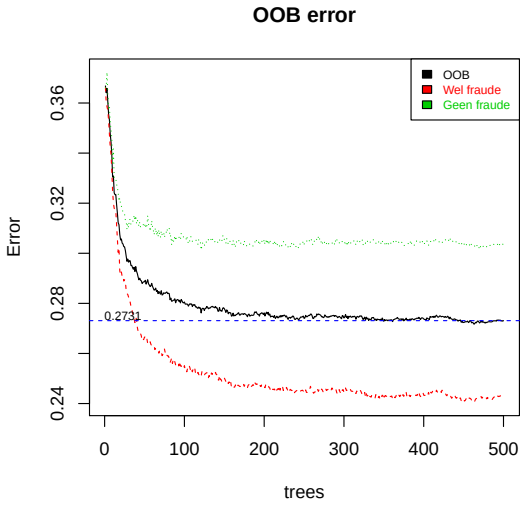


Figure 3.5: The OOB error rate plotted against number of trees for the Achmea Random Forest ensemble model.

3.5 Questions

The recently introduced predictive model is able to help with determining case relevance. However, a large portion of the process model in Figure 3.3 remains unchanged. After a case is found relevant, every feature still has to be checked for suspicion of fraud. The loop in the model persists, and so the time it takes the fraud team to investigate the case itself is unchanged. In this thesis we propose a method that could improve the process further, by giving insight in the reasoning of the model (phase three in Figure 3.1). This impacts the *Relevant features left to check?* and *Investigate feature* choices in the process model in Figure 3.3.

To aid the fraud team with their work, the Achmea analytics team posed the following main question:

> How can we analyze the choices the model makes for fraud detection of a specific case, and display them in a comprehensible manner?

Note that the main question of this thesis and the main question by Achmea differ slightly. For Achmea the sole purpose is to obtain explanations about fraud detection, whereas this thesis focuses on a method applicable to any Random Forest, trained on any data set with binary class label.

In discussion with the fraud team we noticed that they had an additional goal for the project. Along reasoning of why a model is fraud, they also asked:

> Can we extract previously unknown patterns in the data from the model which indicate fraudulent behavior?

Knowing such a pattern can aid them in detecting fraud more effectively, even without using the model.

We can make these questions more concrete by asking the following sub-questions. Answering these sub-questions should be sufficient for answering the main questions.

Global sub-questions about the data and predictive model:

- How accurate is the model in predicting fraud?
- Which case is most important to investigate?
- What previously unknown patterns in the data exist that indicate fraud?

Sub-questions about a specific insurance policy:

- How certain is the model that a classification is correct?
- Which features are most relevant for a case prediction?
- What are the decisions the model made to end up at a certain classification?

3.6 User Tasks

We can translate these questions into user tasks that our visualization should accommodate. Following the multi-level typology of abstract visualization tasks by Brehmer and Munzner [9], we define user tasks that facilitate model interpretation. The user establishes context by *consuming* information, finds elements of interest by *searching* and finally obtains more information about a target instance by *querying*.

Consume

T1.1 *Present* accuracy of the model globally.

Search

T2.1 *Lookup* relevance of case.

T2.2 *Lookup* certainty of the model prediction for that case.

Query

T3.1 *Summarize* feature values.

T3.2 *Identify* relevant features.*

T3.3 *Identify* combinations of features that play a role (feature interaction).*

T3.4 *Identify* relationship between feature values and model outcome.*

T3.5 *Compare* with different cases.

The tasks marked with asterisks (*) can be interpreted in two different ways. We can either apply these tasks globally: looking at the behavior of the entire model, but also locally: considering the classification of a single instance (*instance-level*). For example, we can identify relevant features for the entire model, but also which features played an important role for a single classification result. Both insights can be informative, but for the use case that Achmea provided, local visualization is most relevant.

The user tasks listed above correspond to the different steps in the process described in Figure 3.3. Checking whether a case is relevant is supported by the task of looking up the prediction certainty of the model. Determining which features should be checked is supported by the summarizing features task and identifying relevant features task. In addition, identifying combinations of features has a link with the second goal described in Section 3.5.

3.7 Requirements

In addition to the requirement of realizing the user tasks mentioned in Section 3.6, we need to satisfy some operational requirements. The Achmea analytics team mostly works with the statistical computing language R [111] to create their predictive models. Furthermore, the package they use for Random Forests is the default *randomForest* package [70]. This package contains the original code written by the author of the Random Forest paper [11]. Achmea does not want to impose any constraint on which language the project is implemented in, however, the solution has to work with Random Forest models created in R. After some investigation it became apparent that no good methods currently exist to export a Random Forest model from R to another language. So, in order to prevent overhead caused by converting models from one to the other language, we decided to work with R as well.

Achmea also explicitly mentioned there is no strict time constraint on the running time of the visualization algorithms. As Achmea is currently transitioning towards using a predictive model (phase two in Figure 3.1) they have yet to figure out how to integrate the model into operation. Even though (given the size of the model) it could take considerable time to compute the metrics, the process is flexible enough to cope with large computation time. In addition, Achmea owns a hefty 32-core workstation which should be able to speed up any process significantly compared to the machine on which we will develop (Macbook Pro, Mid 2012).

3.8 Conclusion

In short, the main research problem of the thesis is: "How can we analyze the choices the model makes for fraud detection of a specific case, and display them in a comprehensible manner?". We defined atomic sub-questions in order to solve this problem, which can in turn be translated into a set of user tasks.

The data for a few of these tasks is trivial to obtain. For Task **T1.1** an existing metric for the performance of the model can be presented, like for instance the OOB-error rate. Task **T2.1** can be facilitated by combining the risk score for fraud predicted by the model with other heuristics like the feature *Feature 33* (total insured salary). Task **T2.2** requires the classification score for the instance, returned by the model. Finally Task **T3.2** only requires us to show the feature values of a particular instance.

The other user tasks (Tasks **T3.2**, **T3.3** and **T3.4**) require deeper algorithmic understanding of the model. Specifically, we need:

- A ranking method for features (Task **T3.2**).
- A method for finding relevant combinations of features for classification (Task **T3.3**).
- A method to extract relationship between feature and model outcome (Task **T3.4**).

We investigate possible techniques to facilitate these user tasks in the next chapter.

Chapter 4

Related work

Now that the problem of this thesis is clearly defined, we consider the solutions that have already been proposed. As mentioned in Section 2.1, there has only been limited focus on the comprehensibility aspect of Machine Learning research. Until recently, most algorithms were verified only by looking at their predictive accuracy. This means that not much work is available on this topic, and the work that has been done is often very recent.

In our literature study, we found a clear distinction between two types of approaches. Authors either analyze the features in context of the model, or worked on creating a simpler model that behaves and performs like the actual model. We found a few works that specifically focus on visualizing the structural aspect of the model. These techniques give a good insight in the structure of trees in the forest, but tell us nothing about the decision making process.

Feature analysis visualization mostly consists of *sensitivity analysis* methods, where the partial influence of variation of the value of a feature on the model outcome is shown directly. An additional category we identify are *feature importance* metrics, where the features of the data can be ranked according to their contribution to the model output.

Model simplification can also be subdivided into two categories: *meta-learning*, where a new model is trained based on the reference model, and *model condensing*, where parts of the existing model with low contribution to the output are removed. Both methods yield a simpler model that is much more comprehensible, while retaining most of the accuracy of the reference model. Of model condensing, we outline a specific category *rule extraction* that seems to be especially apparent in Random Forest literature.

A visual overview of the taxonomy is shown in Figure 4.1.

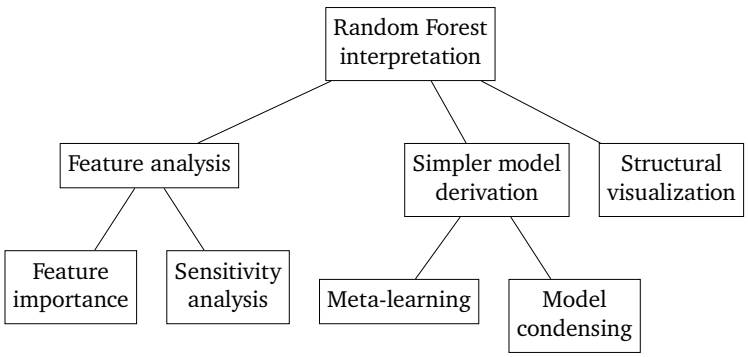


Figure 4.1: The taxonomy of interpretation techniques that are applicable to Random Forest models.

4.1 Feature analysis

Feature analysis techniques work by showing which features of the data are relevant for the model outcome. By looking at which features are important, we can derive that the model made decisions based on this feature, explaining the decision making process.

4.1.1 Feature importance

To effectively compare and rank features, *feature importance* metrics have been developed. These metrics output a single score for a feature based on their contribution to the outcome.

This can be done globally or locally. In the original implementation of Random Forests by Breiman [11], a global feature importance metric was already included (as described in Section 2.3.6). It gives an intuition on the importance of the features for classification, and is typically represented as a ranking of features. An example of such a ranking is shown in Table 6.2.

This ranking might be too generalized and simplified for some complex models [107]. If the decision for a feature f_1 always is strictly subsequent to a choice $f_2 \leq s$, then the choice for f_1 is always operated on the same subset of the data. The choice may thus not be relevant for the data where $f_2 > s$, even though the global ranking may state it is. The global ranking can thus be misleading, as the importance of a feature can differ per instance.

Local feature importance can thus be a lot more informative. Recently there have been efforts to create such local feature importance metrics [91, 67]. The work by Palczewska et al. [91] is most complete, as it works for both categorical and continuous features. It is a white-box approach as it utilizes the structure of the model in order to derive the feature importance. It is based on the concept of *local increments* LI_f^c for a feature f between a parent node p and child node c:

$$LI_f^c = \begin{cases} Y_{mean}^c - Y_{mean}^p, & \text{Parent splits on feature } f. \\ 0 & \text{Otherwise.} \end{cases} \tag{4.1}$$

where Y_{mean}^N is the probability that a randomly selected element from the training data subset in node N belongs to the target class. It thus represents the relative difference of the ratio of target class and other instances between parent and node. Note that this metric is insensitive to the amount of data split, an issue that is discussed in more detail in Section 9.3. The feature contribution $FC_{i,t}^f$ for an instance i is first calculated for every tree t in the forest as

$$FC_{i,t}^f = \sum_{N \in R_{i,t}} LI_f^N \tag{4.2}$$

where $R_{i,t}$ is the composition of all nodes on the path of instance i from the root node to the leaf node in tree t. This is essentially the same as a decision rule. The feature contribution FC_i^f for the entire Random Forest that Palczewska et al. [91] introduce is calculated as

$$FC_i^f = \frac{1}{T} \sum_{t=1}^{T} FC_{i,t}^f \tag{4.3}$$

where T is the number of trees in the forest. To evaluate their work, the author shows that if all the local feature importance metrics are combined to form a global insight, the resulting ranking is very similar result to a ranking obtained with the global feature importance metric as proposed by Breiman [11].

An example for the Play Golf data set is shown in Table 4.1. Consider the paths down the forest in Figure 2.5 that correspond to the classification of the first record in Table 2.1. Features *Outlook* and *Windy* are not used for any choice in the paths, so the importance of these features is 0. Two choices are made for the feature *Humidity* with small local increments, and one more important choice for the feature *Temperature* with large local increment. As the first record is classified as *Don't Play*, but our target class is *Play*, the contribution scores are negative.

Feature	Feature contribution (Don't Play / Play)
Temperature	−0.750
Humidity	−0.408
Outlook	0.000
Windy	0.000

Table 4.1: Feature contribution [91] for the first instance in the Play Golf data set.

4.1.2 Sensitivity analysis

Another method to analyze features is through *sensitivity analysis* [19]. This approach analyzes how the output of the model changes when the value of a feature of interest is varied. It is a typical black box approach, as only the input and output of the model are considered.

The typical way of visualizing this sensitivity information in the context of a Machine Learning model is to use a *partial dependence plot* (PDP) [38]. For a local understanding on a feature f_a of instance i, sample n uniformly distributed points along the range of this feature. Next, create n records with values of instance i of all features f with $f \in F$, $f \neq f_a$, and use the uniformly sampled values for feature f_a. Finally, obtain a prediction score for all n created records, and plot this score against the values of f_a. The resulting curve shows how the prediction score changes when feature f_a in instance i is varied.

For a global understanding the local sensitivity analysis results are obtained for all training data instances, and for each of the n samples, the mean of all prediction scores is plotted instead.

An example of a PDP for the 'Play Golf' Random Forest introduced in Section 2.3.6 is shown in Figure 4.3a. This plot focuses on the *Humidity* feature only. The x-axis corresponds with the feature range within the training set, the y-axis represents a metric on how the output behaves. This can simply be the mean classification score from the classifier. In the example however, we used the implementation from Liaw and Wiener [70] which uses a logit scale:

$$f(x) = \log p_k(x) - \frac{1}{|\mathcal{Y}|} \sum_{j=1}^{|\mathcal{Y}|} \log p_j(x) \tag{4.4}$$

where x is the feature for which partial dependence is sought, $|\mathcal{Y}|$ is the number of classes, k the target class *Play Golf*, and p_j is the probability (classification score) of class j. This method ensures when a value of the feature results in a higher than average probability for class *Play Golf* (i.e., a classification score of 0.5 or higher on average), the value is positive. Likewise, when the value is negative, the feature results in a lower than average probability for the target class (smaller than 0.5). Moreover, it magnifies values closer to 0 and 1, and suppresses values around 0.5 to be closer to 0. A visual representation of the logit scale is shown in Figure 4.2.

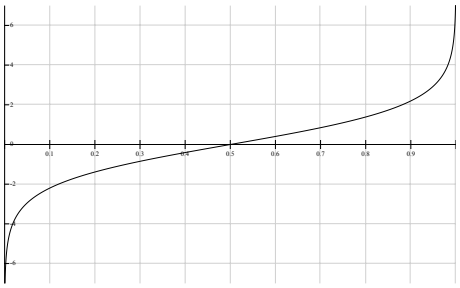


Figure 4.2: Logit scale. Function $\log(\frac{x}{1-x})$

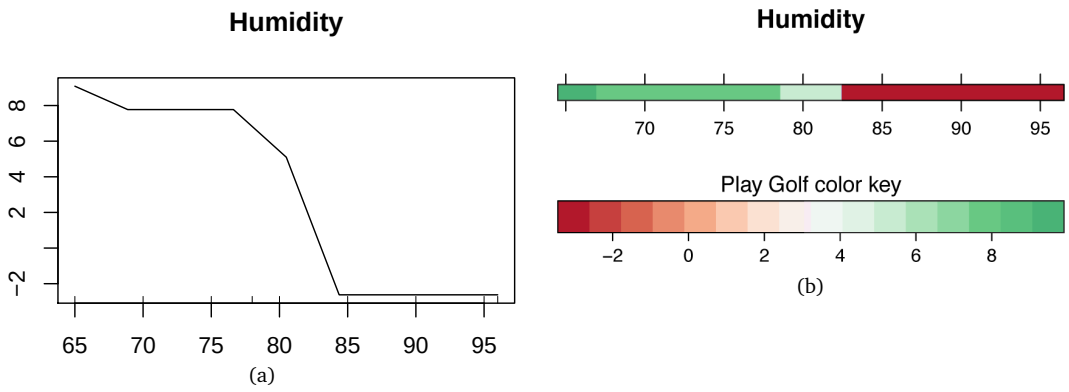


Figure 4.3: Sensitivity analysis of the *Humidity* feature in the Play Golf example for the Random Forest introduced in Section 2.3.6. Left: a partial dependence plot (PDP), right: a one dimensional colored level plot.

Figure 4.3 shows that around $Humidity = 82.5$ the metric greatly decreases, which means this feature value is less likely for the class feature. This directly corresponds to the choices made in the first tree of the model in Figure 2.5. This effect is so strong that the metric for $Humidity > 82.5$ is negative. This means that these values more often correlate to the class *Don't Play* rather than our target class *Play*.

Note that a line plot can only be used for continuous features. For categorical features, a bar plot can be used instead.

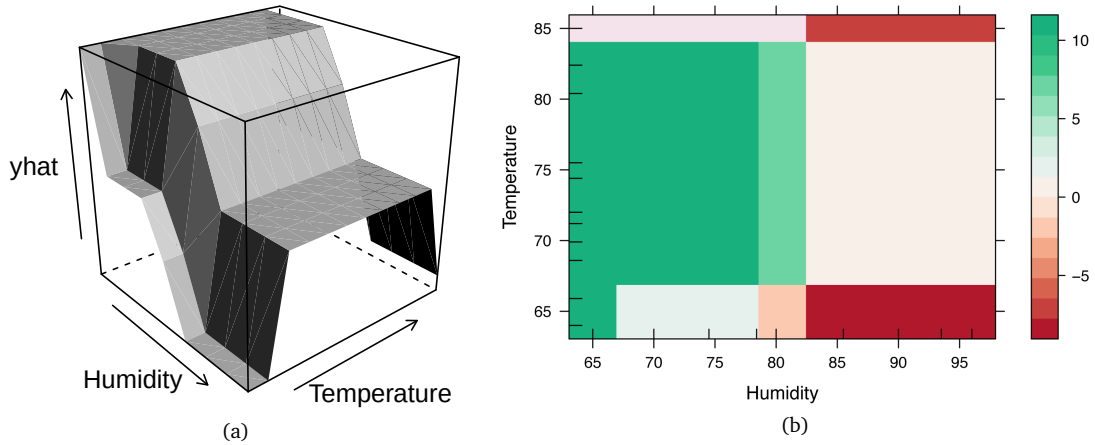
A more compact visualization technique is a one dimensional level plot, as used by Krause et al. [61, 62]. An example of such a visualization is shown in Figure 4.3b. The same data is used, but the model outcome metric is encoded as colors on the top bar. Green represents the positive *Play* class, and red the negative *Don't Play* class. The bottom bar is the legend showing which colors correspond to what score.

In addition to the single feature PDP, two features can be plotted in three dimensions [116] to show their combined effect on the model outcome. This visualization can uncover interaction effects between two features, as shown in Figure 4.4a. The two features *Humidity* and *Temperature* are plotted along the z- and x-axis respectively. The y-axis corresponds with the metric on how the output behaves, just like in Figure 4.3a. Note that the z-axis was flipped in order to prevent occlusion. This visualization can also be compressed into a two dimensional level plot where color represents the output metric (Figure 4.4b).

A slight interaction effect is noticeable: the probability for class *Play Golf* is especially low when the *Humidity* is high and the *Temperature* is either exceptionally high or low. As the 'Play Golf' example is really simple, these visualizations show very sudden changes. Realistic data sets would display more gradual curves and gradients.

PDPs for two features take considerably more time to generate as the entire surface of possible combinations needs to be sampled. Additionally, with no prior knowledge on which variables interact, the number of possible plots to generate is $\binom{|F|}{2}$, with $|F|$ being the total number of features in the data. This quickly becomes too much to give a comprehensible overview of the features. For instance, for a real world data set, the number of features is likely to exceed 20, in which case we need to draw 190 plots to cover all possible interactions.

A big disadvantage of PDPs is that they can obfuscate the modeled behavior. Consider the example by Goldstein et al. [40] shown in Figure 4.5. In Figure 4.5a 1000 generated data points are shown where roughly half of the points have a positive linear relationship with the class feature, and the other half an inverse linear relationship. Next, Goldstein et al. [40] train a Stochastic Gradient Boosting (SGB) model on the data (this is also a decision tree based ensemble model). The PDP created from this model looks like Figure 4.5b, and suggests there is no meaningful association with

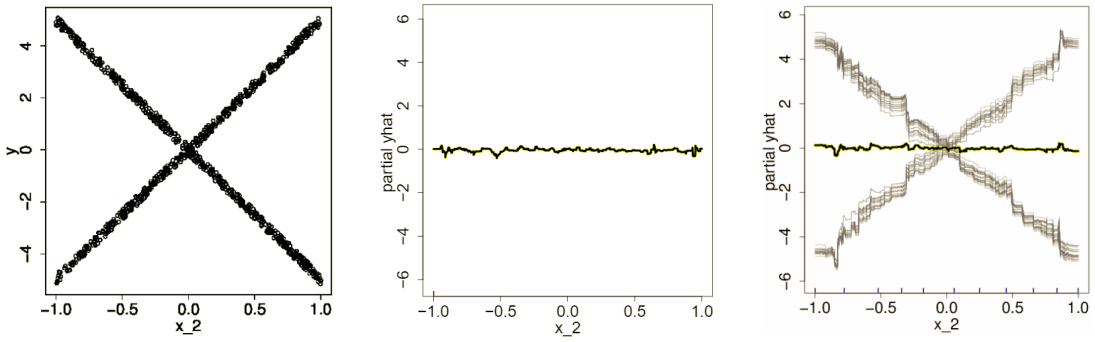


(a) (b)

Figure 4.4: Sensitivity analysis of the *Humidity* and *Temperature* feature for the Play Golf example for the Random Forest introduced in Section 2.3.6. Left: a three dimensional PDP, right: a two dimensional colored level plot.

the model outcome. This conclusion is clearly misleading.

Goldstein et al. [40] introduce a technique called Individual Conditional Expectation (ICE) plots that disaggregates the output of classical PDPs. Instead of measuring the average partial effect, the authors plot an estimated conditional expectation curve for every unique value of the feature of interest. It presents a global view of the model, by drawing all local insights into one plot. The result is shown in Figure 4.5c. In this plot the effect is clearly visible again. For real world applications the number of curves can become overwhelming. The author introduces two methods, centered ICE (c-ICE) and derivative ICE (d-ICE), which show effects much clearer compared to ICE for specific types of data.



(a) (b) (c)

Figure 4.5: Example showing the issue of PDPs where homogeneous feature behavior obfuscates the true effect. Left: 1000 generated data points where roughly half of the points follow a positive linear relationship with the class feature, and the other half an inverse linear relationship. Middle: PDP derived from a Stochastic Gradient Boosting (SGB) model, which misleadingly shows no effect. Right: Corresponding ICE plot in which the effect is visible again. Taken from Goldstein et al. [40].

4.2 Simpler model derivation

Model simplification methods work by deriving a simpler model that has the same behavior as the reference model to some extent. These simplified models have lower accuracy than the reference model, but are far less complex and thus easier to interpret. We distinguish two varieties of methods: *meta-learning*, a black box approach where another model is trained on synthetically generated data from the reference model, and *model condensing*, which is a white-box method that tries to reduce the size of the model significantly using queues from the model structure, while keeping the decision boundary and accuracy roughly the same.

4.2.1 Meta-learning

One of the first attempts to generate a more interpretable model was Combined Multiple Models (CMM) by Domingos [28, 29]. His method simplifies an existing model by generating synthetic data from a reference model. This can be visualized as placing ever more data points in the feature space. As the number of data points increases, the decision boundary will be clear from looking at the classification of these data points alone. If we train a different, more simple model on this synthetic data (meta-learning) and let it massively overfit on the data, it should approximate the same decision boundary as the reference model. This is a black box approach, as the only thing we need from the reference model is the output.

This method can be used to train any possible model, as long as it is compatible with the synthetically generated data. For instance, both the authors Buciluǎ et al. [14] and Zhou et al. [122] apply a very similar approach, and train a neural network on the synthetic data. This yields a very memory efficient model that could be beneficial in mobile and wearable devices where storage comes at a premium.

Various improvements have been proposed to reduce the complexity, and improve accuracy and comprehensibility of the simplified CMM model. CMMC-EVO [105, 106] uses an evolutionary approach that achieves a higher predictive accuracy than Domingos [28]. Unfortunately the authors have not evaluated the complexity and comprehensibility of the simplified model as compared to CMM. Also, there are different approaches to generate more appropriate synthetic data for CMM, like MUNGE [14] and very recently Model Based Sampling (MBE) [72] that work specifically for Random Forests. These methods yield synthetic data on which more accurate or smaller models can be trained.

An important thing to note is that the decisions the new model makes could deviate from the decisions made in the reference model; the only guarantee about the simplified model is that the decision boundary matches and thus will always yield the same output. This could result in misleading explanations, due to mismatches between the simplified and reference model.

4.2.2 Model condensing

To deal with this mismatch between simplified and reference model, *model condensing* methods exist that simplify the reference model itself directly. By using the decisions from the reference model, we ensure no different decisions are selected with the same effect on model outcome, ultimately matching the behavior of the reference model more closely. This is a white-box approach, as the structure of the model is considered, which also means that the described methods are only applicable to one specific model. For the purpose of this project, we only consider Random Forest model condensing.

Decision trees are a fairly easy to understand model as the trees remain small. Additionally, as Assche and Blockeel [2] argue, a decision tree in principle is powerful enough to represent the behavior of any tree ensemble. It is able to represent any function described over the instance space as it can separate the instance space completely if necessary. Hence many approaches focus on simplifying Random Forest models into decision trees. When the decision tree matches the decision boundary of the Random Forest exactly, it is likely to contain a lot of nodes. Hence simplifications are researched to decrease the size of these trees while containing the predictive accuracy as good as possible.

ISM One key tree ensemble condensing method is Interpretable Single Model (ISM) by Assche and Blockeel [2]. It creates a single decision tree that approximates the behavior of the Random Forest. The authors argue that, in order to match the behavior of the ensemble exactly, the resulting tree would contain 2^d leaves, with d being the number of constraints in the ensemble. This is way too big to be interpretable. They thus introduce a heuristic to iteratively determine the '*most informative*' node of the ensemble, to be used in the simplified model.

To retain the size of the simplified model, the method exploits the class distributions predicted by the ensemble. They calculate information gain and entropy with the same equations as described in Section 2.3.4, but rather than calculate p_i (the probability of the event given class $\mathcal{Y}_i$) as the fraction of items of class $\mathcal{Y}_i$ in the subset of data within the node, they estimate p_i using the law of total probability as

$$p_i \approx P_k(\mathcal{Y}_i|A) = \sum_{j=0}^{\#leaves\ in\ T_k} P(\mathcal{Y}_i|R_{kj} \wedge A)P(R_{kj}|A) \tag{4.5}$$

p_i is more explicitly written as $P_k(\mathcal{Y}_i|A)$, where $P_k()$ is the estimate of $P()$ by tree T_k, and A is a set of constraints for the simplified model subset during training. R_{kj} is the conjunction of constraints from the root of the tree T_k until the j^{th} leaf node N_{kj}. The class probability estimate $P_E(\mathcal{Y}_i|A)$ of ensemble E is the average over the class probability estimates $P_k(\mathcal{Y}_i|A)$ of N trees in E.

The authors also introduce a safe prepruning method and stopping technique to prevent that the simplified model introduces irrelevant splits, and becomes unnecessarily large.

The authors compare the result of ISM with Random Forests, CMM models and single decision trees. They evaluate the models using 34 UCI data sets [8]. They find that the decision tree yielded from ISM outperforms the single decision tree and CMM models, and is very close to the performance of a Random Forest.

The size of the simplified model is measured in number of nodes relative to a single trained decision tree. Where the trained ensembles is orders of magnitude larger than the single tree (25.73), both CMM and ISM stay relatively small with a 1.82 and 2.4 relative model size respectively, compared to a single decision tree. They also investigate stability of the mode, measured as the probability that the model generates the same prediction to a randomly selected instance. ISM and CMM both are as stable as a single decision tree, but lag behind the stability of the full Random Forest model.

CTC A different method for model condensing is Consolidated Tree Construction (CTC) by Perez et al. [93, 92, 41]. Their aim is to condense a tree ensemble model to a decision tree that is more stable than a single trained decision tree.

CTC works by considering the bootstrap samples from the training data which were used to train the tree ensemble. For all subsets the best split is determined based on information gain. Next, a consolidated split is determined by means of voting among all proposal splits. This consolidated split is applied onto all bootstrap samples, and the process is repeated. The algorithm yields a leaf node if most of the bootstrap samples return that not splitting has the most information gain. Pseudocode for the algorithm is available [93].

Gurrutxaga et al. [41] evaluated this method against CMM on 11 UCI data sets [8]. They measure complexity as the number of internal nodes and stability as the number of identical nodes per level, variable and division among two trees. Interestingly, CTC shows a higher error value as compared to CMM, being almost equal to the error of a single trained decision tree. The complexity of CTC, as compared to CMM, is lower though: in 10 out of 11 data sets the produced model contains fewer nodes. It is also more stable than a tree produced by CMM in 9 out of 11 data sets. They prove statistical significance for all their findings.

Rule extraction

Rule extraction is a special category of model condensing methods that seems to be especially apparent in Random Forest literature [43, 22, 96]. It is concerned with obtaining concise decision rules from a reference model. We argued in Section 2.3.5 that decision rules are an alternative way of representing a decision tree. The only key difference is that in these rules, there is no tight dependency between the constraints, whereas the tests in a decision tree need to be executed in a specific order. This makes rules easier to interpret, as we do not have to consider the structure of the tree [96].

Rules also convey the information of feature importance, as well as feature interaction. The most important features should appear more often in important rules, and features that interact should appear together in important rules.

Rule extraction is a method to simplify a tree ensemble model into a set of comprehensive decision rules. Clearly, this is a white-box method, as the structure of the forests in the trees is used to extract rules. All works encountered in this literature study focus on global interpretation of the model, although it should be possible to extract a set of rules for a local explanation of the model.

Even though the result of this method is fairly similar to decision tree model condensing, the way of approaching the problem is quite different. In literature it also seems a fairly separate field of research.

We discuss a few key techniques that extract a comprehensive list of decision rules from tree ensembles.

C4.5rules This technique was already present in 1993 as a tool provided with the original implementation of the C4.5 decision tree by [96]. This tool extracted rules by following all possible paths from the root to the leaf nodes. An example of such decision rules is shown in Section 2.3.5.

In addition the rules are pruned by trying to leave a constraint in the rule out and reevaluating the predictive error on the data [96]. If the predictive accuracy does not drop more than a predefined threshold, then the constraint can be omitted. Pruning can often significantly reduce the number of constraints in the rule, which makes the rules easier to understand. The process is concluded by filtering out duplicates that can arise as a result of pruning the rules, which further reduces complexity.

InTrees This technique developed by Deng [22] is an extension of the method presented above.

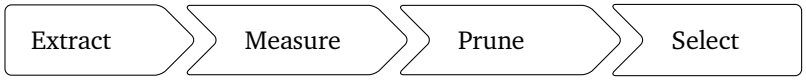


Figure 4.6: Four steps of the InTrees [22] process.

The framework consists of algorithms to *extract* rules, *measure* (or rank) rules, *prune* irrelevant or redundant constraints of a rule, and *select* a compact set of relevant rules. The extraction and pruning steps are equal to the approach of C4.5rules.

The measure step tests each extracted rule for *frequency* (proportion of data instances satisfying the rule), *error* (mean square error for the rule itself) and *length* (number of constraints in the rule). With these heuristics we can rank the rules and roughly determine the quality of each rule.

The second addition to C4.5rules is the select rule step. The list of rules produced in the first steps can contain redundancy. For instance, with two almost identical rules in terms of feature and split point, one rule conveys most of the information captured by the two rules already. One of the two rules is thus redundant and can be omitted.

The author introduced an innovative way to determine the importance of rules by means of using feature importance from a meta-trained Random Forest model [25]. The idea is to create a binary matrix with the training data along the rows and set of rules along the columns. The boolean values of the cells indicate whether the rule is applicable on the data instance. Next, a special type of Random Forest, called a Guided Regularized Random Forest (GRRF) [23], is trained on this matrix. A GRRF introduces a penalty coefficient λ_i for each feature i, used during the calculation of information gain:

$$G_{regularized}(S, f) = \begin{cases} \lambda_i \cdot G(S, f), & f \notin F \\ G(S, f), & f \in F. \end{cases} \tag{4.6}$$

where $G(S, f)$ is normal information gain as described in Section 2.3.4, and F refers to the set of indices of features used for splitting the previous nodes. It is an empty set at the root node. This means that the penalty coefficient is only effective if the feature has not yet been used for a split in the node.

For InTrees, the penalty coefficient λ_i is chosen based on the desired length of the rule. This means that the feature importance is biased towards shorter and thus easier to interpret rules.

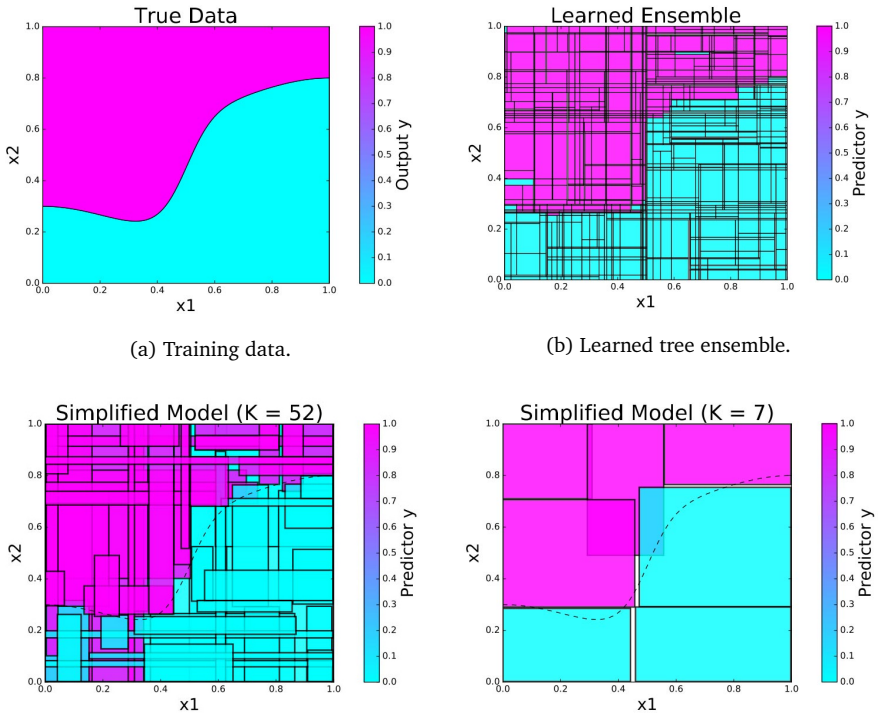
The algorithm trains the GRRF on the created matrix. Note that in the rules themselves are the features that predict the target class for the training data instances. This means that the feature importance of the model (as described in Section 2.3.6) will constitute a heuristic measure on the importance of a rule. Other than the regularization, the GRRF model is identical to a normal Random Forest. This means that feature importance can still be naturally computed during construction of the model. As the authors show, filtering the rules based on this rule metric eliminates irrelevant and redundant rules.

Conveniently, the authors provide InTrees as an R library, which can be used in conjunction with the default *randomForest* library.

DefragTrees DefragTrees is a very recent iteration of the rule extraction process by Hara and Hayashi [43]. They do this by reducing the problem to a Bayesian model selection problem [57]. This is a method to determine which model fits the data better, but is limited to work with probabilistic models only. It uses marginal likelihood, which is not defined for a tree ensemble model. The authors introduce a method to express a tree ensemble as a probabilistic model, which allows them to use Bayesian model selection methods. For more detail on how a tree ensemble can be expressed as a probabilistic model, we refer to their publication [43].

Once the model is expressed as a probabilistic model, the authors can apply Bayesian model selection approaches. The marginal likelihood can be estimated by using an Estimation-Maximization (EM) algorithm. This yields good results but is very computationally intensive. To deal with this complexity the authors use the recent Bayesian model selection algorithm Factorized Asymptotic Bayesian (FAB) inference [39, 46] to provide an asymptotic approximation of the marginal likelihood. This is 5 to 20 times faster than the EM algorithm, while retaining similar prediction error of the generated rules.

The evaluation seen in Figure 4.7 shows a vastly reduced number of rules as compared to InTrees, while it still describes the ensemble accurately. However, the authors only evaluated their method using two synthetically generated, and 4 UCI data sets for only part of the benchmarks.



(a) Training data.

(b) Learned tree ensemble.

(c) 52 rules extracted from (b) using InTrees.

(d) 7 rules extracted from (b) using DefragTrees.

Figure 4.7: Evaluation of the decision boundary of Defrag trees (d) as compared to the original training data (a), trained ensemble (b) and rules extracted by InTrees (c). The number of rules is significantly reduced, while the predictive accuracy is maintained (decision boundary matches the training data). Taken from Hara and Hayashi [43].

4.3 Structural visualization

Finally, there are a few methods we encountered that did not fit the aforementioned categories. They visualize the tree ensemble directly by showing the structure of the model. This is not relevant for our purposes, as the structure of the model alone does not tell us anything about the decision making process of the model.

A noteworthy example by Lau [68] is shown in Figure 4.8, in which an ensemble of 500 trees is shown. Each row represents a node of a tree. The cells show aggregated information about trees that follow that same structure. An overview is shown of all the possible constraints the trees use as root node. feature $m5$ is used in most trees (172), while $m2$ is hardly ever chosen as root node (4 times). By clicking on the row corresponding to $m5$ it unfolds, showing the constraints that follow in trees with this root node. The example shows that $m4$ is often the right child of root node $m5$. For all trees that have a root node with a constraint on feature $m5$, followed by a decision on feature $m4$, the class distribution of all leaf nodes is displayed in the cells of the row. For instance, the example shows that *class-1* and *class-2* are more likely to be predicted by these trees than the other classes. The further the forest is navigated, the more distinguished this class distribution becomes.

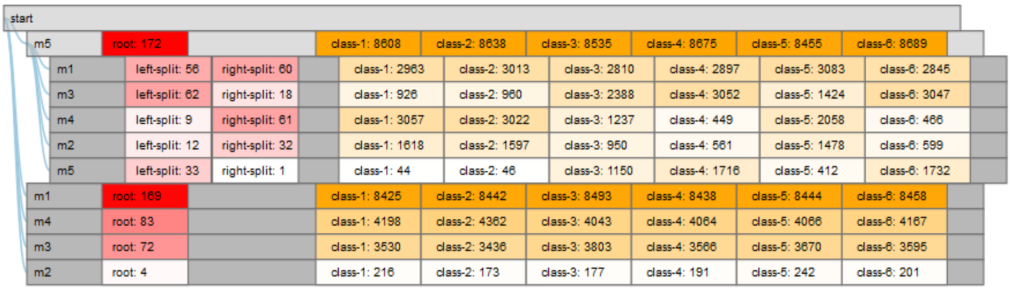


Figure 4.8: Example of structural visualization by Lau [68]

4.4 Conclusion

We have shown various methods in order to extract useful information from the model. Feature analysis methods give an insight on which features of the data are relevant for the model outcome. Various methods for this exist, such as feature importance metrics and sensitivity analysis. A different field of research which is also relevant for getting insights into model behavior are model simplification methods (including meta-learning) such as CMM, and model condensing. We highlighted a special case of model condensing, rule extraction, which derives decision rules from an ensemble. Finally, structural visualization exists which makes it easy to navigate the structure of a complex tree ensemble.

These methods can be combined in a single dashboard visualization in order to interpret the model. Which methods are required heavily depends on the context of the problem. Structural visualization can be beneficial for Machine Learning model developers, but gives no insight in how a decision is made, and is thus not relevant for the purpose of this project. Feature analysis on the other hand can give a great insight in how the model operates, but is a heavy simplification and does not capture feature interactions. Model condensing can be used to capture the behavior of the model while incorporating feature interactions, but can be complex to derive.

Chapter 5

Visualization

In this section we present visualization techniques to facilitate the user tasks mentioned in Chapter 3. We use three key methods identified in the previous chapter: feature importance, sensitivity analysis and simpler model derivation. For the latter method, we focus on rule extraction, as there has been little to no research into visualizing decision rules. We reckon such rules convey a lot of information while still remaining simple to interpret.

To identify relevant features (Task **T3.2**), feature importance metrics can be visualized. Relevant combinations of features (Task **T3.3**) can be found by analyzing decision rules obtained with rule extraction. Finally, visualization of sensitivity analysis, as well as rule extraction, can show the relationship between feature and model outcome (Task **T3.4**).

5.1 Feature importance

There are various ways of displaying feature importance information, depending on the complexity needed to be conveyed. In the following section a variety of methods is presented, ordered from simple to more complex.

Ranked list For starters, for very simple insights, it can be sufficient to show a ranked list of features (Figure 5.1a). This is useful if the features need not to be compared relatively, and are combined with more visual elements in a dashboard. In such case it can be desirable to keep the complexity low, such that the user is not overloaded with information.

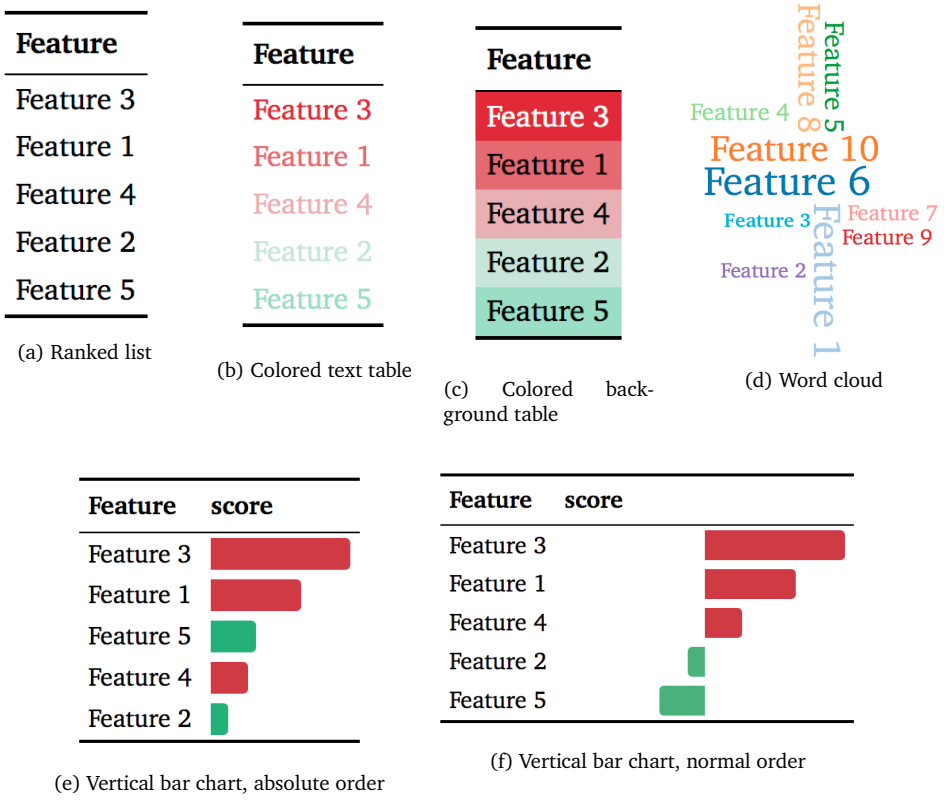
Colored table A slight variation is a colored feature ranking (Figures 5.1b and 5.1c). Here the color of text or cell background can give a clue on how the features relate, while keeping the complexity of the presented information low. Although the color can give an insight in relative differences, it can be difficult to spot small differences between features.

Word cloud A variation of colored text or background is a list where, instead of color, the font *size* is varied according to the ranking score (Figure 5.1d). Like color, it can be difficult to distinguish small differences in font size, but gives a good overall insight in the relations between features.

Vertical bar chart In case the relative differences are more important, a vertical bar chart (as shown in Figures 5.1e and 5.1f) can be used. This enables easy comparison between features, even if the differences are tiny. It is also very natural to show the negative importance introduced by Palczewska et al. [91], as the bar can just cross the y-axis.

There are multiple ways to order such a list. In Figure 5.1e the features are ranked according to the absolute value of the local importance score. This way features with an important impact, whether this is a positive or a negative influence, are always displayed on top. This achieves a similar result as the global feature importance scores, which cannot distinguish between positive and negative influences. This information can be retained by coloring the bars in the charts according to their influence.

In Figure 5.1f the features are ordered to the local importance *without* absolute value. This way the features with a negative influence appear all the way at the bottom of the list. This ordering makes it easier to consider all negative-impact features as a group, and compare them to the positive-impact features.



(a) Ranked list

(b) Colored text table

(c) Colored background table

(d) Word cloud

(e) Vertical bar chart, absolute order

(f) Vertical bar chart, normal order

Figure 5.1: Various methods to visualize feature importance rankings.

For the Achmea model we deal with an ensemble of different Random Forests. To show the feature importance the average of the scores of all models is considered. However, it could be interesting to see the variance of the models as well, showing the amount of agreement among the models. There are again a variety of different methods to display this information.

Vertical box plot A standard visualization for displaying the distribution of data is a box (-and-whisker) plot [112]. It depicts groups of numerical data based on the *quartiles*: sets of data values obtained when dividing the data in four equal portions. An example is shown in Figure 5.2. A box is drawn for the area between quartile 1 and 3 (Q1 and Q3), which is where 50% of the data is located. The width of this box is also called the inter-quartile range (IQR). There are multiple ways of drawing the whiskers, but most predominantly $1.5 \times IQR$ is used as the distance from Q1 and Q3. Everything outside this range considered an outlier. Outliers are often ignored, but can also be drawn as a dot.

For the feature contribution ranking a vertical version of a box plot can be used to incorporate information about variance. An example of such a plot is shown in Figure 5.3a. The white dot in the middle of the plot represents the median value. *Feature 3* seems to have a small variance: most forests give this feature a high contribution score. In contrast, the forests vary quite a lot for *Feature 5*: some forests even attribute the feature a positive importance. This feature is thus not unanimously ranked as a low-contribution feature.

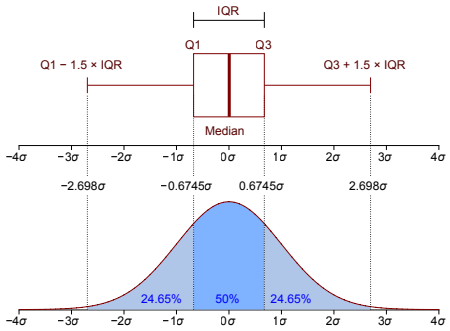
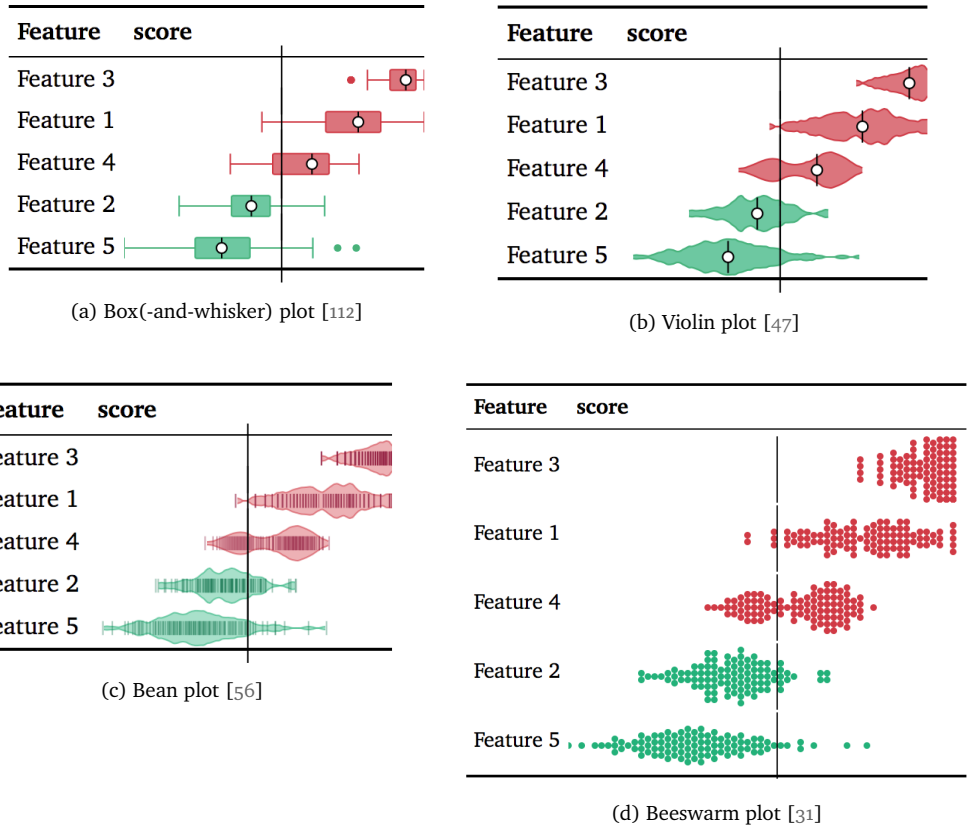


Figure 5.2: Box plot with its corresponding density distribution.

Violin plot A different way of visualizing density information is a violin plot [47]. It uses Kernel Density Estimation (KDE) to obtain an estimate of the probability density function of the data distribution. Next, this density function is used for the width of an area chart, resulting in plots like in Figure 5.3b. This plot can be combined with a box plot to show the IQR as well. A violin plot can reveal more information about the density than a box plot. Consider *Feature 4* in the box plot and violin plot. The violin plot example shows there is a portion of the forests claiming feature importance around 0.1, as well a slightly larger portion of the trees scoring around 0.3. These two clusters are not visible in the box plot example. It generalizes to only shows that the values range between 0.1 and 0.3, but not that there is relatively few forests scoring in between those peaks.

Bean plot A more recent visualization devised for this kind of information is a bean plot [56]. It is similar to a violin plot as it also shows a density trace, as is shown in Figure 5.3c. However, as the authors note, individual data points besides the minimum and maximum are not visible in a violin plot, and no indication of the number of observations in a group is given. A bean plot is a combination between a 1d-scatter plot and a density trace, which also reveals outliers. The 1d-scatter plot, often referred to as a RUG plot, draws a line for every forest score in the data. This way we can spot outliers for *Feature 5* around -0.8 and 0.4 that would often be unnoticeable in a violin plot, as the density trace averages them out.

Beeswarm plot Another recent visualization is called a Beeswarm plot [31]. An example is shown in Figure 5.3d. For every forest, its score is drawn as a small dot. The density of the dots corresponds with the density function, obtained in a similar fashion as the violin plot. This plot is intuitive as there is a notion of individual forests in the plot, and their position directly corresponds with the feature contribution score. This also holds for RUG plots, but in such a plot lines often overlap, making it more difficult to interpret the density. This plot only works with relatively few data points, but works well in our case with 100 forests. It does take considerably more space than the previous plots.



(a) Box(-and-whisker) plot [112]

(b) Violin plot [47]

(c) Bean plot [56]

(d) Beeswarm plot [31]

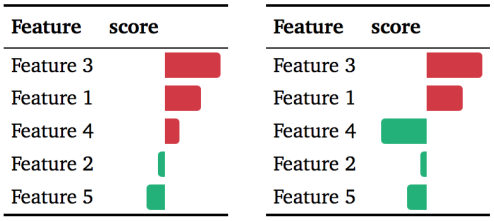
Figure 5.3: Various methods to visualize feature importance distributions.

It can also be informative to show the deviation of the local feature importance from the global importance. This can identify which insurance policies are classified just like most others, as opposed to policies for which very different decisions are made. The latter category can have vastly different feature importance scores than typical policies. Various options are available to compare rankings:

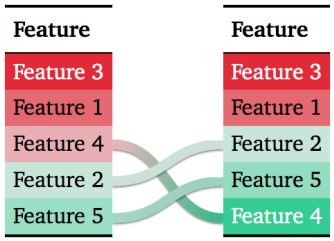
Side-by-side Watching two rankings side by side can already give a good insight in the differences. When every row shows the same features, it is easy to compare them. This means that lists have to be either sorted by local, or global importance, but not both. For this comparison colored tables or vertical bar charts can be used. For the latter, an example is shown in Figure 5.4a. It shows that feature 4 diverges quite a lot from the global importance, while the other features remain similarly important.

Bumps chart Another option is to sort both rankings according to their importance, and draw lines connecting the same feature in each list. This is called a bumps chart, and is shown in Figure 5.4b. The chart was originally designed for comparing the position of several teams within a competition. It works best when the compared rankings only differ slightly. If the rankings diverge too much, the many crossing lines will make the visualization unclear and confusing. The example shows that *Feature 4* moved down in the ranking, but *Feature 2* and *Feature 5* moved up as a consequence.

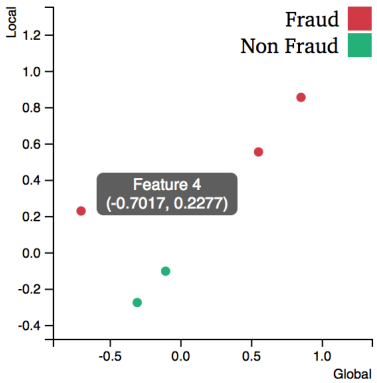
Scatter plot A different approach is to plot every feature as a scatter plot with the local feature importance in one dimension, and the global importance in another. An example is shown in Figure 5.4c. The more the feature importance rankings match up, the more a linear trend will be visible. The example shows that *Feature 4* deviates from the rest of the features.



(a) Side-by-side comparison. Left: local, right: global.



(b) Bumps chart



(c) Scatter plot

Figure 5.4: Various visualization methods to compare feature importance rankings.

5.2 Sensitivity analysis

Existing visualization techniques for sensitivity analysis are largely covered in Section 4.1.2. As we seek to discover insights on a per-feature basis, we only focus on the PDPs concerning a single feature (as shown in Figure 4.3a).

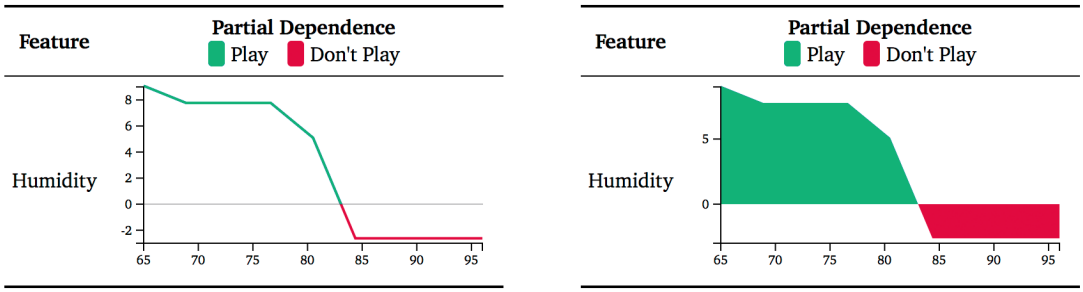
In literature often line plots are used to plot the partial dependence. For the y-axis traditional methods usually use a logit scale, which we discussed in Section 4.1.2. This scale can magnify the effect shown in the plots, making it easier to infer for which values the model outcome changes. However, the meaning of the numbers on the scale can be difficult to explain to the end user.

As an alternative, we propose using a linear scale corresponding to the classification score returned by the model. On this linear scale, numbers directly correlate to the model score. This can be much easier to interpret by the end user.

Another lack of conventional methods is that normal line plots understate the importance of the switch point from one class to another. For the logit scale, positive values indicate a positive effect on the target class, whereas negative values mean the model is more likely to predict the non-target class. Yet, this is not explicitly indicated in conventional PDPs. We propose two visualization methods to make this switch point at $y = 0$ more salient, shown in Figure 5.5. We use the same colors and data as used in our running example in Section 4.1.2. In the figure a logit scale is used. This means that positive values correspond to our target class (*Play*), and negative values to the remaining class (*Don't Play*).

Colored line chart First we propose changing the color of the line in the plot according to the class they represent (see Figure 5.5a). If the plot was created for the *Play* class, positive values represent a positive impact on that class, and are thus colored green. Likewise, negative values have a negative impact on class *Play*, and are colored red.

Colored area chart The change in color on a line plot can be difficult to discern. To make the switch stand out more, we propose to use an area chart, like in Figure 5.5b. This way the switch between positive to negative numbers is a lot more salient. The large colored area also has a higher visibility than a line. This means that the area chart can be comprehensible even when the chart is small, whereas it would be difficult to spot the line in a line chart of similar size.



(a) Colored line chart

(b) Colored area chart

Figure 5.5: Various methods to plot partial dependence information.

5.3 Rule extraction

To the best of our knowledge, very few authors have focused on developing visualization techniques for decision rules. One of the few works that we found, by Ilczuk and Wakulicz-Deja [52], simply deduces a decision tree to visualize the rules. This is a plausible visualization technique, but does not use the advantage that rules do not have a strict order.

We propose the following methods to display a set of rules in a comprehensive manner.

1D range plot The most basic visualization is a list of one dimensional plots of the constraint range within the feature range. An example is shown in Figure 5.6a which shows three rules consisting of two constraints each, for the Play Golf running example.

The first rule contains a constraint on feature *Humidity* > *77.5*. This corresponds to a path in the trees of the Random Forest trained for this problem, shown in Figure 2.5. The range of the feature is derived using Equation 6.1 introduced earlier. In the example, the range of the values covered by the constraint is marked. The color corresponds with the class that the rule predicts; in this case red for *Don't Play*.

This visualization can be expanded by plotting a histogram on the x-axis to derive how much data is captured by the constraint. If the histogram contains information on the distributions of classes within each bin, the accuracy of the rules can also be intuitively derived. When showing this information, one has to take into consideration that it is the combination of all rules that produces the model outcome, not a single rule or constraint. Considering that a model exists out of a multitude of rules, it is unlikely that the individual constraints show a clear class distinction.

Multi-dimensional plot Conceptually, a rule of length l can be seen as a l-dimensional hypersurface dividing the feature space into two different regions: one belonging to the target class, and one which does not.

If the dimensionality of this hypersurface is small (two or three dimensions), the surface can be plotted directly. This approach for the first rule in Figure 5.6a is shown in Figure 5.6b.

Like the histograms in the 1d rule plot, it is interesting to see how the rules relate to the actual data. A l-dimensional projection of the data can be plotted within the chart to show actual data samples within the feature ranges. The same benefits apply: it becomes intuitive to derive the amount of data captured, and to some extent the predictive accuracy of the rule.

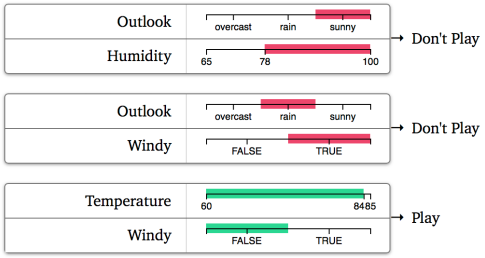
The drawback of this method is that rules with more constraints cannot be represented directly. In Chapter 9 we propose a method how this problem can be circumvented.

Sankey diagram This method focuses more on the causality of the produced prediction, inspired by the work by Scheepens et al. [100]. Instead of the constraints, the target classes are used as a starting point of the visualization. From here, every target class is connected with rules that contribute to the prediction of that class. These rules are again connected to constraints that comprise this rule.

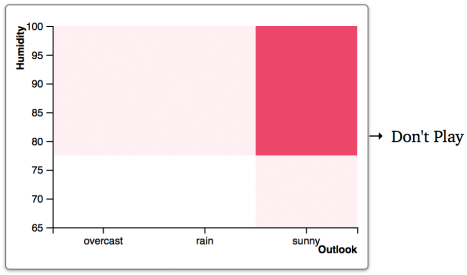
How much each rule and constraint contribute can be effectively captured by a Sankey diagram, as shown in Figure 5.6c. In this example the model predicts *Don't Play* when it is windy (*Windy* ∈ *(TRUE)*) and raining (*Outlook* ∈ *("rain")*). Likewise, the model predicts *Play* when it is not windy (*Windy* ∈ *(FALSE)*) and not too warm (*Temperature* ≤ *84*).

Metrics for the importance of each individual rule, as well as feature importance, can be encoded in the width of the edges of the Sankey diagram. More important rules and constraints are connected with wider edges, and thus stand out more in the visualization. Rule importance is shown in the example in Figure 5.6c as a larger edge is connected to the top rule than the bottom rule. Likewise, feature importance is shown as the constraint on *Temperature* is connected with a more narrow edge than the ones connected to the constraint on the *Windy* feature.

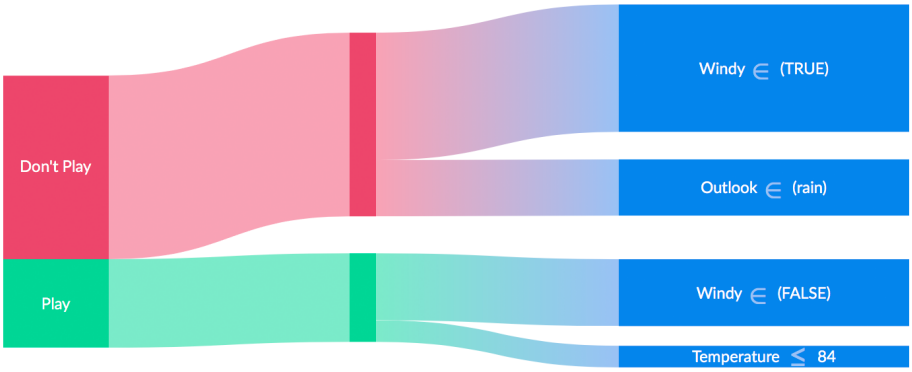
This visualization works best when the number of rules is limited. When too many constraints and/or rules are present, it is difficult to derive meaningful insights. As our goal is to visualize a concise set of the most important rules, this caveat does not apply to our project.



(a) 1D range plot, with three rules with two constraints each.



(b) Multi-dimensional plot of one rule with two constraints.



(c) Sankey diagram of two rules of two constraints each.

Figure 5.6: Various methods to visualize decision rule rankings obtained with rule extraction.

5.4 Conclusion

We have shown various methods of visualization in order to give insights corresponding with the user tasks defined in Chapter 3. These visualizations range from simple to more complex, depending on required complexity. For a rudimentary insight, simple visualizations are sufficient. For instance, if information is displayed in a dashboard along more visualizations, a complex visualization can be overwhelming, and a more simple depiction can be desirable. The user can be given the option to switch between visualization techniques according to its preference.

These visualizations form the building blocks for our visualization dashboard for the Achmea model. To this end, we first apply these techniques on the model, and obtain insights on the model in the process.

Chapter 6

Model Analysis

In this section we analyze the Achmea model by means of the proposed methods in Chapter 3. We consider the same three key methods used in the previous chapter: feature importance, sensitivity analysis and rule extraction.

For each method, we investigate both global, as well as local views on the model. An overview of the structure of this chapter is shown in Table 6.1. We introduce a novel method of local rule extraction in Section 6.3.2.

Table 6.1: Structure of the Model Analysis chapter.

	Global	**Local**
Feature importance	Breiman [11]	Palczewska et al. [91]
Sensitivity analysis	Friedman [38]	Krause et al. [63]
Rule extraction	Deng [22]	Collaris

6.1 Feature importance

We start by identifying relevant features (corresponding to Task **T3.2**) for the Achmea model, both globally for the entire model, as well as locally for specific insurance policies.

6.1.1 Global

The original Random Forest paper by Breiman [11] included a method to compute the global importance of individual features (see Section 2.3.6). In order for this method to work with the Achmea ensemble model, we made a modification to the algorithm so it can be used with a multitude of Random Forests. For this end, we use the mean of all importance scores of all forests. The resulting global feature importance score for the entire model is shown in Table 6.2.

Feature	MeanDecreaseGini
Feature 25	1.000
Feature 7	0.748
Feature 15	0.607
Feature 24	0.604
Feature 14	0.599
Feature 29	0.597
Feature 27	0.567
Feature 6	0.532
Feature 8	0.531
Feature 28	0.510
Feature 2	0.501
Feature 45	0.483
Feature 34	0.478
Feature 44	0.468
Feature 26	0.454
Feature 41	0.447
Feature 40	0.445
Feature 35	0.442
Feature 33	0.436
Feature 36	0.430
Feature 23	0.405
Feature 46	0.397
Feature 38	0.377
Feature 18	0.375
Feature 12	0.370
Feature 47	0.370
Feature 5	0.370
Feature 17	0.360
Feature 20	0.346
Feature 43	0.345
Feature 19	0.335
Feature 37	0.300
Feature 48	0.291
Feature 13	0.269
Feature 22	0.246
Feature 16	0.235
Feature 21	0.234
Feature 42	0.232
Feature 4	0.223
Feature 32	0.222
Feature 3	0.222
Feature 39	0.221
Feature 49	0.212
Feature 10	0.201
Feature 11	0.199
Feature 30	0.168
Feature 31	0.149

Table 6.2: Global feature importance of the Achmea fraud detection model.

The two factors *Feature 25* and *Feature 7* are most important overall, and *Feature 30* and *Feature 31* seem to have almost no impact. Interestingly the score of the rest of the features seems to decrease fairly linearly. This indicates that most features are relevant for the classification rather than just a small subset.

To analyze the variance of importance scores among Random Forests, we also created a box, violin, bean and beeswarm plot of the data. The violin plot is shown in Figure 6.1.

The variance seems to be a lot higher for high importance features than low importance features. In fact, the variance seems to increase linearly with the importance. There are a few exceptions to this, as *Feature 30* has a few outliers with significantly higher importance, and *Feature 6* and *Feature 8* which have lower variance compared to the features around them.

The only few exception to this are the features *Feature 2*, *Feature 6* and *Feature 27*. For these features there is a small bump after the main curve, indicating that a subset of forests had such a different importance score, that it is visible in the plot.

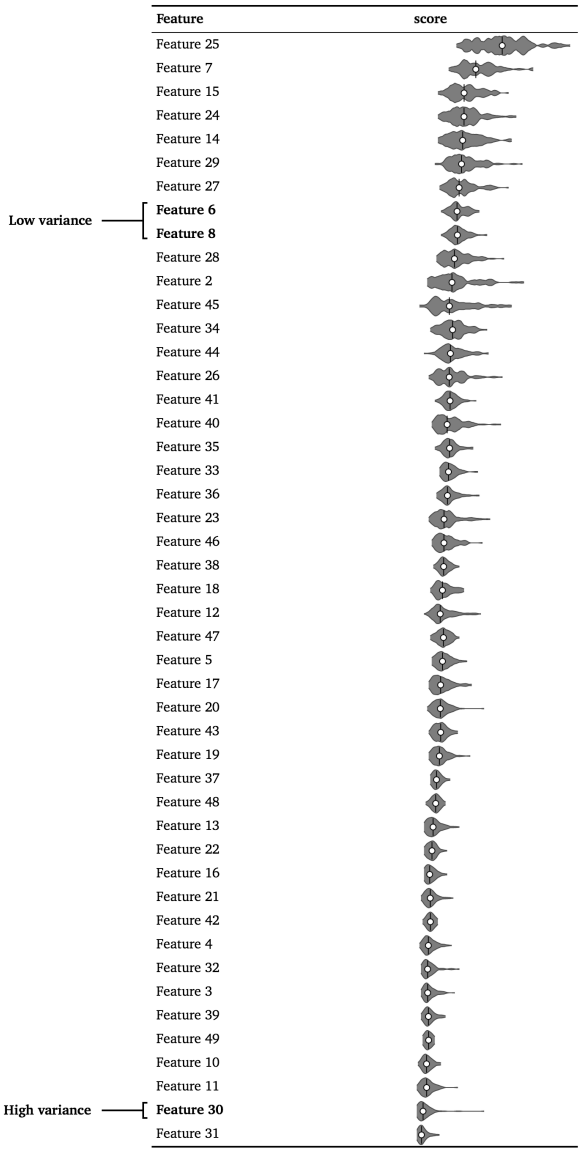


Figure 6.1: Violin plot of the global importance for the Achmea ensemble model.

6.1.2 Local

A global feature importance ranking can be misleading. For a certain subset of the data, a feature could be very important, while for another subset it may not. To more accurately describe the model, we need local feature importance. This shows the features that were most relevant for the model outcome of a single instance. Local feature importance also more closely answers the main question by Achmea, which is concerned with a per-instance explanation.

We applied the feature contribution method by Palczewska et al. [91] to obtain a feature importance metric for an individual classification. This represents the features that were most important for the classification of that single insurance policy. As the method by Palczewska et al. [91] uses local increments calculated for a specific target class, the method also includes information on features that have a negative impact on the model outcome. For instance, if the class of interest is "Fraud", and a constraint on feature f_i leads to a subset of data in which the fraud class is less abundant, the score for that feature would be negative. For our visualization we used the target class "Fraud".

An example for case *ANP128* (fraud) is shown in Table 6.3a. The feature importance for instance *ANP128* corresponds fairly well with the global feature importance; *Feature 25* and *Feature 7* both appear as most important features. However, as the method by Palczewska et al. [91] considers local increments for a specific target class, we can distinguish between features that had a positive or negative impact on the predicted class. Consider the feature *Feature 2* in the local explanation. It has a very negative influence on the prediction. This means that the feature leads the model to believe that the instance was non-fraud instead of fraud. When the feature is not considered, the model would be more certain that the instance was fraud. The global feature importance method is unable to distinguish this difference, and will rank the feature *Feature 2* among the positive scoring features.

Feature	Feature contribution (Geen fraude / Wel fraude)
Feature 25	1.000
Feature 7	0.521
Feature 45	0.498
Feature 8	0.348
Feature 24	0.340
Feature 40	0.331
Feature 26	0.291
Feature 18	0.291
Feature 29	0.288
Feature 46	0.271
Feature 28	0.264
Feature 33	0.253
Feature 15	0.244
Feature 17	0.241
Feature 19	0.234
Feature 23	0.233
Feature 47	0.226
Feature 14	0.220
Feature 27	0.210
Feature 20	0.203
Feature 22	0.192
Feature 37	0.150
Feature 38	0.147
Feature 5	0.141
Feature 11	0.140
Feature 16	0.125
Feature 35	0.123
Feature 43	0.112
Feature 21	0.109
...	...
Feature 39	−0.142
Feature 44	−0.244
Feature 2	−0.586

(a) ANP128 (fraud)

Feature	Feature contribution (Geen fraude / Wel fraude)
Feature 34	1.000
Feature 44	0.607
Feature 2	0.451
...	...
Feature 21	−0.103
Feature 38	−0.105
Feature 6	−0.122
Feature 17	−0.125
Feature 23	−0.134
Feature 19	−0.134
Feature 20	−0.145
Feature 3	−0.150
Feature 4	−0.154
Feature 41	−0.171
Feature 40	−0.180
Feature 14	−0.185
Feature 24	−0.190
Feature 15	−0.225
Feature 28	−0.258
Feature 25	−0.264
Feature 26	−0.308
Feature 29	−0.348
Feature 7	−0.359
Feature 27	−0.379

(b) ANP87 (non-fraud)

Table 6.3: Local feature importance of the Achmea fraud detection model for insurance policy (a) *ANP128* (fraud, score: 0.88) and (b) *ANP87* (non-fraud, score: 0.256). Low importance features between -0.1 and 0.1 are filtered out.

From Table 6.3 it is clear that the importance for individual cases can differ significantly. Where in the global ranking and ranking for *ANP128*, feature *Feature 25* is very important, it plays a way less important role for *ANP87* (Non-fraud). Likewise, the feature *Feature 34* is the most important feature for *ANP87*, while it is not that important globally, and does not even appear in the ranking for *ANP128*. These differences shows that global importance can indeed be misleading, and is not representative for every instance.

Some features for *ANP87* also seem to be the opposite from *ANP128*: *Feature 7* has high contribution for *ANP128* but is a very negatively affecting the classification for *ANP87*. This can be explained by the value for the feature for *ANP128* (Fraud) being highly indicative of fraud, whereas the value for *ANP87* (Non-fraud) is not. As the feature in general has a high importance, the impact (or size) of the bar is high, but whether it is negative depends on the value of the instance.

To analyze the variance of importance scores among trees, we also created a box, violin, bean and beeswarm plot of the data. The variance for local feature importance seems to behave very similarly to the global scores. There is again a correlation between importance and variance, and there are no conflicting scores between groups of trees that are worth exploring further.

6.2 Sensitivity analysis

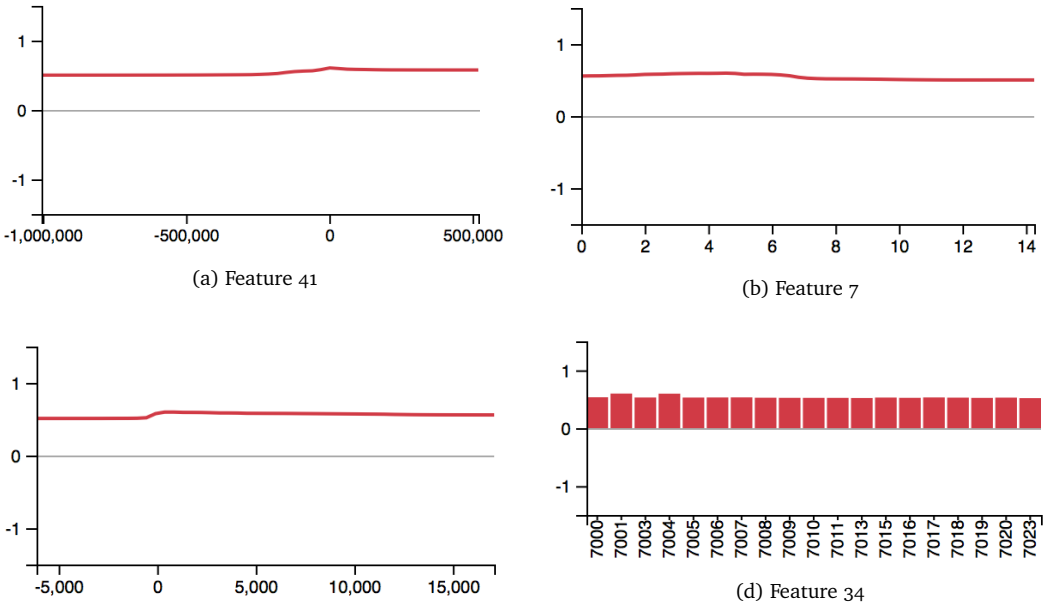
To get a better insight into the relation between features and model outcome (Task **T3.4**), we can enrich the visualization by providing some information about the ranked features. One interesting insight is to see how the outcome of the model changes when the value for a feature is changed, which corresponds with Task **T3.4**. Sensitivity analysis can be used to extract these kind of insights.

6.2.1 Global

We used the partial dependence method by Friedman [38] to extract global PDPs for every feature in the data set. This is a computationally intensive process. For n different feature values observed in the data set for feature of interest f, the model needs to classify $N \times D \times n$ records, with N being the total number of records in the training sets, and D the number of samples to take within the feature range. For the entire Achmea data set and choosing $D = 50$ for numerical features, and the domain D_{f_i} for categorical features, the number of classifications to make is 80,621,685. This, combined with the fact that classification by an ensemble of a 100 forests can take significantly longer than a single model, makes it impossible to calculate the partial dependence with the entire training set within reasonable time. Instead, a sample of 500 records is used, in which all 129 fraud cases are included. Reducing the number of records not only reduces N, but also n, as there are fewer possible values for each feature in a smaller subset. This reduces the number of classifications to 1,057,500, which is more manageable. This process still took roughly 1.5 hours to complete.

Even though 500 records is a small subset from the entire data set, the partial dependence seems to capture the same effects. We ran the PDP algorithm for a few categorical features on the entire data set and found very similar results to the ones found with the subset.

We sorted the plots based on their variance. The top 4 plots are shown in Figure 6.2, and all plots are attached in Appendix C. For the y-axis scale, a logit scale was used, and ranges from -1.5 to 1.5.



(a) Feature 41

(b) Feature 7

(c) Feature 46

(d) Feature 34

Figure 6.2: Global partial dependence plots for the three continuous features with highest variance (a,b,c), and highest variance categorical feature (d).

Partial dependence for all features seems to average around 0.55. This is because the average predicted score of the model for the subset of 500 records is 0.726, which on logit scale works out to be 0.562. If a feature has hardly any effect, the line in the plot stays close to the average predicted score.

This seems to be the case for almost all features. The highest variance among all PDPs is 0.00138 for feature *Feature 41*, with a range of 0.104.

A possible explanation for this small effect is that there are many interactions taking place between features. If an interaction takes place between feature f_1 and f_2, the prediction only changes significantly when both features are changed, but not individually.

Another contributing factor is that the problem of fraud detection is very difficult, and as mentioned in Section 2.3.6, a single Random Forest is not able to generalize any meaningful function for fraud detection. It is thus unlikely that a single feature has significant impact on the prediction, as in that case the problem would have been easy and a single Random Forest model would have been sufficient.

Finally, and perhaps the most contributing factor, is that a lot of detail can get lost as a result of averaging [40]. We mentioned this problem earlier in Section 4.1.2. When the model behaves differently for two subsets of the data, the average of both behaviors can be inaccurate for both.

In the plots, we also see some counterintuitive results. The PDP for feature *Feature 7* (corresponding to the premium percentage) shows that the model is more likely to predict fraud for insurance policies with a low percentage of premium. We expect (and also verified with the fraud team) that fraud cases usually have a higher premium percentage, as this increases the revenue for fraudsters. In our local findings, this pattern also emerged.

We hypothesized that features sorted on variance should match the global feature importance ranking. For instance, the feature *Feature 7* is considered important by both the variance and feature importance ranking. This however does not always seem to be the case. Feature *Feature 25* has one of the lowest variances, while being the best feature according to feature importance. Likewise, feature *Feature 41* has the highest variance, but is only the 14^{th} best feature considering the feature importance ranking.

It is clear that not much can be learned from the global PDPs. It seems that too much information is lost due to averaging over complex feature interactions. Alternatively, only 500 records could be insufficient to capture the model behavior. Instead we may again look at local effects, which should provide more accurate information for a specific instance.

6.2.2 Local

We have not found libraries that explicitly support local partial dependence plots. All implementations use the training data to determine possible feature values to vary features. For local partial dependence we only want to consider one instance, from which the implementations cannot derive multiple feature values.

In literature local partial dependence plots seem to be uncommon as well. An exception is the work by Krause et al. [63, 61, 62], who introduced the colored bar plot mentioned in Section 4.1.2. Otherwise PDPs are seen as a global interpretation tool.

We decided to create our own implementation. Synthetic data points within the ranges of the features are created. For categorical features, all possible values are used. For continuous features, we create 50 data points equally spaced along the feature range. Some feature ranges contain extreme values due to incorrectly filled in data. *[As a consequence of anonymization, a concrete example of outliers has been removed.]* To remove such outliers, the feature range is set to:

$$\begin{aligned}&[\max\big(\min(X),\ \mu_X - 3\sigma_X\big),\\ &\ \min\big(\max(X),\ \mu_X + 3\sigma_X\big)]\end{aligned} \tag{6.1}$$

For each synthetic data point, a copy of the instance of interest is permuted to contain the synthetic value. Next, we classify all records and plot the values.

The implementation collects results in both a linear, and a logit scale, such that the decision for which scaling to be used can be configured by the end user. For the results shown in this section, we applied logit scaling so they are comparable with the global results.

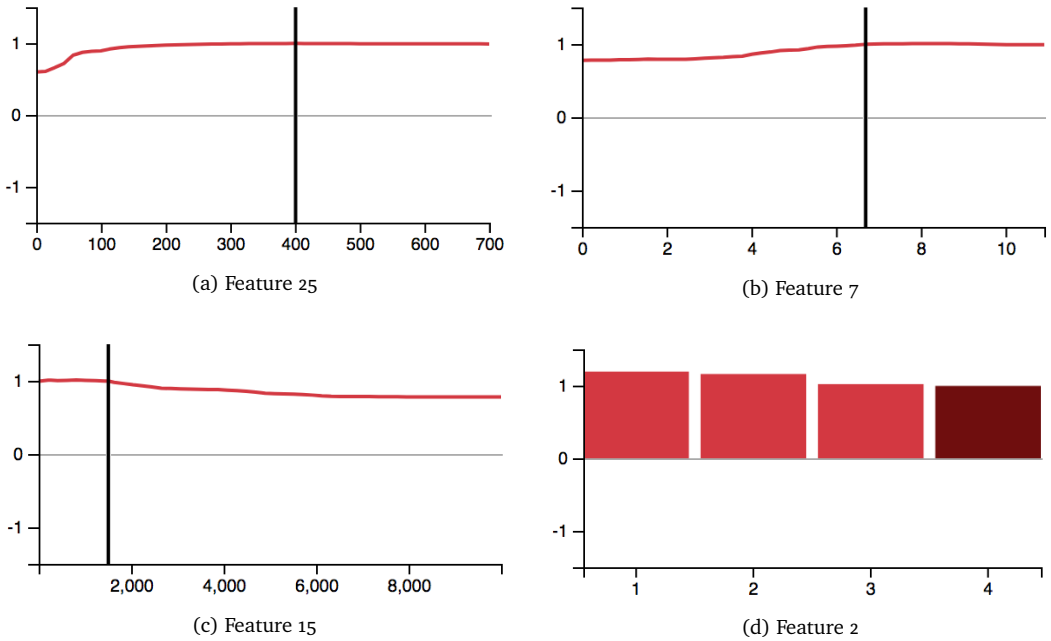
We show the partial dependence of the same two insurance policies as we considered for feature importance: *ANP128* (fraud) in Figure 6.3 and *ANP87* (non-fraud) in Figure 6.4. The PDPs for all features are available in Sections D.1 and D.2 respectively. The features are again ranked according to the variance in partial dependence data.

ANP128 It stands out that the effects in these plots are a bit more salient. The increased variance can be explained by the locality of this plot. By only considering one instance, the averaging effect of global PDPs is diminished, and a sharper image is the result.

These results also seem a lot more intuitive. As we mentioned for the feature *Feature 7* (corresponding to the premium percentage) in the global partial dependence results, the fraud team expects the prediction of fraud to go up for higher values. This is the case for these results: the prediction score steadily increases as the percentage of premium increases.

This also holds for the feature *Feature 25* (corresponding to the average duration of illness). It makes no sense for fraudsters to commit fraud for illness of a very short duration, as it would make them little to no money. The longer the duration, the more can be gained. As such the fraud team expects fraud to be more probable for higher average duration of illness, which is exactly what the plot depicts.

The results also match the local feature importance for *ANP128* much better. The top two variance features also are the top two important features in 6.3a. Likewise, the feature *Feature 2* has high importance (but a negative contribution) and also shows up with high variance. Feature *Feature 15* however slightly deviates, while it is the fourth highest variance PDP, it is only the fourteenth most important feature.



(a) Feature 25

(b) Feature 7

(c) Feature 15

(d) Feature 2

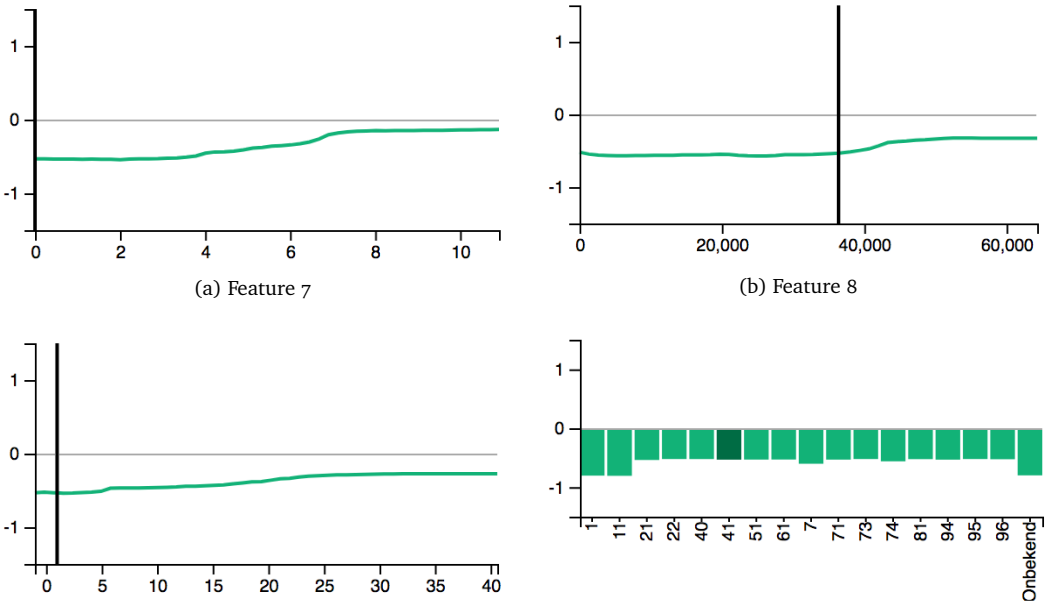
Figure 6.3: Local partial dependence plots for *ANP128*.The three continuous features with highest variance (a,b,c), and highest variance categorical feature (d) are shown.

ANP87 For the non-fraud case *ANP87*, the results are again much more salient than the global approach. This cannot be attributed to the logit scale, as the average prediction score (0.256) is almost as far from 0.5 as the average used during global partial dependence. This gives more confidence that local partial dependence can display effects more effectively, and that global PDPs suffer from averaging for this model.

Just like for *ANP128*, feature *Feature 7* shows intuitive results. *[As a consequence of anonymization, a few examples have been removed.]* Finally, feature *Feature 45* (corresponding to the legal code of a company) shows that especially companies with legal code 1, 11 and *Onbekend* are more likely to be predicted as non-fraud. This codes corresponds with a sole proprietorship, partnership (VOF) or unknown company legal from. These first two forms are used for companies with very few employees, which would mean that fraud for insurance policies like *ANP87* often occurs for larger companies. The PDP for feature *Feature 43* (number of employees) supports this, although the effect is only small.

Contrary to the PDPs for *ANP128*, the ranking of these features does not compare well with the feature importance. Only feature *Feature 7* has the highest variance PDP and is the fourth most important feature. The other three high variance features seem to have a very low feature importance score.

Instance *ANP87* has a very high number of features for which values are imputed (27). Interestingly, it appears that these features all appear on the lower end of the partial dependence variance ranking. This also holds for the results for *ANP128*. This suggests that if a value is missing for a feature, that feature is less likely to be important to the classification.



(a) Feature 7

(b) Feature 8

(c) Feature 43

(d) Feature 45

Figure 6.4: Local partial dependence plots for *ANP87*.The three continuous features with highest variance (a,b,c), and highest variance categorical feature (d) are shown.

6.3 Rule extraction

As we have seen in Section 6.2, PDPs show only very slight effects due to averaging. An alternative way of showing the relationship between features and model outcome (corresponding to Task **T3.4**) is rule extraction. We can use rule extraction to extract atomic rules from the ensemble to show a simplified feature and model outcome relation. These rules contain constraints based on a subset of features. This method can also help to identify relevant combinations of features (Task **T3.3**), as these combinations of features are likely to occur more often together in the extracted decision rules.

6.3.1 Global

In order to extract rules from the Achmea data set we evaluated different rule extraction methods. As mentioned in Section 3.4.1 there is no test data set available for the ensemble model. We are thus not able to properly evaluate a condensed model using any rule extraction method on the data set provided by Achmea.

Evaluation with Pima Indian data set

We identified the Pima data set [102] as a good candidate, as it shares a lot of similarities with the Achmea data set. It is concerned with the prediction of diabetes for Pima Indian women. Like fraud, it is a difficult problem and not possible to predict this class with perfect accuracy. Hence, a typical Random Forest trained on the data set obtains an OOB error of around 30%; very similar to the Achmea model. The data set also reflects a binary classification problem, as the classes concerning diabetes prediction are either *Yes* or *No*. Finally, the data set contains missing values that are imputed, and mostly numerical features, of which a few are integers.

The main differences are that there are only seven features, and that the data set is fairly balanced, with 194 records with ground truth *No* and 106 *Yes*. Also, rather than an ensemble of forests, a single Random Forest is used for classification.

To evaluate how the base model performed, we created 50 samples of 70% training and 30% test data. For each training set a Random Forest was trained with 150 trees, and evaluated on the test set. The mean predictive accuracy of this experiment was 0.73.

InTrees To evaluate the InTrees method by Deng [22] we used the same setup to test- and training data and model creation. After the Random Forest is created the rules are extracted, pruned and selected according to the algorithm. Next, we evaluate the predictive power of the rules on the test set. For the 50 samples, the mean predictive accuracy of the rules was 0.723 with a standard deviation of 0.49. This accuracy is very close to what the original Random Forest was able to achieve. It did take significantly longer to run the algorithm: where the Random Forest took 3.3 seconds, 50 repetitions of InTrees took 42.8 minutes to complete.

DefragTrees Using the same method for training- and test data splits, the more recent method DefragTrees by Hara and Hayashi [43] was also evaluated. Over all samples, the mean predictive accuracy was 0.656; significantly lower compared to InTrees. The standard deviation was very similar: 0.056. Finally, the algorithm is a lot quicker than InTrees, and with default parameters takes exactly 1 minute to run. Additionally it offers an option for parallelization, which can boost performance up to n times depending on the number of threads n supported by the machine.

Table 6.4: Evaluation results for InTrees and DefragTrees, compared to the reference Random Forest.

	Accuracy (μ)	**Accuracy** (σ)	**Running time** (s)	**Model size** (# decisions)
Random Forest	0.734	0.034	3.3	10585.88
InTrees	0.723	0.049	2571.05	5.20
DefragTrees	0.656	0.056	59.7	8.44

We conclude that in order to get an accurate rule representation of the model, InTrees is the most effective algorithm to use. It is able to generalize rules with much higher accuracy than its competitor, at the cost of running time. As we mentioned in Section 3.7, Achmea does not put explicit requirements on the running time of the algorithm, and we thus chose disregard this factor.

The rule extraction mechanisms need to be altered slightly in order to work with an ensemble of Random Forests rather than just one. DefragTrees is not easily adaptable to work with multiple forests. Also, he library is written in Python, which makes it more difficult to work with the Achmea model written in R. InTrees is much easier to adapt to an ensemble of forests. As the algorithm extracts all rules in the first step and only acts on the total set of rules in the subsequent steps, it is easy to just combine the set of rules from every Random Forest. However, combining multiple models leads to long running times.

Optimization

For the Achmea data set, it is not practical to run the standard InTrees algorithm. Running times blow up when the number of rules becomes large, and rough early estimates indicated running times of over 50 hours. The library does offer an option to discard trees as well as an option to only consider trees until a certain depth. However, this would not be a fair representation of the entire model, and could leave out important decisions.

Hence, we decided to parallelize various aspects of the algorithm. The algorithm lends itself well for this: the same procedure is run for every tree for the extract step. For the measure, prune and select step, most procedures are run for every rule. Of these steps, the prune and select step require the most resources.

The extract step is parallelized by running a separate worker for every tree and running the extract and measure step in parallel. As the measure step takes relatively little time, there is no benefit in splitting the work out in a separate worker.

Subsequently, the extracted rules are pruned. This step is parallelized by dividing the set of rules into n partitions, with n the number of available threads on the machine. We also experimented with running a worker for every rule, but the costs of starting up workers quickly overtook the benefit of parallelization.

The rule selection step creates a matrix of rules and training data. This process can be parallelized in a similar fashion as the prune rule step. However, the most significant step for selection is the training of the GRRF. Unfortunately there was no parallel implementation available and facilitating this was not trivial. Hence, the selection step was ran without optimization.

Rule extraction on Achmea model

We ran the InTrees algorithm on the Achmea model using all training data available. Despite all optimization, this process still took considerable time. The extraction step alone took 32 minutes to complete. This yielded 1,328,786 rules with 7,582,365 decisions in total. Intermediately we filtered out duplicated rules. Next, the prune step was ran, which took 4.2 hours to run. During this step contraints are removed from the rules, which can make two different rules identical after pruning. This was the case for 1,178,783 rules, greatly reducing the number of rules used for the selection step.

During the selection step however, we noticed a problem with the current approach. As InTrees relies on evaluating rules on the training data set, it is also sensitive to imbalanced data. The pruning step disregarded all rules that predicted fraud, as fraud is only a tiny portion of the training data set used for evaluation. Hence, the select step only yielded rules predicting *Geen fraude*, which let to a rule set which was unable to identify any fraud case.

The process was repeated with a balanced training data set by heavily undersampling the *Geen fraude* cases. This set contained all 129 *Wel fraude* records, and 129 *Geen fraude* records. As a lot less data is used, the subsequent InTrees steps took less time to run. Selection took 8 minutes, pruning 27 minutes and the selection step 108 minutes to complete. Considering this step took this long despite all optimization, we consider the InTrees optimizations were still necessary in order to run the algorithm.

The final obtained rule set is shown in Appendix E. The rule set contains 41 rules, with a total of 141 decisions. This is a dramatic reduction of the number of rules and decisions compared to the reference model: only 0.003% of rules remain.

We would like to evaluate these obtained rules in terms of predictive accuracy. This is however difficult due to the lack of available test data. Therefore, we evaluate the rules based on their predictive accuracy on the training data. This way we can evaluate if the rules behave similarly to the Random Forest. However, this approach will not be able to capture if the new rules generalize well to new fraud cases.

The results are shown in Table 6.5. Just like our evaluation on the Pima data set, the simplified rule set seems to perform slightly better than the trained Random Forest. We also evaluated a subset of the rules, by filtering out rules with a low GRRF score from the InTrees selection step (*impRF*). When filtering *impRF>0.05* only 12 rules remain, and *impRF>0.1* only 5 rules are left. Impressively, these few rules are able to capture a lot of predictive power from the Random Forest. When reducing to only 5 rules, the accuracy is only 6% lower than the original Random Forest.

Table 6.5: Predictive accuracy of Random Forest and corresponding extracted rules by InTrees.

Model	**Accuracy**	
Random Forest	0.8488	
InTrees rules (original)	0.8682	+0.0077
InTrees rules (impRRF>0.05)	0.8450	-0.0155
InTrees rules (impRRF>0.1)	0.7907	-0.0698

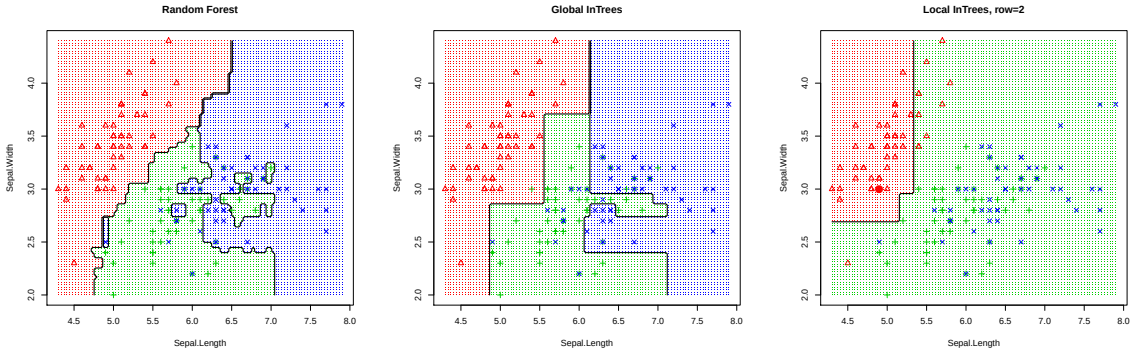
6.3.2 Local

Considering the promising results of global rule extraction, we reckon it should be possible to obtain even smaller rule sets that can accurately describe the behavior of the model on a local scale. This closely follows the insights mentioned by Ribeiro et al. [98].

As an example, consider the decision boundary of the Random Forest in Figure 6.5a. This example uses an extremely simplified data set with only two features from the Iris data set [33], and a Random Forest consisting of 200 trees. In the example the decision boundary is shown separating which region of the feature space is classified as *setosa* (red), *versicolor* (green) and *virginica* (blue). The boundary is very complex and even overfits around a few data points.

In Figure 6.5b a global simplification is applied. In general, the classification behavior of the Random Forest is retained. However, small details are removed for the sake of simplicity. This has an impact on the accuracy of the simplified model. For this example, the simplified model achieves an accuracy of 86%, as compared to 89.3% by the Random Forest.

A different approach is to simplify the model locally. If we only consider the actions the model takes for an instance with $Sepal.Length = 4.9$ and $Sepal.Width = 3$, indicated with a filled red dot in Figure 6.5c, a lot of decisions do not apply. If we only consider the decision rules that are applicable to the reference instance, we capture all decisions that the model made for that instance. If we simplify this subset of rules, we can achieve a much simpler model, that should be accurate for the instance locally. This model will only be a good explanation for the behavior given the reference instance, and would be inaccurate for very different instances.



(a) Random Forest.

(b) Global simplified rules.

(c) Local simplified rules for the instance with *Sepal.Length* = *4.9* and *Sepal.Width* = *3*, indicated with a filled red dot.

Figure 6.5: Decision boundaries of global and local simplified models (b,c) compared to the reference Random Forest model (a).

The evaluation of the validity of these rules is challenging. Global rules can be validated against the original training data from the Random Forest. For local rules we only have a single instance on which we can evaluate the rules. This is not informative as the model can not divide the data into multiple classes. In such case a classifier that classifies any data item as *Fraud* can still be labeled with 100% predictive accuracy. Instead we consider three types of data for the purpose of local rule evaluation: local synthetic data, local real data, or simply use the global training data the reference model was trained with. Next, we elaborate on the local methods, which are visually demonstrated in Figure 6.7.

Local synthetic data Local synthetic data around the reference instance can be used to model the vicinity of the instance. To obtain class labels, we classify these synthetic data points with the original Random Forest. This would give us ground truth that matches the behavior of the model. As we want the simplified model to match the behavior of the Random Forest and not the actual effect in the data, this data can be used for the evaluation of decision rules of a simplified model. This approach is similar to the methodology proposed with CMM by Domingos [29].

A straightforward method to generate such data points is a naive rejection method. This is a Monte Carlo approach where uniform samples are drawn for each feature range individually. Next, the Euclidean distance from the reference instance is checked to see if the synthetic record falls within range. This can be thought of as checking whether these points fall within an n-dimensional hypersphere (n-sphere).

We implemented this method, but found that it is not feasible to use on a high dimensional data set. The ratio between the region of the feature space in- and outside the n-sphere grows significantly as dimensionality increases.

We can show this by calculating the difference between the volume of a unit hypercube and unit n-sphere by

$$1^n \; - \; \frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)} 0.5^n \tag{6.2}$$

where n is the number of dimensions, and Γ the gamma function by Euler (an extension of the factorial function which allows non-integer values). This function evaluated for dimensions between 2 to 41 is shown in Figure 6.6. This example shows that the volume outside the n-sphere approaches 1 already for the 10^{th} dimension. This means that almost 100% of feature space lays outside the n-sphere, rendering Monte Carlo techniques infeasible.

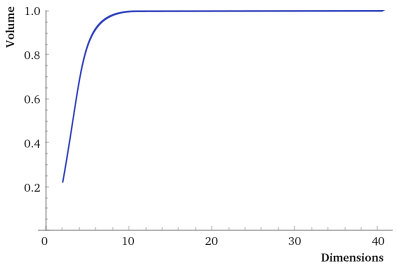


Figure 6.6: Volume outside an n-sphere evaluated for dimensions 2 to 41 (Equation 6.2).

A more efficient way to obtain data points within an n-sphere is described by Harman and Lacko [44]. As Muller [84] shows, if $Y \sim N(0, 1)$, then $S_n = \frac{Y}{\|Y\|}$ (called L^2 *normalization*) has the uniform distribution on the edge of a unit n-sphere. Moreover, by multiplying S_n by $U^{\frac{1}{n}}$, where U has the uniform distribution on the unit interval (0, 1), we obtain the uniform distribution on a unit n-ball [44, 34] (the region enclosed by an n-sphere). Uniform samples from this distribution correspond to points that are uniformly distributed within the n-ball.

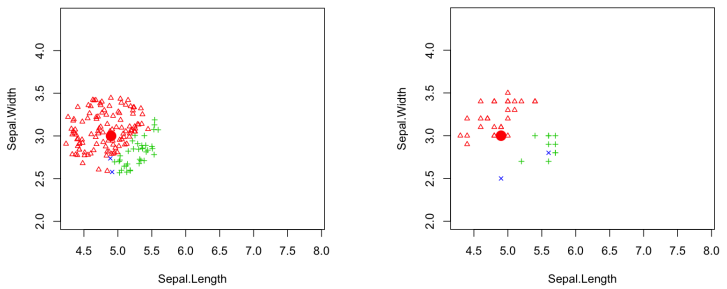
These samples can be used to permute the values of continuous features of the reference instance, by a distance factor δ. Note that this method only works with continuous features. For categorical features, either the instance value can be used, or a arbitrary value from the training data set can be sampled (to preserve distribution in the data). The permuted continuous feature values combined with the sampled categorical feature values yield samples that are relatively close to the instance, establishing locality.

A limitation of this method is that it may yield unrealistic records. First, the approach ignores any relationship between features. For instance, if a feature f_i is always a multiple of feature f_j, then records generated with this method will not show the same relationship. Additionally, if feature values follow a certain distribution, this distribution is not preserved for continuous feature values. For our purposes, this issue is less relevant, as the data is merely used to approach the models decision boundary.

Local real data Rather than synthetic data, we can also use a subset of training data to evaluate rules. This yields realistic data points that do preserve feature relationships and feature distributions.

A way to obtain such data is by sampling nearest neighbors from the training data set around the reference instance. For higher dimensional data, Approximate Nearest Neighbor (ANN) [1] can be used to efficiently derive neighboring data points.

This method depends on the amount of available training data and density of data around the reference instance to be effective. If there are no neighbors near the instance, the data can not be considered local.



(a) Local synthetic data generated with n-ball approach.

(b) Local real data generated with Nearest Neighbor approach.

Figure 6.7: Different methods of obtaining local data for the purpose of evaluation.

The difficulty for both methods is to determine the distance from the instance for which either synthetic data or nearest neighbors are sampled. For the purpose of evaluation, data points are needed that belong to different classes. To obtain items from different classes the distance from the reference instance should be sufficiently large. However, when the distance is too large the data can not be considered local anymore. To support the decision on which distance for the data point generation we should choose, we created a lattice plot of all data, and compared it to the lattice plot of the local evaluation data to see the portion of feature space covered by the data points.

The data points also need to be fairly balanced. This can be achieved by undersampling the majority class, but leaves only few data points for the evaluation. Another alternative is to use a modern oversampling technique like the Synthetic Minority Oversampling Technique (SMOTE) [16]. Along with undersampling, this method creates new records for the minority class. However, these new synthetic instances may not be reliable, depending on the nature of the original data.

InTrees adaptation

We propose the following adaptation of the InTrees algorithm to condense a locally accurate model for a reference instance.

First, the *extract* step should only return rules that are applicable to a certain instance. This can be done by evaluating every gathered rule against the reference instance and discarding it when it is not applicable. This decreases the number of rules significantly. However, for the Achmea data set, we are still left with 50,000 rules that are applicable. We thus need further simplification.

To decrease the number of rules we can run the prune and select step of the InTrees algorithm. As mentioned in Section 4.2.2, these steps in the original InTrees algorithm use the training data used for the Random Forest to evaluate the accuracy of simplified of rules. We refer to this type of data as *pruning data*. For local rule extraction, we can use the three variants of evaluation data we identified earlier as pruning data.

We hypothesize that, when using the Nearest Neighbor approach for pruning data, the rules will be optimized to fit the *data*. This means that locally, the rules are more accurate to the data. The synthetic data on the other hand will optimize the rules to fit the behavior of the *model*.

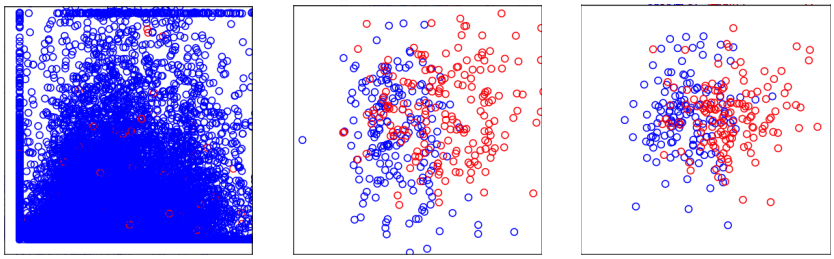
Local rules for Achmea model

We evaluated the quality of the rules for the Achmea model generated with various pruning data techniques. We decided to use the synthetic data approach as evaluation data for the rules. For the purpose of this project, it is most important to find rules that are most true to the behavior of the model.

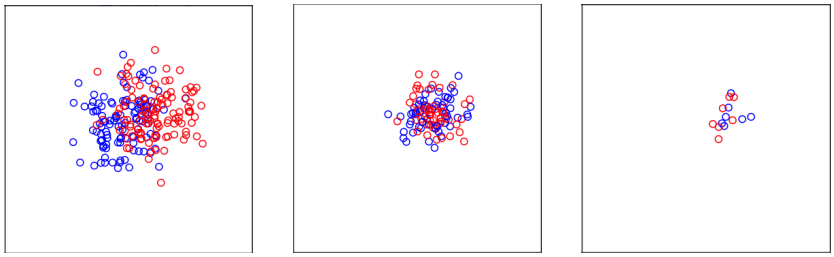
A difficult task is to determine the best factor δ for synthetic data generation. To give more intuition what this δ means, Figure 6.8 shows a two dimensional projection on features *Feature 25* and *Feature 7* for the entire Achmea data set (a), and five synthetically generated evaluation sets for policy *ANP128* with different distances (b,c,d,e,f). The data set is undersampled to be balanced. The example shows that, as the distance factor δ decreases, the data points are centered more locally around the instance.

The delta we pick is depending on the insurance policy used as reference instance. The further the instance is from a decision boundary, the larger the δ needs to be in order to obtain enough data points for both classes. For evaluation, we want a low δ which results in a more local rule set. However, the lower δ we pick, the longer it takes to generate enough feasible data points for evaluation. When δ gets too low, the generated data points are all generated at the same side of the decision boundary, resulting in synthetic data set where all points belong to the same class.

The synthetic data generated with $\delta = 0.75$ for *ANP128* (in Figure 6.8f) only contains fourteen records. This is because the process of obtaining synthetic data takes a long time for $\delta \leq 1$. With $\delta = 0.75$ we generated 4,000,000 data points. The generation of the data points is quite fast, but the Achmea model is so large that the classification of these data points took more than five hours to complete. Of these data points, only seven data points were found that were of class *Geen fraude*, which means that when we balance the data set, we only obtain fourteen data points. For this reason we excluded synthetic data with $\delta < 1$. We also tried generating synthetic data with $\delta = 0.5$, but abandoned that experiment after 12 hours, when still no records were found.



(a) All data (b) Synthetic, δ=2.5 (c) Synthetic, δ=2



(d) Synthetic, δ=1.5 (e) Synthetic, δ=1 (f) Synthetic, δ=0.75

Figure 6.8: Two dimensional projections on features *Feature 25* and *Feature 7* of the entire Achmea data set (a), and five synthetically generated evaluation sets for policy *ANP128* with different distances (b,c,d,e,f). Red circles indicate *Fraud* cases, blue circles indicate *Geen fraude* cases.

We extracted rules for insurance policy *ANP128* from the Achmea model using a variety of different pruning sets. First we used the global original training data from the model. Additionally, we used the nearest neighbor data for 50%, 25%, 12.5% and 6.25% of the entire training set as pruning data. Finally we use the same distances factors mentioned earlier for the evaluation data sets to generate local synthetic pruning data. Next, the obtained rules were evaluated against our four different evaluation sets, of which the results are shown in Table 6.6. The percentages represent the accuracy on the synthetic data set, which we refer to as the *match-with-model* score, as the model would score 100% on any of these evaluation sets.

Table 6.6: Match-with-model score of extracted rules for policy *ANP128*, trained with various pruning data sets (columns) and evaluated on various evaluation data sets (rows).

Test \ Pruning	**Global**	**NN 6.25%**	**NN 12.5%**	**NN 25%**	**NN 50%**	**Synthetic δ=1**	**Synthetic δ=1.5**	**Synthetic δ=2**	**Synthetic δ=2.5**	Average
Synthetic, δ=1	0.804	0.696	0.768	0.679	0.732	0.804	0.982	0.911	0.893	0.808
Synthetic, δ=1.5	0.819	0.633	0.792	0.597	0.761	0.748	0.872	0.889	0.912	0.775
Synthetic, δ=2	0.713	0.596	0.780	0.560	0.674	0.745	0.855	0.879	0.872	0.730
Synthetic, δ=2.5	0.762	0.600	0.776	0.586	0.697	0.714	0.819	0.859	0.868	0.740
Average	0.774	0.631	0.779	0.606	0.716	0.752	0.882	0.885	0.886	

From the different types of pruning data, the Nearest Neighbor data yields rules with the lowest match with the reference model; even lower than the base-case rules pruned with global training data. As mentioned earlier, we hypothesized that this type of pruning data will optimize the rules to fit the data better, and not the model behavior.

Using the Nearest Neighbor local pruning data yields rules that have higher predictive accuracy than the Achmea model. This can be seen in Table 6.7, where we compare the Achmea model with rules extracted with global and Nearest Neighbor pruning data. In this experiment indeed the Nearest Neighbor pruned rules outperform the rules pruned with different pruning data, confirming our hypothesis.

For classification this higher accuracy is a good property. However, our priority is not to obtain highly accurate rules, but rules that explain the behavior of the Achmea model most accurately. The behavior of Nearest Neighbor trained rules deviate much more from the true behavior of the model than rules trained with different training data.

Table 6.7: Predictive accuracy of extracted rules for policy *ANP128* trained with Nearest Neighbor pruning data, compared with the Achmea model and rules pruned with global data.

Test \ Pruning	**Achmea model**	**Global**	**Nearest Neighbor**
NN 50%	0.685	0.803	0.899
NN 25%	0.628	0.791	0.988
NN 12.5%	0.625	0.821	0.911
Average	0.646	0.805	0.933

In these results there is also an interesting trend visible where the match-with-model scores seems to increase as the locality (distance factor δ) decreases. On a local level it is easier to segment fraud from not fraud. This makes intuitive sense, as we only consider a small portion of the decision boundary of the Achmea model. Clearly a subset of the behavior of the model is less complex than the whole, which leads to better decision rules.

We also plotted the number of generated rules for each of the different pruning data set, shown in Table 6.8. We see that both Nearest Neighbor as synthetic pruning data yields fewer rules than using global data, even though these rules are match the model much better than the globally trained rules. Additionally, we show a subset of the obtained rules with an importance score from the GRRF (*impRRF*), obtained during the InTrees select step. The yields a smaller set of rules which often performs very close to the original rule set.

Table 6.8: Number of rules obtained using each of the pruning data sets.

Filter \ Pruning	**Global**	**NN 6.25%**	**NN 12.5%**	**NN 25%**	**NN 50%**	**Synthetic δ=1**	**Synthetic δ=1.5**	**Synthetic δ=2**	**Synthetic δ=2.5**	Average
Original	57	4	29	15	25	18	20	35	19	25
impRRF >0.05	20	2	10	3	10	12	5	15	6	9
impRRF >0.1	12	2	4	3	6	8	2	10	6	6
Average	30	3	14	7	14	13	9	20	10	

These results are by no means a complete and representative analysis, as we only cover rules for one specific insurance policy (*ANP128*). However, our results are an indication that synthetic data used as pruning data yields rules that are most similar to the model behavior.

Next, we present the locally obtained rule sets for the example insurance policies *ANP128* and *ANP87*.

ANP128 The local behavior for this instance can be covered by three rules with two, one and two constraints respectively. The last constraint of the last rule is filtered out, as it had such a small combined rule and feature importance, we considered it irrelevant. This small set of rules is still able to classify 94.1% of the evaluation set correctly. The ratio between the classes *Wel fraude* and *Geen fraude* is equal to the prediction score of the model, in this case 0.881. To create the pruning and evaluation set, we used synthetic data with $\delta = 1$. This was the lowest value for which we could obtain a reasonable amount of pruning data. All local data points on which none of the two rules is applicable, are considered *Geen fraude*.

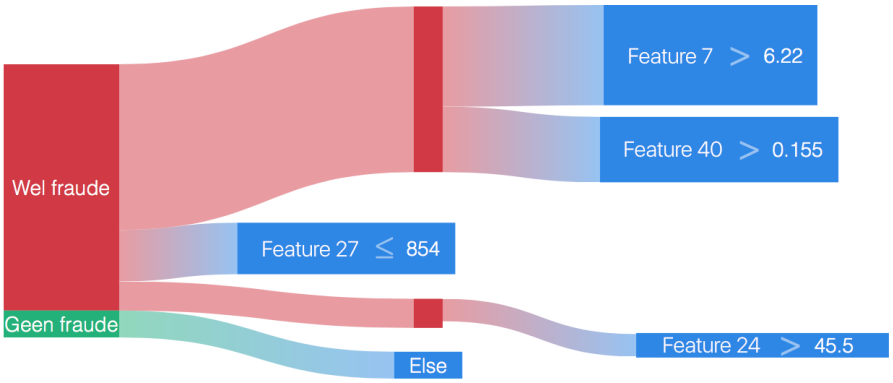


Figure 6.9: Simplified local rules for *ANP128*.

ANP87 The local behavior for this policy can be described by three rules with two, four and one constraints respectively. This set is slightly larger, but classifies 88.7% of the evaluation set correctly. The ratio between classes is different, and as this instance is predicted as *Geen fraude*, the corresponding starting node is larger. This instance is a lot closer to the decision boundary, which means that it was possible to obtain pruning- and evaluation data with $\delta = 0.5$.

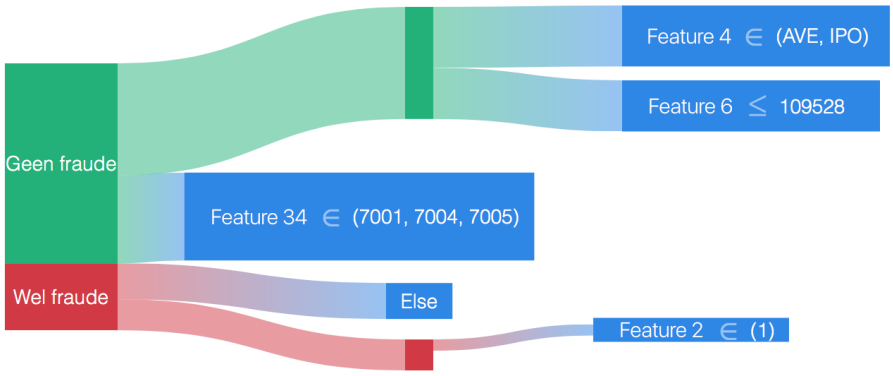


Figure 6.10: Simplified local rules for *ANP87*.

Mismatch with feature contribution

We would expect the features ranked highest with the local feature contribution metric, to also appear in the most important local rules extracted from the model. However, for the two insurance policies tested, this does not seem to be the case. When we want to use both visualizations in the same dashboard, such disparities will not establish trust in the presented explanations (even though such differences can be reasonable). Due to instability of trees, and by extension also Random Forests, a lot of different possible local rules and feature contribution rankings can exist, that are all reasonable explanations.

To make the results of these methods more consistent, we propose to constrain the set of possible selected rules by rule extraction, to a subset which fits the feature contribution metric better. To do this we first introduce a new rule contribution metric to evaluate the importance of a rule R, as

$$RC(R) = \frac{1}{|R|} \sum_{i=1}^{|R|} LI^c_{C_i} \tag{6.3}$$

where $LI^c_{R_i}$ is the local increment of the i^{th} constraint in R (C_i), for class c, and $|R|$ the length of the rule, being the amount of constraints.

During the *select* step of the InTrees algorithm, we can use this rule contribution RC as regularization coefficient for the GRRF. This biases the selection process towards rules with a higher contribution score according to Equation 6.3.

The feature contribution ranking can also be derived from a set of rules in a similar fashion as the calculation for Random Forests. As every constraint in every rule has an associated local increment score, the feature importance for a feature f in all rules can be calculated as the average of local increments for a class c and feature f in the rule set.

Unfortunately our preliminary results showed a non-significant change in produced rules, and we thus decided to not pursue this enhancement on the rule extraction algorithm further. We discuss this issue in more depth in Chapter 9.

6.4 Conclusion

From the analyzed methods, rule extraction has the most potential to be useful. Extracted rules are able to show important features for that instance (Task **T3.2**), convey combinations of features (Task **T3.3**) and also show the relationship between features and model outcome (Task **T3.4**).

Local rule extraction is, to the best of our knowledge, a novel approach. We have shown initial steps towards extracting local rules from a tree ensemble model, and our results show potential. However, there is still a lot of research required to create a stable and reliable approach.

Sensitivity analysis with PDPs could also be a useful method to show the relationship between features and model outcome (Task **T3.4**). However, for some data sets PDPs are not powerful enough to show effects. This was the case for the Achmea data set: due to averaging, almost no effects were visible in the plots.

Feature importance can help identify relevant features in the data set (Task **T3.2**), and feature contribution provides a good indication of which features have a positive or negative impact on the model outcome. The information is limited to features though, and the effect of different values of a feature nor relevant combinations of features are uncovered by this method.

In general, we have seen that local approaches produce more useful results than global approaches. For instance, features can be listed in global feature importance which are not important to a subset of the instances. The global rule extraction did yield a much simpler model, but at the cost of predictive accuracy.

Local approaches are able to capture much more detail, without losing accuracy during the simplification process. Additionally, the local approaches answer the main question by Achmea more directly, by providing information about a specific instance instead.

Chapter 7

Integration

In this chapter we combine the various model analysis techniques covered in Chapter 6 to form a dashboard to be used by the fraud team at Achmea. We incorporate the best visualization techniques that we identified in Chapter 5 to give a consistent insight into the behavior of the model. The locality of the explanation of the created dashboards is key, as the main objective for the fraud team was to obtain instance-level explanations.

Two dashboards were created to give two different perspectives on instances. The first is centered around features, giving a per-feature explanation of its contribution. The second dashboard takes the possible target classes as a starting point, and uses model simplification to present a set of rules that describe the choices the model made for the prediction of those classes.

To showcase all functionalities of the dashboard in an optimal situation, we show explanations for a Random Forest model trained on the Pima data set [102].

Both dashboards have a similar structure and appearance such that they can be used interchangeably. We first describe the elements of the dashboards that are shared. In the following sections we describe the two dashboards in more detail.

7.1 Dashboard home

Both dashboards feature an overview page, shown in Figure 7.1. In Figure 7.1a general information is displayed about the model: the average OOB error of all models, along with a breakdown of the error per target classes. This helps to establish a context about the model performance, which corresponds to Task **T1.1**.

The left sidebar (Figure 7.1b) shows a list of instances of the data that the user can inspect. The instances are sorted according to the model prediction score, so the user can easily identify the most relevant cases to investigate (Task **T2.1**).

The classes concerning diabetes prediction are either *Yes*, shown as a red color, or *No*, shown as a green color. These colors are used throughout both dashboards. The prediction score is explicitly included to convey the certainty of the model about a prediction Task **T2.2**. We chose to show the raw prediction score as a number between 0 and 1 rather than a percentage to stay coherent with the partial dependence plots, which use the same metric.

Once an instance is selected from the sidebar, a page opens that describes the specific instance. At the top of the page the identifier for the case is shown, along with a button to copy this identifier to the clipboard (Figure 7.2a). This functionality emerged from discussions with the fraud team, as they copy and paste the insurance policy identifier regularly to look up information in various systems (Section 3.2).

Below the case identifier a "quick-access" list is shown with certain feature values of the instance (Figure 7.2b). The user is enabled to configure any subset features from the data to be displayed here. This feature was added in response to an early feedback session with the fraud team. To identify the relevance of a case they often consider the amount of salary that is insured. Fraud with

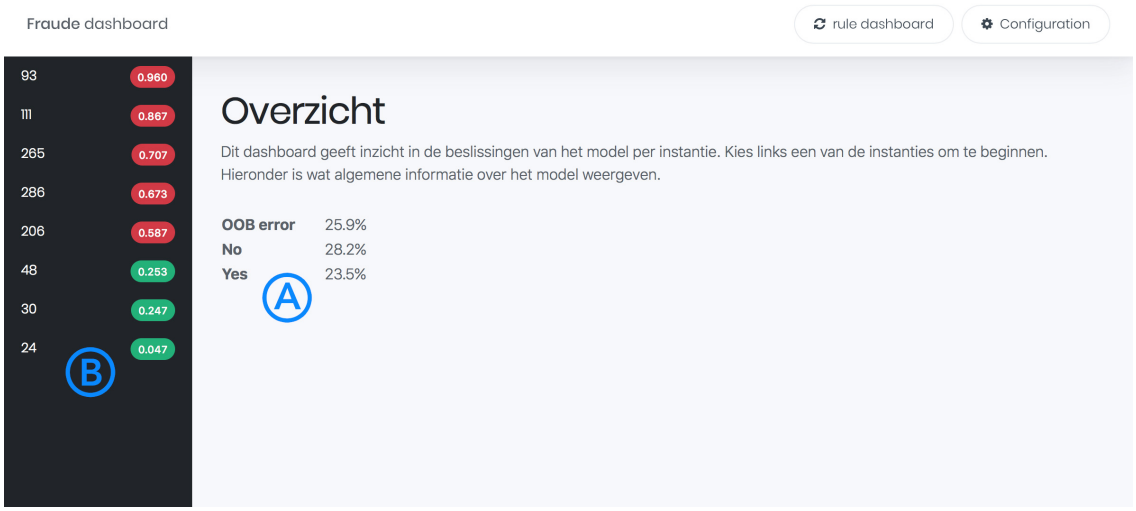


Figure 7.1: Dashboard homepage for the Pima data set.

policies with a lot of insured salary are most relevant to identify. Showing such information at the top of the page helps to identify the relevance of a case more quickly (Task **T2.1**).

A button in the title bar allows the user to switch between dashboards. Switching dashboards keeps the same instance selected. This helps the user to conveniently compare different explanations for that instance.

Both dashboards also feature a configuration button in the title bar with configuration options for the quick access area, as well as settings specific for the dashboard visualizations.

7.2 Feature dashboard

This dashboard uses features as a starting point to create insights about the model. The main element is a table of features along with their values for the selected instance (Task **T3.1**), ranked according to feature importance.

The first column of the table, shown in Figure 7.2c, shows the feature importance metric. This allows the user to identify relevant features (Task **T3.2**). Every method mentioned in Section 5.1 can be used, and configured using the configuration window (shown in Figure 7.3). For the visualization options rank, colored text and colored background, the column is hidden and the feature column is altered to convey the feature importance instead. For the Pima data set, only one Random Forest model is trained, and thus the density plots like box plot and violin plots have no significance. We chose to show a bar plot such that small relative differences are salient.

For this instance, there are three features *age*, *skin* and *bmi* that have a positive impact on the decision. The *glu* feature has a negative importance, and leads the model to believe this case is less likely to be a diabetes case, compared to when the feature was not considered for prediction.

Next, the fourth column of the table (Figure 7.2d) shows a combined histogram of the data density distribution given the feature range. This helps to compare feature values of this instance with the entire feature space, and identify whether this value is common or an outlier (Task **T3.5**). A distribution is shown for every class, and is color coded accordingly. Where the distributions overlap, the colors are blended together. Optionally the amounts can be normalized to deal with heavily imbalanced data sets. The x-axis scale is defined using Equation 6.1. The feature value for the selected instance is indicated with a vertical line. This links the mostly local information presented in this view with a global insight.

Figure 7.2c shows that the features that have a positive importance for the diabetes prediction *Yes*, have also values are mostly present in women with diabetes (red) in the training data set. For

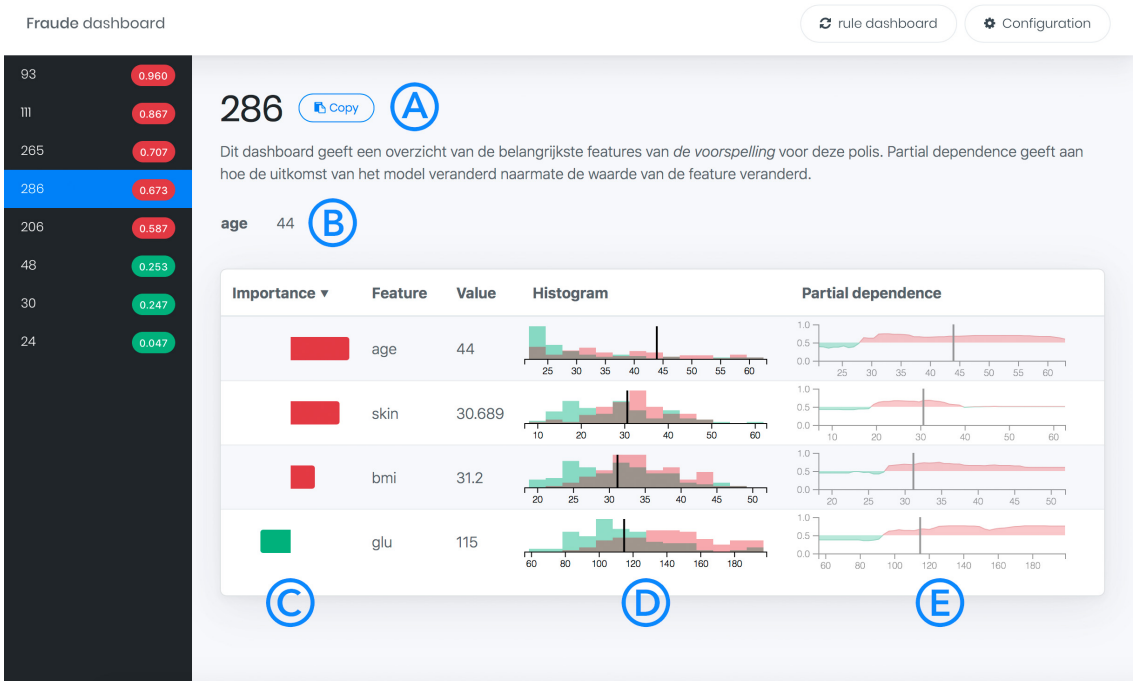


Figure 7.2: Screenshot of the feature dashboard for the Pima data set.

the *age* feature most women between 40 and 45 have diabetes, which supports the claim that the feature *age* is important. Likewise, the *glu* feature has a negative impact for this instance, and has a value that is often present for women without diabetes (green).

Finally, the last column (Figure 7.2e) shows a PDP plot, as described in Section 5.2. Using this plot the user can identify the relationship between feature values and the model outcome (Task **T3.4**). The same y-axis scale is used as in the histogram plot, and the current value is again shown by a vertical line. The partial dependence can be plotted as either an area- or line chart. Users are enabled to chance their preference in the configuration modal (Figure 7.3). As the number of features can be quite large, we saved space by keeping the plots small. Because of this it may be difficult to spot small changes in partial dependence. We chose to use an area chart for the PDP for optimal visibility.

The histograms show that, for this data set, it is possible to separate most diabetes cases from the non-diabetes cases, as the distribution for *Yes* and *No* differ significantly. For instance, most *skin* feature values above 23 belong to the *Yes* class. We can see that the model learned this distinction, as the PDP curve for the *skin* feature increases around $skin = 20$, and switches the prediction for this case from *No* to *Yes*.

7.2.1 Configuration

Various aspects of the dashboard can be configured using the configuration modal, shown in Figure 7.3. A few settings were already mentioned: it is possible to select a subset of features for quick access on the top of the view, switch between feature importance and partial dependence visualization techniques, and to toggle histogram class normalization.

Additionally, the configuration allows the user to show or hide any of the columns of the tables when desired.

With the feature threshold option, a range can be specified to automatically filter out features with low feature importance.

As we have shown in Section 6.2, the signal in the PDPs can be difficult to discern. To provide the most salient effect, we implemented different ways of presenting the partial dependence data. First of all we can dynamically switch between logit scale (which should magnify strong effects) and

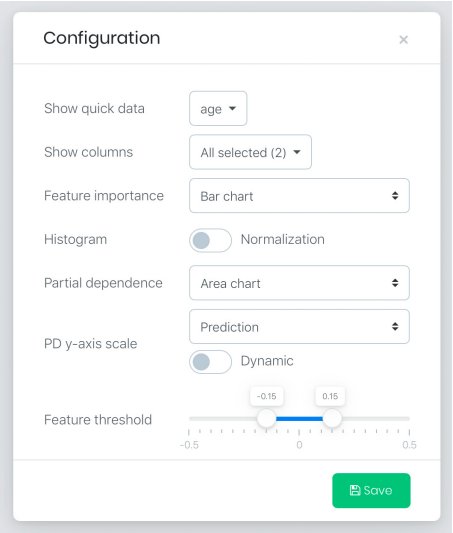


Figure 7.3: Screenshot of the feature dashboard configuration modal.

Figure 7.4: Dynamic PDP y-axis scaling. As the bottom plot lies entirely above the threshold to switch prediction class (at $y = 0$), the y-axis scale is reduced to $[0, 1.5]$ rather than $[-1.5, 1.5]$.

original prediction scale. In Section 5.2 we argued that the logit scale could be confusing to the user, hence a prediction score is also supported. We reckon the prediction score scale should be easier to interpret by the user.

Secondly, for many instances the partial dependence signal does not cross the turning point where the prediction switches from one class to another. This means that the chart lies fully above or below the threshold at $y = 0$ (for logit scale). We implemented a dynamic y-axis scale that, in such a case, only shows the relevant range above or below the threshold. An example is shown in Figure 7.4.

7.2.2 Interaction

To further improve the partial dependence saliency, the histogram or PDP can be clicked to zoom in the charts for that feature. Multiple features can be zoomed to the histograms and PDPs can be compared.

Moving the cursor over the PDP reveals a tooltip that follows the chart line, and shows the feature value, the logit and prediction score at that point in the chart.

A tooltip is also implemented for the feature importance plots, and shows the raw, non-normalized feature importance values. This allows for comparison of the feature importance across instances.

Finally every row of the table (with the exception of the histogram columns) can be sorted either ascending of descending, by clicking the table header. Clicking the partial dependence column sorts the rows based on the variance of partial dependence effect in the plot.

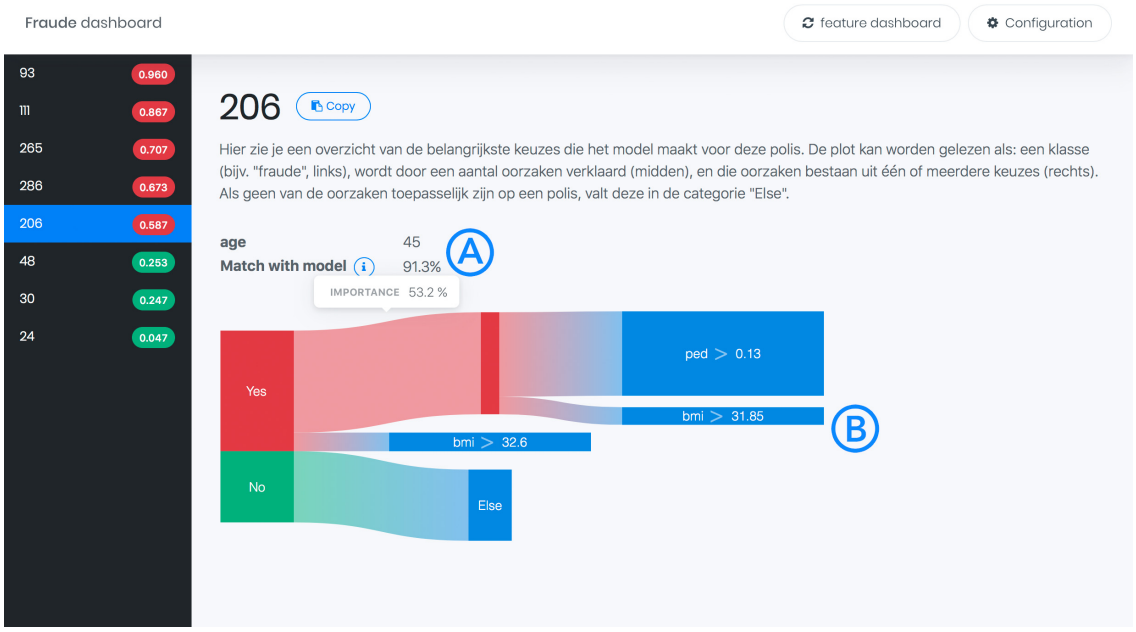


Figure 7.5: Screenshot of the rule dashboard for the Pima data set.

7.3 Rule dashboard

This dashboard takes the possible target classes as a starting point, and uses model simplification to present a set of rules that describe the choices the model made for the prediction of those classes.

In the quick access area an item is added that shows how well the simplified set of rules matches the behavior of the model on a local scale (Figure 7.5a). This indication was added to convey that some detail of the model behavior has been lost to obtain such a simple visualization.

The set of rules is visualized using either a 1D range plot or Sankey diagram, which we introduced in Section 5.3. Which method is shown in the dashboard can be customized in the configuration modal. The multi-dimensional plot was left out, as it is limited to rules with less than four constraints.

Figure 7.5b shows that locally, 58.7% of the explanation are rules predicting class *Yes*. This corresponds with the class prediction for this instance. 53.2% of all evaluation cases (revealed in a tooltip when hovering over a link in the diagram) were classified by the top rule, with two constraints on the *ped* and *bmi* features. The width of links from the rule to the constraints is proportional to their feature importance scores. In this case feature *ped* has a lot more importance, and is thus shown at the top, and has a wider link. If evaluation cases were not matched by one of the two shown rules, they apply to the "Else" category, which is responsible for 37.1% of the evaluation data. This particular set of rules predicts 91.3% of the evaluation data set the same as the reference Random Forest did, and thus Figure 7.5a shows this percentage to indicate the match between simplified and reference model.

To get more information about a certain constraint, it can be clicked on in any rule visualization to reveal a feature sidebar (shown in Figure 7.6). This sidebar shows a 1D range plot (Figure 7.6a) of the constraint on the feature space, the histogram of the models training data (Figure 7.6b), as well as a partial dependence plot (Figure 7.6c). We ensured that the axis of every plot is identical so the values can be easily compared.

In Figure 7.6 the constraint $bmi > 32.6$ is selected. This constraint seems to perform well on the global data, because in the captured range, the histogram shows predominantly instances of class *Yes* (red).

We can put the behavior of this simple rule into context with the behavior of the entire model by comparing it with the partial dependence plot. This shows that a lot of detail from the original

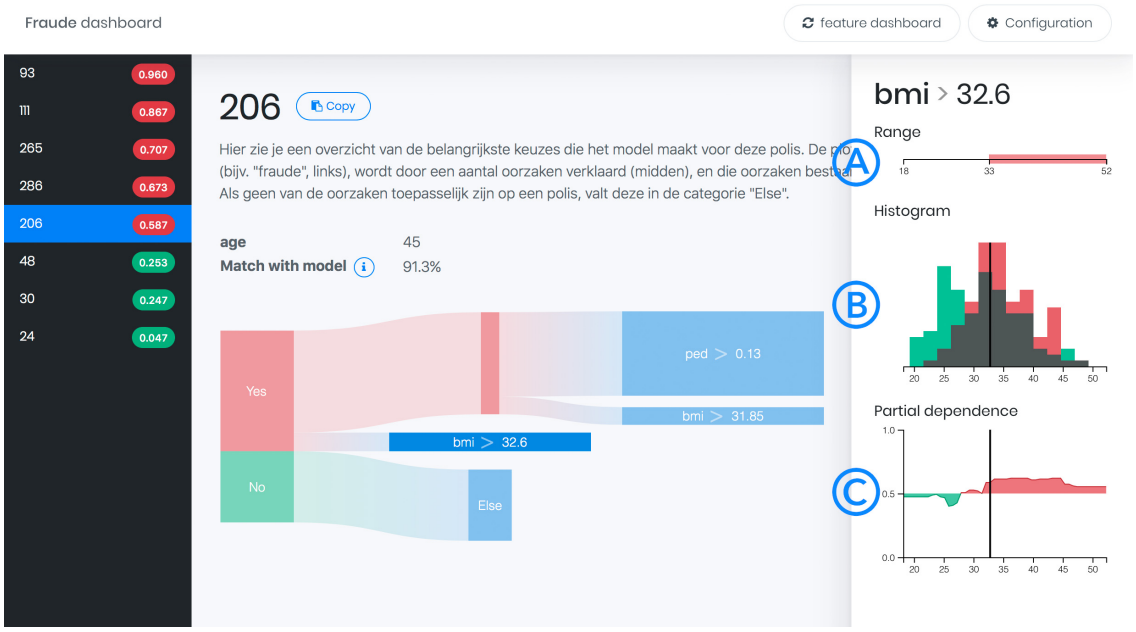


Figure 7.6: Screenshot of the rule dashboard for the Pima data set, showing the feature sidebar for the selected constraint $bmi > 32.6$.

model is removed. After $mbi > 45$ the PDP shows the model is less certain of diabetes for this patient. Another region $25 < bmi < 28$ is visible where the model is particularly certain of no diabetes. These details are all lost during simplification, contributing to the lower than 100% match-with-model percentage.

7.3.1 Configuration

Various aspects of the dashboard can be configured using the configuration modal, shown in Figure 7.7. The setting to choose between rule visualization methods was briefly mentioned before. This configuration modal also includes settings that are identical to the feature dashboard. Quick access, partial dependence and histogram related settings are shared. In addition, the values are shared across dashboards, so the preference for histograms and partial dependence plots is reflected in both dashboards.

In addition, two threshold sliders are provided in order to filter out rules and constraints according to their importance. First, a rule importance threshold defines below which value low importance rules are left out. Second, the feature importance threshold filters out constraints from rules that have a low feature importance. The filter is applied to the rule importance multiplied by the feature importance of the constraint feature. This means that constraints are less likely to be filtered from rules with high rule importance.

7.3.2 Interaction

This dashboard also features interactivity. The most important interaction was already discussed: when clicking on a constraint a sidebar appears that shows more information about that feature. This is indicated by a pointer cursor for clickable items. When a constraint is clicked, the rest of the visualization fades out, drawing more attention to the clicked constraint. This effect is shown in Figure 7.6. Clicking another constraint reloads the content of the sidebar, and clicking anywhere else on the page hides it.

Next, a few interactions were implemented specific for the Sankey diagram visualization. The

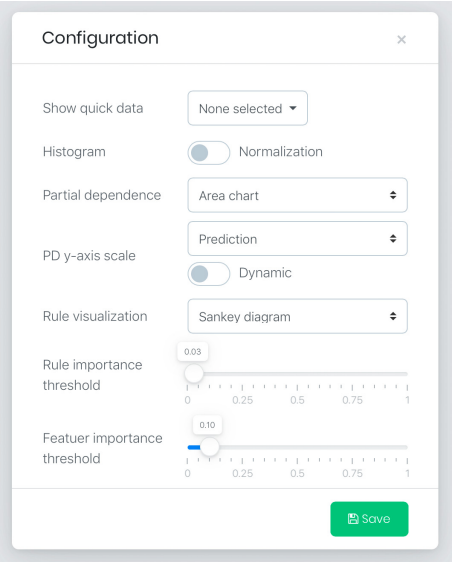


Figure 7.7: Screenshot of the rule dashboard configuration modal.

nodes of the diagram can be dragged and rearranged vertically. This enables users to group together constraints that they deem useful to investigate. This is particularly useful for large sets of rules and constraints.

Furthermore, when hovering over Sankey diagram links, a tooltip shows the importance of that rule or constraint as a percentage. This helps to pinpoint the exact importance, which can be difficult to derive from the proportions in the diagram alone.

Chapter 8

Evaluation

In this chapter we evaluate the final dashboards created in the previous chapter. We first show the dashboards for the Achmea data and model and draw conclusions on the effectiveness of the system. Next, we conducted a user test with the fraud team in order to evaluate if the dashboards meet their expectations and needs.

8.1 Observations

We evaluate how effective the dashboards are when populated with the data from the Achmea data set and model. The Pima data set used in Chapter 7 shows the optimal situation for the dashboard. In contrast, the Achmea data set and model are a lot more complex (as described in Chapter 3). This can cause the dashboard to be less effective.

We start by considering the feature dashboard, shown in Figure 8.1. Note that various features are filtered out by the feature importance threshold setting.

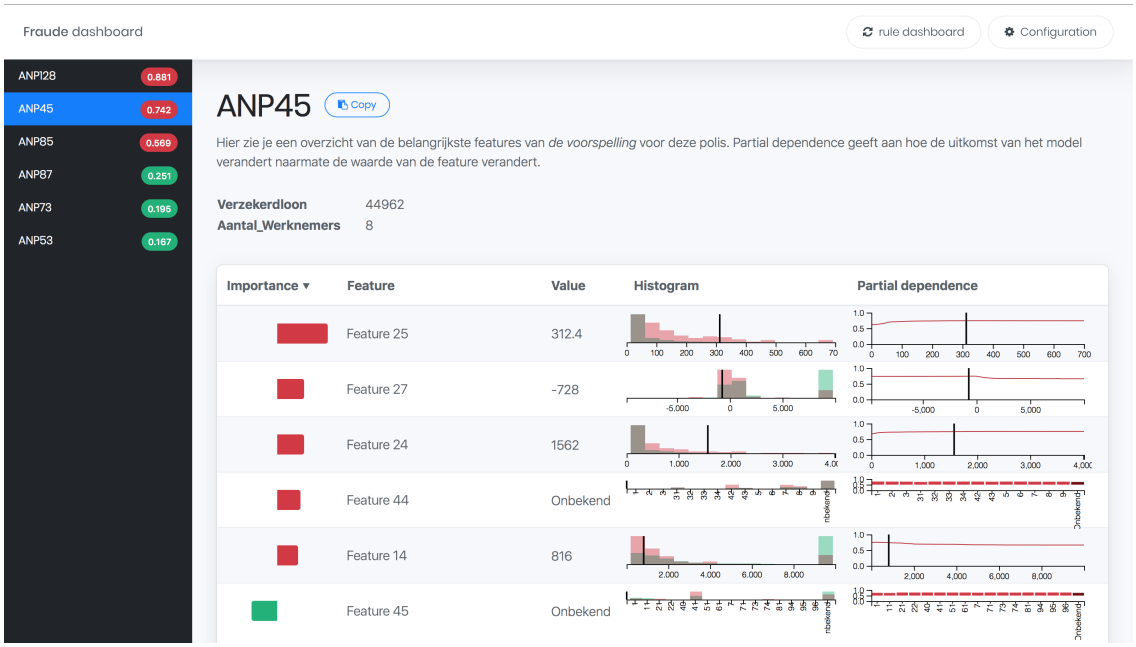
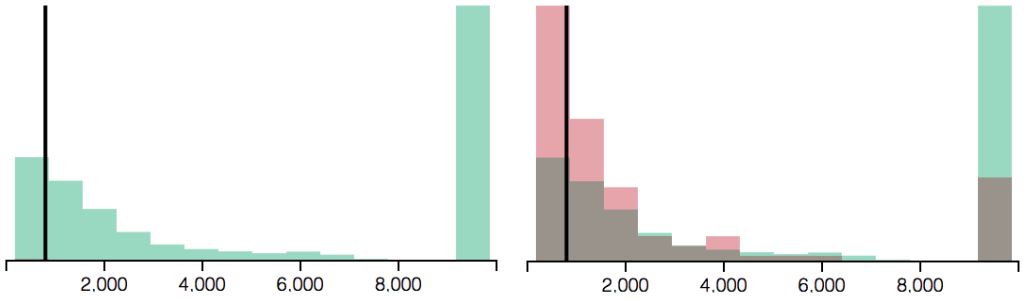


Figure 8.1: Screenshot of the feature dashboard for the Achmea data set.

The first difference with the dashboard for the Pima data set, is the imbalance of the data set. This has such a significant impact on the histograms, that no *Wel fraude* cases are visible in the plot. This is shown in Figure 8.2a.

In the example in Figure 8.2b, histogram normalization was enabled. This enables the user to compare the densities of classes, but can be misleading as the density of fraud cases is scaled up a lot. The example histogram in Figure 8.2b could convey the insight that instances with low values of the feature are predominantly *Wel fraude* cases. However, even if there are *relatively* more fraud cases found in this range, there are still significantly more non-fraud cases with this value than fraud cases (as Figure 8.2b conveys).



(a) Normalization disabled.

(b) Normalization enabled.

Figure 8.2: Difference for histogram normalization setting for the *Feature 14* feature.

A second data set characteristic that has an impact on the visualization is the imputation of missing values. Even though the values of the Pima data set were also imputed, the imputed values were not chosen as extreme as they were in the Achmea data set. As shown in the example in Figure 8.1, the histograms for features *Feature 27* and *Feature 14* show a large peak at 9999. This is because the majority of data points in the training data had an imputed value for these features. This extends the range of the feature to large and unrealistic values, and makes the histogram convey less information. We mention more about this in Section 9.4. The dashboard does show that non-fraud cases have relatively more imputed values.

The imputation also has an effect on the decision that the model makes. For instance, in Figure 8.3 a decision *Feature 27* $\leq$ *854* is shown. As the fraud team noted during the evaluation session, this value is completely unreasonable when considering what the feature means. As such a large portion of the training data has missing values, and the difference with the real values is so large, the decision of the model is often made to distinguish imputed values of 9999 with any other value. Note that a rule distinguishing imputed values from real data can indeed be very informative, however, the yielded rules can be confusing for the user.

Another difference with the Pima dashboard is shown in the example of Figure 8.1: the very slight effect visible in the PDPs. This was discussed in more detail in Section 6.2. Contrary to the PDPs for the Pima model in Chapter 7, which allow the user to spot a tipping point between classification of classes, these plots only show a barely noticeable difference in prediction score.

Finally, a last limitation we encountered preparing the dashboards for the Achmea model is that extracting rules from the vastly complex Achmea model takes unreasonably long time to complete. We traced back this efficiency problem to the running time of classification of our synthetic data, which can take hours to complete for a single instance. Simplifying the model by choosing fewer trees (which is feasible, as we show in Section 3.4.2), or using fewer Random Forest could significantly improve the running time. We mention more about this in Section 9.4.

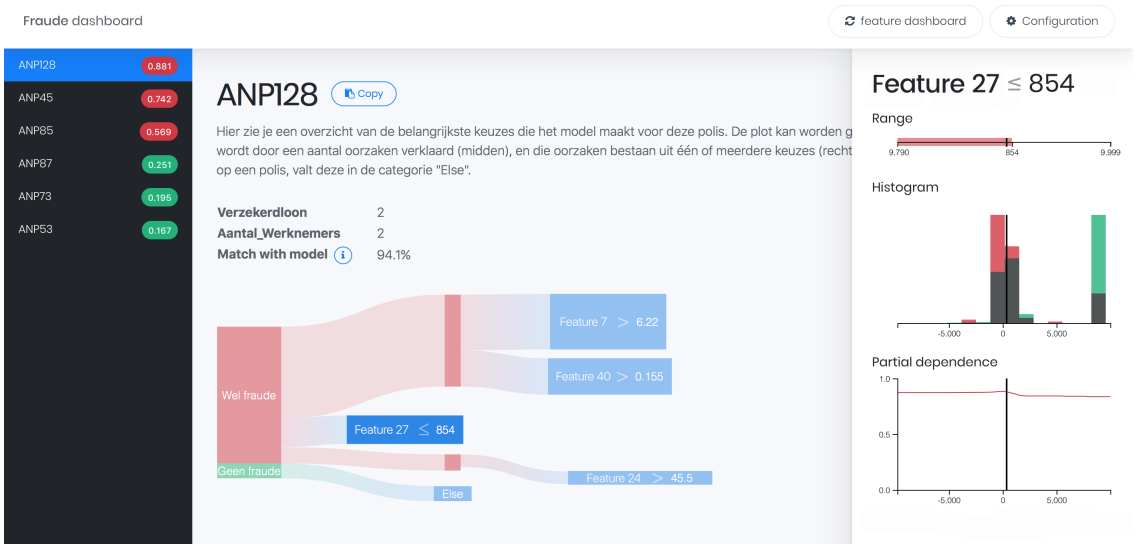


Figure 8.3: Screenshot of the rule dashboard for the Achmea data set, showing the feature sidebar for the selected constraint *Feature 27 ≤ 854*

8.2 User testing

In order to evaluate the usability of the system for the intended users, we organized a two hour evaluation session in which two members of the Achmea fraud team participated. The session was structured as follows:

Exploration phase Without any introduction, we let the fraud team discover the dashboards on their own. This gives us an impression whether certain elements are easy to find and understand, and whether concepts are interpreted correctly.

Use case phase We explain the model in detail. Next, we ask the team to complete the user tasks mentioned in Section 3.6, and to identify if and why a particular insurance case is considered fraud by using the dashboard. This phase helps us to determine whether the dashboards indeed help the fraud team to understand the decisions the model made.

Preferences phase We enumerate all possible visualization options in the configuration modal, and ask which preferences the fraud team prefers, and why.

Evaluation phase We evaluate the usability of both dashboards by asking the team to fill in a System Usability Scale (SUS) survey [13]. We also ask some concluding questions about their preference for visualization techniques and dashboards.

8.2.1 Exploration phase

We started the session by letting the fraud team explore the dashboards on their own, without any introduction or explanation. For each visualization technique, we mention which elements were clear, as well as misinterpretations.

General The team immediately started focusing on the table with features, without reading the explanation above it, or considering the sidebar with insurance policies. It was not immediately clear that the explanation concerned a single instance, but this became clear as soon as they noticed that the feature values corresponded with the value lines in the histogram.

They went over the dashboard row by row, and analyzed what each feature meant by using the visualizations.

The team focused a lot on the meaning of the features within the presented data. As the Achmea analytics team created the data set, they are not familiar with the format of the data like feature names and units of measurement. We were not able to include such information, as it was not included with the Achmea data set. The meaning of features was clear after some discussion among the team. They also questioned the range of some features. For instance: they noticed the feature *Feature 24* has some unreasonably high values. These are the result of human error in data collection, as well as value imputation. We cannot improve these issues, as the Achmea data set was considered fixed for this project.

Feature importance The feature importance chart was taken for granted from the start. They considered it logical that the feature at the top was most important. They also did not question the order of the features. During this phase the fraud team did not figure out that all columns could be sorted according to value, like for instance name. However, the default sort on importance was probably a good choice.

Histogram When focusing on the histograms, they considered the grey color (the result when the two class distributions overlap) as a separate distribution. When we corrected their interpretation, it was clear and they did not confuse the colors again during the session.

It was clear that the histograms were well understood when they used these for deducing the meaning of the feature *Feature 45* (corresponding to the legal code of a company). The fraud team has good knowledge of what the most common values for certain values are. Looking at the histogram information, they deduced that the values 1 and 41 for feature *Feature 45* corresponded with sole proprietorship and Besloten Vennootschap (BV) companies, as they knew that these are the most often occurring company legal forms. As we mentioned, this information was not available in the data set, and as the dashboards are meant to be applicable to any Random Forest, we did not include any functionality specific for the Achmea data set.

By looking at the histograms they also observed that fraud cases typically have higher values of the features *Feature 7* and *Feature 8* (corresponding with high premium percentage and high insured salary). This matched what they expected, as fraudsters typically try to maximize their profits. This shows that the concept of the different distributions in the histogram was clear, and they could use these to create insights about fraud cases.

Partial dependence The fraud team disregarded the PDPs at first. When asked to explain what these plots mean, they expressed they had difficulties understanding what information the plot conveyed. They did notice that the y-axis corresponded with the feature range, and the vertical line indicated the instance value for this feature. Their initial guesses included "the weight of importance of feature values" and "which values are considered normal". Eventually they linked the information with the prediction by the model, but were very hesitant about their decision.

Rule visualization The first thing they noticed was that the ratio between the explanation for classes *Wel fraude* and *Geen fraude* corresponded with the model prediction. They did however confuse the *match-with-model* percentage with the percentage of prediction of fraud by the model.

Very early on, the experts explained: "There is a significantly higher chance of fraud because of these features", "the features that score the highest are *Feature 7* and *Feature 27*" and "When the case has a minimum difference between illness and start of work of 854, which is the case for this instance, then there is an increased risk of fraud". This is exactly what we want to convey by means of the Sankey diagram.

However, they had difficulties with understanding the value for the choice *Feature 27 ≤ 854*. They did not believe that the number represented amount of days (which it does), as it was exceptionally high and did not match what they expected. This can be attributed to the data imputation, as missing values are filled in with the value 9999. This model decision seems to separate the missing values from actual values for this feature. This would be great feedback to the Analytics team, such that they can improve the data imputation to more realistic values.

When opening up the feature sidebar, they started to confuse the value of the current instance with the decision point of the rule constraint, as the instance value for the feature is not explicitly stated. This should be added to make the sidebar more self-explanatory.

8.2.2 Use case phase

We continued by explaining the dashboard in detail. We explained the three main visualization techniques and mentioned aspects they did not notice during the exploration phase, like the overview page summarizing the model error, copy button for the insurance policy identifier, quick access at the top and negative feature importance scores.

Next, we asked the team to complete the user tasks mentioned in Section 3.6, and to identify if and why a particular insurance case is considered fraud by using the dashboard. This phase helps us to determine whether the dashboards indeed help the fraud team to understand the decisions the model made.

User tasks

In Section 3.6 we presented a set of user tasks that our solution should accommodate. When these tasks are completed, the fraud team should be able to answer their main question: "what are the choices the model makes for fraud detection of a specific case?".

T1.1 ***Present* accuracy of the model globally.**
The team immediately navigated to the overview page, showing the error of the model. During the session they expressed that showing the global feature importance on this page would be helpful, although they agreed that it can be difficult for new users to see the difference between the overview explanation and instance-level explanations.

T2.1 ***Lookup* relevance of case.**
They initially responded by considering the insurance policies with a red badge in the cases list. After some discussion they changed their answer that they would only consider cases with a prediction score of above 80%. This way the chance of a case that is predicted as fraud actually being a lead is higher.

When mentioning the quick access data, the fraud team responded with moderation. They started discussing which features could be useful to determine the relevance, but argued that none of the features would help them much. They thus would initially only use the model prediction to determine the relevance of a case.

T2.2 ***Lookup* certainty of the model prediction for that case.**
They again pointed to the prediction score listed in the cases list. They found this fairly straightforward. The team mentioned that it would be more clear to show percentages instead of a number between 0 and 1.

T3.1 ***Summarize* feature values.**
To see the feature values they pointed towards the value column of the feature dashboard table. An issue we noticed here is that features that are considered unimportant for the decision of the model are filtered from the table. The value of filtered features could still be practical information for the team to know; this is why the quick access list was implemented. However, they did not mention the quick access list for this task.

T3.2 ***Identify*** **relevant features.**
They mention the feature importance column in the table for this task. When asked if negative importance features would also be relevant, they agreed, saying these features could indicate false-positives. They did not mention the partial dependence plot, nor the rule visualization for this task.

T3.3 ***Identify*** **combinations of features that play a role (feature interaction).**
They switched to the rule dashboard for this task, and pointed towards two constraints within the same rule. This was surprisingly logical for them, and they seemed to have a clear idea that the two different rules are not a relevant combination, but the constraints within a rule are. Their interpretation matches what we defined as feature interactions.

They also agreed that this can be used to extract new patterns from the data, as described by the second main question by Achmea mentioned in Section 3.5.

T3.4 ***Identify*** **relationship between feature values and model outcome.**
In their use with the system, the team had difficulty with understanding the PDPs. The confusion persisted, even after it was explained. Apart from that, we reckon the effect shown in the plot was too small for them to draw conclusions. This visualization was thus not as effective at conveying the relationship between feature values and model outcome as we hoped.

The rule visualization was a lot more effective at this task, as they clearly articulated why the model made a decision based on which feature values.

T3.5 ***Compare*** **with different cases.**
For this task, the team mentioned that it is not possible to see two cases side-by-side. They stated they are used to improved solutions by making print screens of interfaces, as other systems they use also do not support comparing multiple cases.

By switching between multiple insurance policies, cases can still be compared, albeit not as effectively.

Real scenario

Next, we asked the team to describe if a particular insurance policy listed in the dashboard is fraud, and why they think so. During the meeting, they were especially interested by the explanation for insurance policy *ANP45*. This case is predicted as being fraudulent (with a score of 0.742), but has a rule explanation that shows only indicators for no fraud. If none of these rules match, the case is considered fraud.

The fraud team argues they should investigate this case, as it might be fraud. They base this on the two constraints related to the difference between the date when the illness started and registration of the illness by the employer. They mention they know from experience that these times often differ for fraud cases. A fair amount of fraud happens in cases where employers try to include a sick employee before the insurance is active. Such illness is not covered by the insurance, but by reporting a different occurrence date, fraudsters can still claim compensation. Instead of immediately reporting illness as soon as the policy is active (which would be suspicious), fraudsters often wait a few weeks, and report the illness retroactively. Based on the explanation, they reckon this might be the case for this insurance.

The team did not consider the other dashboard for *ANP45*, and thus did not notice that the importance of features varies between dashboards. This is a issue we elaborate on more in Chapter 9. We reckon the team would less convinced when comparing the results of both dashboards.

All in all, the evaluation shows that all the user tasks are facilitated by the dashboards. Additionally, the team is enabled to obtain insights about the decisions of the model, and use these insights as a basis for further fraud investigation.

8.2.3 Preferences phase

We explained all options present in the configuration modal. The team largely agreed with our presets.

They were intrigued by the amount of options for feature importance. However, the rank, colored text and colored background visualizations were not detailed enough to be useful. The plots where the density is displayed (e.g., box- or violin plot) were too complicated for their liking. Whether different parts of the model agree or disagree could be useful for an analyst, but for their job the average importance of features is sufficient. Hence, they preferred the bar plot.

Most notably, the team disliked the area chart for PDPs, as it falsely let them to believe that the volume of the chart was meaningful. They also did not prefer the dynamic scaling, as it would make the PDPs even more difficult to understand. For the same reason, they disliked the logit scale and preferred to see the actual prediction plotted in the PDP.

Finally, we showed them the 1D range plot as an alternative to the Sankey diagram. For this setting the fraud team was quick to respond: "this visualization is not pleasant". They found the range plot more difficult to interpret than a textual representation of the constraints. Additionally, they are fond of the flow of importance scores in the Sankey diagram, which are not present in the 1D range plot.

Finally, we mentioned the thresholds to filter out irrelevant information. They liked this functionality, but argued they needed to use the system for a while to be able to determine the right values.

8.2.4 Evaluation phase

To evaluate the usability in a quantifiable manner, we asked the fraud team to complete a System Usability Scale (SUS) survey [13], one for both dashboards. It contains ten questions which are answered on a Likert scale [71]. It is a standardized tool that is widely used. A meta study analyzed ten years worth of data and concluded that it is a highly robust and versatile tool for usability professionals [3]. The exact survey used is attached in Appendix F. The results for our dashboards are shown below.

Table 8.1: SUS survey results. Score range between 0 and 100.

	Feature dashboard	**Rule dashboard**	Average
Participant 1	75	92.5	83.75
Participant 2	95	100	97.5
Average	85	96.3	90.3

On average, both dashboards combined achieve SUS score of 90.3. According to Bangor et al. [3], this score is considered *Excellent*, out of the categories *Awful*, *Poor*, *Ok*, *Good* and *Excellent*.

When considering their preference for each dashboard individually, the team initially expressed they did not want to choose between dashboards, but would like both dashboards to be available for use in practice. If they *had* to choose, they prefer the rule dashboard for usage on a day-to-day basis, as it provides the quickest insight on which aspect they can base their further investigation. The team did mention that the feature dashboard conveys more information and be used for more in depth analysis. They praised that the dashboards would be usable for a wide audience, ranging from a novice to an analytics professional.

This preferences is reflected in their SUS scores, as the rule dashboard consistently scores higher than the feature dashboard. According to Bangor et al. [3], the feature dashboard is considered *Good* to *Excellent*, whereas the rule dashboard is considered *Excellent*.

The resulting scores from the SUS survey are very positive. We reckon this can be explained as the fraud team compared the system with the current systems they use, instead of with the best imaginable system. As they are not very satisfied with the current systems, the rating of our dashboards are higher than expected.

Chapter 9

Conclusions

In this chapter we provide conclusion on the research as presented in this thesis. First, the achievements and advantages of our dashboard are presented. Next, we discuss limitations of the current concept dashboards and research. Finally, recommendations for Achmea, research and future work are given.

9.1 Achievements

In this thesis we discussed the problem of Random Forest visualization. We defined the goal "How can we make the instance-level decision making process of a Random Forest comprehensible?".

To this end, we showed a comprehensive overview of current interpretation methods. We analyzed the effectiveness of these methods and various visualization techniques in order to effectively convey the information.

We found that global interpretation methods are insufficient for showing the behavior of very complex models. These methods take the mean response of various local effects, which can obscure the real effect. We showed that local interpretation methods can be much more effective and loose less accuracy in the process. Additionally, for certain purposes like fraud detection, a local explanation can be much more desirable than the global one.

Not many authors have focused on using interpretation techniques in the context of local explanations. Only very recently papers started to appear to investigate these issues, and many challenges remain.

To explore the possibilities of local interpretation approaches, we developed a local rule extraction technique that is, to the best of our knowledge, a novel approach in obtaining model insights. Various permutations of the approach were technically evaluated and the best technique was identified.

We developed a dashboard combining various visualization techniques, that facilitates a dialogue between fraud experts of Achmea and a given model. By exposing explanations about the decisions the model made, we support the experts in their every day work.

In our evaluation, we showed that explanations about the model are understood and can indeed help the fraud experts to identify potential fraud cases more quickly.

9.2 Limitations and improvements

The project was done in a very broad manner. This resulted in a large coverage of possible solutions, which eventually led us to a useful result for Achmea. However, this also caused us to have less time to extensively test our findings, and evaluate methods in more depth. We would have liked to evaluate more, using various UCI data sets, to ensure our observations and claims are sound.

At first glance, our current findings show a significant inconsistency across interpretation techniques. We presented three different methods to determine which features are important: feature

importance gives us a score, variance of a PDP plot should also be an indication of importance, and we expect features that show up in important rules extracted with local rule extraction to also be important for the classification. The result of these three methods do not always correspond; in the worst case they all identify different features as important.

However, we have an intuition why this disparity occurs. Consider the example shown in Figure 9.1. Two decisions are shown that predict *green*. If we want to obtain important features for the class *green*, feature importance would consider both constraints $C1$ and $C2$ important. However, when we consider local pruning data within the circle, $C1$ has no significance, as it does not divide the instances within the circle in any way. In this case only $C2$ would be considered important.

It is very difficult to tell which of these methods represents feature importance correctly, as no ground truth is available for such a metric. More research is needed in order to explain this disparity.

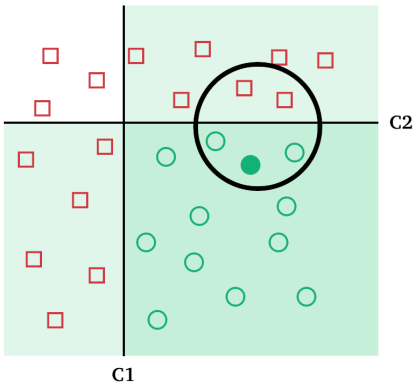


Figure 9.1: Visualization of two decisions $C1$ and $C2$ to show important feature disparity.

.

Also our novel local rule selection algorithm requires more research to prove its effectiveness, and improve its efficiency. In its current form, the algorithm needs to classify millions of synthetic data points to find sufficient data belonging to a different class. This is a Monte Carlo approach that can be significantly improved by using prior knowledge about the decision boundary. Very recent work by Krishnan and Wu [64] extracts relevant training data for the classification of an instance, which could prove useful for this research. Our technique merely scratched the surface, but we think this method shows a lot of potential to explain Random Forests.

Finally, we like to explore the possibility to combine the two dashboards developed during this project. One way to do this would be to connect the constraints at the right of the Sankey diagram to a table of features, which could display similar information to what is shown in the feature dashboard. Before this combination is implemented, it would be desirable if the feature importance disparity we mentioned earlier is resolved, as this new dashboard would draw more attention to the differences.

9.3 Future work

Feature interactions During the project we noted that complex models such as the Achmea fraud model are unlikely to make decisions based on a single feature. They derive complex combinations of features instead. Hence, the PDP plots generated for the Achmea model show very small effects: the plots only show effects for a single feature. It would be interesting to explore whether it is possible to extract such combinations of features from the model.

Some authors have created techniques to find combinations of features that interact [104, 75, 76]. However, these methods focus on finding patterns in the data and only consider that data set. They propose new models that can be fitted on the data to find which features interact, while we want to extract the interactions from the existing model. Doing so would give an insight into what

combinations the model identifies as important, disregarding whether the combinations are actually realistic. For instance, these combinations can help data scientists to identify wrong behavior of the model.

To find such interactions, we identified two possible approaches. Firstly, n-dimensional partial dependence can be used to find which features interact. This is a very computationally heavy approach and running time will increase exponentially with n. Second, rules can be extracted and analyzed to see which features often occur in the same decision rules of a Random Forest. More research is needed to investigate whether these approaches are feasible.

Multidimensional decision boundary projection We thought of another interesting approach when we created a simplified visualization of decision boundaries. A two dimensional plot, like shown in Figure 6.5a, can very clearly convey the decisions of a model, but is limited to two dimensional feature space. Dimensionality reduction can be used in combination with Voronoi tessellation to create such a plot for any higher dimensional model. A very recent work by Migut et al. [80] demonstrates this approach.

Such method would not be effective for very complex models such as the Achmea model we worked with. However, this could also be used to show the decision boundary of a locally simplified model, or even a single decision rule.

In addition, we think such a plot could be enriched with rule and feature importance metrics. The width or color of the line dividing areas could vary depending on the rule that corresponds to a part of the decision boundary. This would intuitively convey this information, without adding much complexity to the visualization.

Finally, we think it may be possible to show a prediction score heatmap rather than just the boundary. We imagine that heatmaps can be toggled on and off for each class, and different heatmaps that overlap will blend together to show that both classes are likely according to the model.

Feature contribution We noted a problem with the feature contribution method presented by Palczewska et al. [91]. In their work they use local increments to determine which features have most contribution to the score. However, local increments are insensitive to the amount of data that is partitioned by the constraint.

Consider the example in Figure 9.2. The local increment of two edges is shown, which in this example turns out to be the same. However, the bottom decision filters out an insignificant amount of *green* cases, compared to the total number of records.

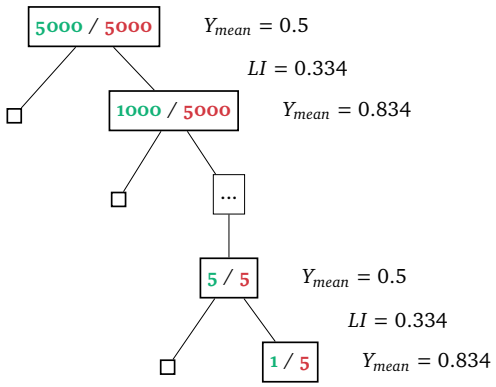


Figure 9.2: Example of issue with feature contribution method.

We think this problem can be solved by making the importance metric dependent on the number of training set records in the node. What the best solution is, is left for future research.

In addition, it would be interesting to see how more common split criteria like information gain and Gini impurity perform as a substitute for local increments. Perhaps, matching the split criteria

used for training the trees could have a positive impact in the accuracy of the feature importance. However, as stated before, it is difficult to evaluate whether a feature importance metric is accurate.

9.4 Recommendations

During the project we tried to leave the model untouched, as it was our goal to explain the model, not to improve it. Nonetheless, we found a few improvements to the model that could significantly improve performance and simplicity, and also positively affects the explanations shown in our dashboards.

We showed that the imputed values for missing values are, for some features, not realistic values within the range of the data. For instance, for feature *Feature 27* this led to rules in the model that the fraud team struggled to explain (described in Chapter 8).

We also recommend simplifying the model. From our model simplification research we conclude that it should be possible to train a single Random Forest that should estimate the behavior of the 100 ones. This is supported by the fact that global model simplification achieved very good results; reducing 2.3 million rules to 41 while retaining most of the accuracy from the model gives us a strong intuition that a lot of decisions made in the model are not significant for the behavior. In addition, we showed in Section 2.3.6 that amount of trees can be reduced from 500 to 200 without any performance loss.

This simplification significantly speeds up model interpretation techniques. Partial dependence plots can be computed with all training data rather than a subset, and local rule extraction will be quick enough to apply on a large set of instances.

Bibliography

[1] Sunil Arya and David M Mount. Approximate nearest neighbor queries in fixed dimensions. In *SODA*, volume 93, pages 271–280, 1993. 55

[2] Anneleen Van Assche and Hendrik Blockeel. Seeing the Forest through the Trees: Learning a Comprehensible Model from an Ensemble. *Ecml*, 2007. 3, 12, 13, 28, 29

[3] Aaron Bangor, Philip T Kortum, and James T Miller. An empirical evaluation of the system usability scale. *Intl. Journal of Human–Computer Interaction*, 24(6):574–594, 2008. 74

[4] Osbert Bastani, Carolyn Kim, and Hamsa Bastani. Interpreting Blackbox Models via Model Extraction. *arXiv preprint arXiv:1705.08504*, 2017. URL http://arxiv.org/abs/1705.08504. 13

[5] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine learning*, 36(1):105–139, 1999. 12

[6] José Manuel Benítez, Juan Luis Castro, and Ignacio Requena. Are artificial neural networks black boxes? *IEEE Transactions on neural networks*, 8(5):1156–1164, 1997. 7

[7] Gérard Biau, Erwan Scornet, et al. A random forest guided tour. *TEST-An Official Journal of the Spanish Society of Statistics and Operations Research*, 25(2):197–227, 2016. 1

[8] Catherine L Blake and Christopher J Merz. Uci repository of machine learning databases [http://www. ics. uci. edu/˜ mlearn/mlrepository. html]. irvine, ca: University of california. *Department of Information and Computer Science*, 55, 1998. 29

[9] Matthew Brehmer and Tamara Munzner. A multi-level typology of abstract visualization tasks. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2376–2385, 2013. ISSN 10772626. doi: 10.1109/TVCG.2013.124. 21

[10] Leo Breiman. Stacked regressions. *Machine learning*, 24(1):49–64, 1996. 12

[11] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. 1, 12, 13, 22, 24, 43

[12] Leo Breiman and Adele Cutler. U.S. trademark registration number 3185828, registered 2006/12/19, 2006. 12

[13] John Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189 (194):4–7, 1996. 70, 74

[14] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '06*, page 535, 2006. doi: 10.1145/1150402.1150464. URL http://portal.acm.org/citation.cfm?doid=1150402.1150464. 13, 28

[15] Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. Intelligible Models for HealthCare. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '15*, pages 1721–1730, 2015. doi: 10.1145/2783258.2788613. URL http://dl.acm.org/citation.cfm?id=2783258.2788613. 1, 4

[16] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002. ISSN 10769757. doi: 10.1613/jair.953. 56

[17] Alexandra Chouldechova. Fair prediction with disparate impact: A study of bias in recidivism prediction instruments. *arXiv preprint arXiv:1703.00056*, 2017. 4

[18] Alejandro Correa Bahnsen, Aleksandar Stojanovic, Djamila Aouada, and Bjorn Ottersten. Improving Credit Card Fraud Detection with Calibrated Probabilities. *Proceedings of the 2014 SIAM International Conference on Data Mining*, (April 24-26, 2014.):677 – 685, 2014. doi: 10.1137/1.9781611973440.78. URL https://www.researchgate.net/publication/272457681{_}Improving{_}Credit{_}Card{_}Fraud{_}Detection{_}with{_}Calibrated{_}Probabilities//hdl.handle.net/10993/15233. 1

[19] Paulo Cortez and Mark J Embrechts. Using sensitivity analysis and visualization techniques to open black box data mining models. *Information Sciences*, 2012. 25

[20] Nello Cristianini and John Shawe-Taylor. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000. 3

[21] Inês de Castro Dutra, David Page, Vítor Santos Costa, and Jude Shavlik. An empirical evaluation of bagging in inductive logic programming. In *International Conference on Inductive Logic Programming*, pages 48–65. Springer, 2002. 12

[22] Houtao Deng. Interpreting tree ensembles with inTrees. *arXiv preprint arXiv:1408.5456*, pages 1–18, 2014. URL http://arxiv.org/abs/1408.5456. 30, 43, 51

[23] Houtao Deng and George Runger. Gene selection with guided regularized random forest. *Pattern Recognition*, 46(12):3483–3489, 2013. 31

[24] Houtao Deng, George Runger, and Eugene Tuv. Bias of importance measures for multi-valued attributes and solutions. *Artificial neural networks and machine Learning–ICANN 2011*, pages 293–300, 2011. 13

[25] Houtao Deng, George Runger, Eugene Tuv, and Wade Bannister. Cbc: An associative classifier with a small number of rules. *Decision Support Systems*, 59:163–170, 2014. 31

[26] Thomas G Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine learning*, 40(2):139–157, 2000. 12

[27] Thomas G Dietterich et al. Ensemble methods in machine learning. *Multiple classifier systems*, 1857:1–15, 2000. 12

[28] Pedro Domingos. Knowledge Acquisition from Examples Via Multiple Models. *In Proceedings of the Fourteenth International Conference on Machine Learning*, pages 98–106, 1997. 1, 3, 4, 13, 28

[29] Pedro Domingos. Knowledge discovery via multiple models. *Intelligent Data Analysis*, 2(3): 187–202, 1998. ISSN 1088467X. doi: 10.3233/IDA-1998-2303. 1, 3, 13, 28, 54

[30] Bradley Efron and Robert J Tibshirani. *An introduction to the bootstrap*. CRC press, 1994. 12

[31] Aron Eklund. Beeswarm: the bee swarm plot, an alternative to stripchart. *R package version 0.1*, 5, 2012. 38

[32] César Ferri, José Hernández-Orallo, and M José Ramírez-Quintana. From ensemble methods to comprehensible models. In *International Conference on Discovery Science*, pages 165–177. Springer, 2002. 3, 13

[33] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of human genetics*, 7(2):179–188, 1936. 53

[34] George Fishman. *Monte Carlo: concepts, algorithms, and applications*. Springer Science & Business Media, 2013. 55

[35] Alex A. Freitas. Comprehensible Classification Models âĂŞ a position paper. *ACM SIGKDD Explorations Newsletter*, 15(1):1–10, 2013. ISSN 19310145. doi: 10.1145/2594473.2594475. URL http://dl.acm.org.miman.bib.bth.se/citation.cfm?id=2594475. 1, 3, 7

[36] Alex A Freitas, Daniela C Wieser, and Rolf Apweiler. On the importance of comprehensible classification models for protein function prediction. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 7(1):172–182, 2010. 1, 3

[37] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *Icml*, volume 96, pages 148–156, 1996. 12

[38] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001. ISSN 00905364. doi: DOI10.1214/aos/1013203451. 2, 12, 25, 43, 47

[39] Ryohei Fujimaki and Satoshi Morinaga. Factorized asymptotic bayesian inference for mixture modeling. In *International Conference on Artificial Intelligence and Statistics*, pages 400–408, 2012. 32

[40] Alex Goldstein, Adam Kapelner, Justin Bleich, and Emil Pitkin. Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65, 2015. ISSN 1061-8600. doi: 10.1080/10618600.2014.907095. URL http://arxiv.org/abs/1309.6392. 26, 27, 48

[41] Ibai Gurrutxaga, Jesús Ma Pérez, Olatz Arbelaitz, Javier Muguerza, José I Martín, and Ander Ansuategi. CTC: An Alternative to Extract Explanation from Bagging. *Artificial Intelligence*, 4177 (1):49–62, 2006. ISSN 03029743. doi: 10.1007/11881216. URL http://www.springerlink.com/content/yvl9v1u61q98077v. 1, 13, 29

[42] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001, 1990. 12

[43] Satoshi Hara and Kohei Hayashi. Making Tree Ensembles Interpretable. *arXiv preprint arXiv:1606.09066v3*, 2016. URL http://arxiv.org/abs/1606.09066. 1, 30, 32, 51

[44] Radoslav Harman and Vladimír Lacko. On decompositional algorithms for uniform sampling from n-spheres and n-balls. *Journal of Multivariate Analysis*, 101(10):2297–2304, 2010. 55

[45] Sherif Hashem. Optimal linear combinations of neural networks. *Neural networks*, 10(4): 599–614, 1997. 12

[46] Kohei Hayashi, Shin-ichi Maeda, and Ryohei Fujimaki. Rebuilding factorized information criterion: Asymptotically accurate marginal likelihood. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1358–1366, 2015. 32

[47] Jerry L Hintze and Ray D Nelson. Violin plots: a box plot-density trace synergism. *The American Statistician*, 52(2):181–184, 1998. 37, 38

[48] Tin Kam Ho. Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 278–282. IEEE, 1995. 1, 12

[49] Tin Kam Ho. A data complexity analysis of comparative advantages of decision forest constructors. *Pattern Analysis & Applications*, 5(2):102–112, 2002. 12

[50] Jin Huang and Charles X Ling. Using auc and accuracy in evaluating learning algorithms. *IEEE Transactions on knowledge and Data Engineering*, 17(3):299–310, 2005. 1, 3

[51] Laurent Hyafil and Ronald L Rivest. Constructing optimal binary decision trees is np-complete. *Information processing letters*, 5(1):15–17, 1976. 10

[52] Grzegorz Ilczuk and Alicja Wakulicz-Deja. Visualization of rough set decision rules for medical diagnosis systems. *Rough sets, fuzzy sets, data mining and granular computing*, pages 371–378, 2007. 41

[53] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013. 10, 13

[54] Nathalie Japkowicz and Mohak Shah. *Evaluating learning algorithms: a classification perspective*. Cambridge University Press, 2011. 1, 3

[55] Tao Jiang and Art B Owen. Quasi-regression for visualization and interpretation of black box functions, 2002. 2

[56] Peter Kampstra et al. Beanplot: A boxplot alternative for visual comparison of distributions. *Journal of Statistical Software*, 28(1):1âĂŞ9, 2008. 37, 38

[57] Robert E Kass and Adrian E Raftery. Bayes factors. *Journal of the american statistical association*, 90(430):773–795, 1995. 32

[58] Been Kim, Rajiv Khanna, and Oluwasanmi Koyejo. Examples are not Enough, Learn to Criticize! Criticism for Interpretability. *Nips*, pages 2280–2288, 2016. ISSN 10495258. 4

[59] Yves Kodratoff. The comprehensibility manifesto. *KDD Nugget Newsletter*, 94(9), 1994. 1, 3, 5

[60] Ron Kohavi and Foster Provost. Glossary of terms. *Machine Learning*, 30(2-3):271–274, 1998. 5, 6

[61] Josua Krause, Adam Perer, and Enrico Bertini. Using Visual Analytics to Interpret Predictive Machine Learning Models. *ICML Workshop on Human Interpretability in Machine Learning*, pages 106–110, 2016. 1, 2, 4, 7, 26, 48

[62] Josua Krause, Adam Perer, and Kenney Ng. Interacting with Predictions: Visual Inspection of Black-box Machine Learning Models. *ACM Conference on Human Factors in Computing Systems*, pages 5686–5697, 2016. doi: 10.1145/2858036.2858529. URL http://dl.acm.org/citation.cfm?doid=2858036.2858529. 1, 2, 4, 7, 26, 48

[63] Josua Krause, Aritra Dasgupta, Jordan Swartz, Yindalon Aphinyanaphongs, and Enrico Bertini. A Workflow for Visual Diagnostics of Binary Classifiers using Instance-Level Explanations. *arXiv preprint arXiv:1705.01968*, 2017. URL http://arxiv.org/abs/1705.01968. 2, 7, 43, 48

[64] Sanjay Krishnan and Eugene Wu. Palm: Machine learning explanations for iterative debugging. In *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics*, page 4. ACM, 2017. 76

[65] Anders Krogh and Jesper Vedelsby. Neural network ensembles, cross validation, and active learning. In *Advances in neural information processing systems*, pages 231–238, 1995. 12

[66] Max Kuhn and Kjell Johnson. *Applied predictive modeling*, volume 810. Springer, 2013. 1

[67] Victor E Kuz'min, Pavel G Polishchuk, Anatoly G Artemenko, and Sergey A Andronati. Interpretation of qsar models based on random forest methods. *Molecular informatics*, 30(6-7): 593–603, 2011. 24

[68] Ken Lau. Random Forest Ensemble Visualization. (Unpublished work), 2014. URL https://pdfs.semanticscholar.org/9cb3/3b2459730e8b2bf2b3364bbf5ffc28337523.pdf. 33

[69] Ruey-Hsia Li and Geneva G Belford. Instability of decision tree classification algorithms. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 570–575. ACM, 2002. 10, 12

[70] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R News*, 2(3): 18–22, 2002. URL http://CRAN.R-project.org/doc/Rnews/. 22, 25

[71] Rensis Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932. 74

[72] Tony Lindgren. Model Based Sampling Fitting an ensemble of models into a single model. In *Computational Science and Computational Intelligence (CSCI), 2015 International Conference on*, pages 186–191. IEEE, 2015. 28

[73] Zachary C Lipton. The Mythos of Model Interpretability. *ICML Workshop on Human Interpretability in Machine Learning*, 2016. 1, 4, 5

[74] Miao Liu, Mingjun Wang, Jun Wang, and Duo Li. Comparison of random forest, support vector machine and back propagation neural network for electronic tongue data classification: Application to the recognition of orange beverage and chinese vinegar. *Sensors and Actuators B: Chemical*, 177:970–980, 2013. 1, 13

[75] Wei-yin Loh. Regression Trees With Unbiased Variable Selection and Interaction Detection. *Statistica Sinica*, 12:361–386, 2002. ISSN 10170405. doi: 10.5351/KJAS.2004.17.3.459. 76

[76] Yin Lou, Rich Caruana, Johannes Gehrke, and Giles Hooker. Accurate intelligible models with pairwise interactions. *Proceedings of the 2013 KDD Conference on Knowledge Discovery and Data Mining*, page 623, 2013. doi: 10.1145/2487575.2487579. URL http://dl.acm.org/citation.cfm?doid=2487575.2487579. 5, 76

[77] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008. 18

[78] Richard Maclin and David Opitz. An empirical evaluation of bagging and boosting. *AAAI/IAAI*, 1997:546–551, 1997. 12

[79] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943. 1, 3

[80] MA Migut, Marcel Worring, and Cor J Veenman. Visualizing multi-dimensional decision boundaries in 2d. *Data Mining and Knowledge Discovery*, 29(1):273–295, 2015. 77

[81] Matthew W Mitchell. Bias of the random forest out-of-bag (oob) error for certain input parameters. *Open Journal of Statistics*, 1(3):205–211, 2011. doi: 10.4236/ojs.2011.13024. 13

[82] Tom M Mitchell. *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research, Rutgers Univ. New Jersey, 1980. 1, 6

[83] Tom M Mitchell. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877, 1997. 7

[84] Mervin E Muller. A note on a method for generating points uniformly on n-dimensional spheres. *Communications of the ACM*, 2(4):19–20, 1959. 55

[85] Sreerama K Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data mining and knowledge discovery*, 2(4):345–389, 1998. 10

[86] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436. IEEE, 2015. 4

[87] UC Berkeley School of Information. What is Data Science? DataScience@Berkley. https://web.archive.org/web/20130817091749/http://datascience.berkeley.edu:80/about/what-is-data-science/, 2013. Accessed: 2017-07-11. 1

[88] David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999. 1, 3, 12

[89] David W Opitz and Jude W Shavlik. Actively searching for an effective neural network ensemble. *Connection Science*, 8(3-4):337–354, 1996. 12

[90] David W Opitz and Jude W Shavlik. Generating accurate and diverse members of a neural-network ensemble. In *Advances in neural information processing systems*, pages 535–541, 1996. 12

[91] Anna Palczewska, Jan Palczewski, Richard Marchese Robinson, and Daniel Neagu. Interpreting random forest classification models using a feature contribution method. In *Integration of reusable systems*, pages 193–218. Springer, 2014. URL http://arxiv.org/abs/1312.1121. 7, 24, 25, 35, 43, 45, 46, 77

[92] Jesús M. Pérez, Javier Muguerza, Olatz Arbelaitz, Ibai Gurrutxaga, and José I. Martín. Combining multiple class distribution modified subsamples in a single tree. *Pattern Recognition Letters*, 28(4):414–422, 2007. ISSN 01678655. doi: 10.1016/j.patrec.2006.08.013. 29

[93] J.M. Perez, J Muguerza, O Arbelaitz, and I Gurrutxaga. A new algorithm to build consolidated trees: study of the error rate and steadiness. *Intelligent information processing and web mining: proceedings of the International IIS: IIPWM'04 Conference held in Zakopane, Poland, May 17-20, 2004*, 25:79, 2004. URL http://books.google.com/books?hl=en{&}lr={&}id=KIEjoKbSypQC{&}oi=fnd{&}pg=PA79{&}dq=A+new+algorithm+to+build+consolidated+trees+:+study+of+the+error+rate+and+steadiness{&}ots=WppPqlhP7d{&}sig=IFZJmuzfbiLuSwX19tCv9T9UsKs. 29

[94] Foster J Provost, Tom Fawcett, Ron Kohavi, et al. The case against accuracy estimation for comparing induction algorithms. In *ICML*, volume 98, pages 445–453, 1998. 1, 3, 4

[95] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986. 9, 11

[96] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1-55860-238-0. 9, 30

[97] EU Regulation. Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation). *Official Journal of the European Union (OJ)*, 59:1–88, 2016. 4

[98] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016. ISBN 9781450321389. doi: 10.1145/2939672.2939778. URL http://arxiv.org/abs/1602.04938. 2, 4, 7, 53

[99] Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990. 12

[100] Roeland Scheepens, Steffen Michels, Huub Van De Wetering, and Jarke J. Van Wijk. Rationale Visualization for Safety and Security. *Computer Graphics Forum*, 34(3):191–200, 2015. ISSN 14678659. doi: 10.1111/cgf.12631. 41

[101] Claude E Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001. 9

[102] Jack W Smith, JE Everhart, WC Dickson, WC Knowler, and RS Johannes. Using the adap learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Annual Symposium on Computer Application in Medical Care*, page 261. American Medical Informatics Association, 1988. 51, 61

[103] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, 2009. 1, 3

[104] Daria Sorokina, Rich Caruana, Mirek Riedewald, and Daniel Fink. Detecting statistical interactions with additive groves of trees. *Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 1000–1007, 2008. doi: 10.1145/1390156.1390282. URL http://portal.acm.org/citation.cfm?doid=1390156.1390282. 76

[105] Gregor Stiglic and Peter Kokol. Evolutionary tuning of combined multiple models. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4252 LNAI:1297–1304, 2006. ISSN 16113349 03029743. 13, 28

[106] Gregor Stiglic and Peter Kokol. Evolutionary approach to combined multiple models tuning. *International Journal of Knowledge-based and Intelligent Engineering Systems*, 11(4):227–235, 2007. ISSN 1327-2314. URL http://portal.acm.org/citation.cfm?id=1374743.1374748. 13, 28

[107] Carolin Strobl, Anne-Laure Boulesteix, Thomas Kneib, Thomas Augustin, and Achim Zeileis. Conditional variable importance for random forests. *BMC bioinformatics*, 9(1):307, 2008. 13, 24

[108] Erik Štrumbelj and Igor Kononenko. A general method for visualizing and explaining black-box regression models. In *International Conference on Adaptive and Natural Computing Algorithms*, pages 21–30. Springer, 2011. 2

[109] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013. 4

[110] Paolo Tamagnini, Josua Krause, Aritra Dasgupta, and Enrico Bertini. Interpreting Black-Box Classifiers Using Instance-Level Visual Explanations. In *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics - HILDA'17*, pages 1–6. ACM Press, 2017. ISBN 9781450350297. 2, 7

[111] R Core Team. R language definition. *Vienna, Austria: R foundation for statistical computing*, 2000. 22

[112] John W Tukey. Exploratory data analysis. *Addison-Wesley Series in Behavioral Science: Quantitative Methods, Reading, Mass.: Addison-Wesley, 1977*, 1977. 37, 38

[113] Berk Ustun and Cynthia Rudin. Methods and Models for Interpretable Linear Classification. *arXiv*, pages 1–57, 2014. URL http://arxiv.org/abs/1405.4047. 5, 6, 7

[114] Laurens Van Der Maaten. Barnes-hut-sne. *arXiv preprint arXiv:1301.3342*, 2013. 18

[115] Hui-Xin Wang, Laura Fratiglioni, Giovanni B Frisoni, Matti Viitanen, and Bengt Winblad. Smoking and the occurence of alzheimer's disease: Cross-sectional and longitudinal data in a population-based study. *American journal of epidemiology*, 149(7):640–644, 1999. 4

[116] Soeren H. Welling, Hanne H. F. Refsgaard, Per B. Brockhoff, and Line H. Clemmensen. Forest Floor Visualizations of Random Forests. *Supplementrary*, pages 1–13, 2016. URL http://arxiv.org/abs/1605.09196. 26

[117] David West, Scott Dellana, and Jingxia Qian. Neural network ensemble strategies for financial decision applications. *Computers & operations research*, 32(10):2543–2559, 2005. 1, 4

[118] J.J. Van Wijk. Bridging the Gaps. *IEEE Computer Graphics and Applications*, 26(6):6–9, 2006. ISSN 0272-1716. doi: 10.1109/MCG.2006.120. 16

[119] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016. 7

[120] Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, S Yu Philip, et al. Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1):1–37, 2008. 9

[121] Abraham J Wyner, Matthew Olson, Justin Bleich, and David Mease. Explaining the success of adaboost and random forests as interpolating classifiers. *arXiv preprint arXiv:1504.07676*, 2015. 13

[122] Zhi-Hua Zhou, Y. Jiang, and S.-F. Chen. Extracting symbolic rules from trained neural network ensembles. *AI Communications*, 16(1):3–15, 2003. ISSN 09217126. URL http://portal.acm.org/citation.cfm?id=1218644. 13, 28

Appendix A: Features in Achmea data set

[As a consequence of anonymization, the descriptions of features in the Achmea data set have been removed.]

Appendix B: Example information gain calculation

Consider the first split in the decision tree in Figure 2.2 on the feature *Outlook*. In the *Play Golf?* column of the training data Table 2.1, among 14 records, there are 5 records labeled *No* and 9 as *Yes*. This corresponds to the values $p_{no} = \frac{5}{14}$ and $p_{yes} = \frac{9}{14}$ respectively. We can now calculate the entropy $H(S)$ of the entire data set before the split as $-\left(\frac{5}{14}\log_2(\frac{5}{14}) + \frac{9}{14}\log_2(\frac{9}{14})\right) \approx 0.94$. In order to compute the information gain, we also need the weighted average of the entropy of the children. The entropy of the children is computed in a similar fashion; for instance, for the first subset belonging to the child where *Outlook* = "*Sunny*", the entropy is $-\left(\frac{3}{5}\log_2(\frac{3}{5}) + \frac{2}{5}\log_2(\frac{2}{5})\right) \approx 0.97$. For *Outlook* = "*Outlook*" we can see that all records in the subset are part of the same class *Yes*, which means entropy is 0. Finally, for *Outlook* = "*Rain*", the entropy is $-\left(\frac{2}{5}\log_2(\frac{2}{5}) + \frac{3}{5}\log_2(\frac{3}{5})\right) \approx 0.97$.

Next, we can compute the information gain for the split according do Equation 2.2 in the following way: $0.94 - \left(\frac{5}{14} \cdot 0.97 + \frac{4}{14} \cdot 0 + \frac{5}{14} \cdot 0.97\right) \approx 0.247$. As the tree was already trained on this data set, we know that the split on feature *Outlook* should have the most information gain compared to the other features.

Appendix C: Global PDPs

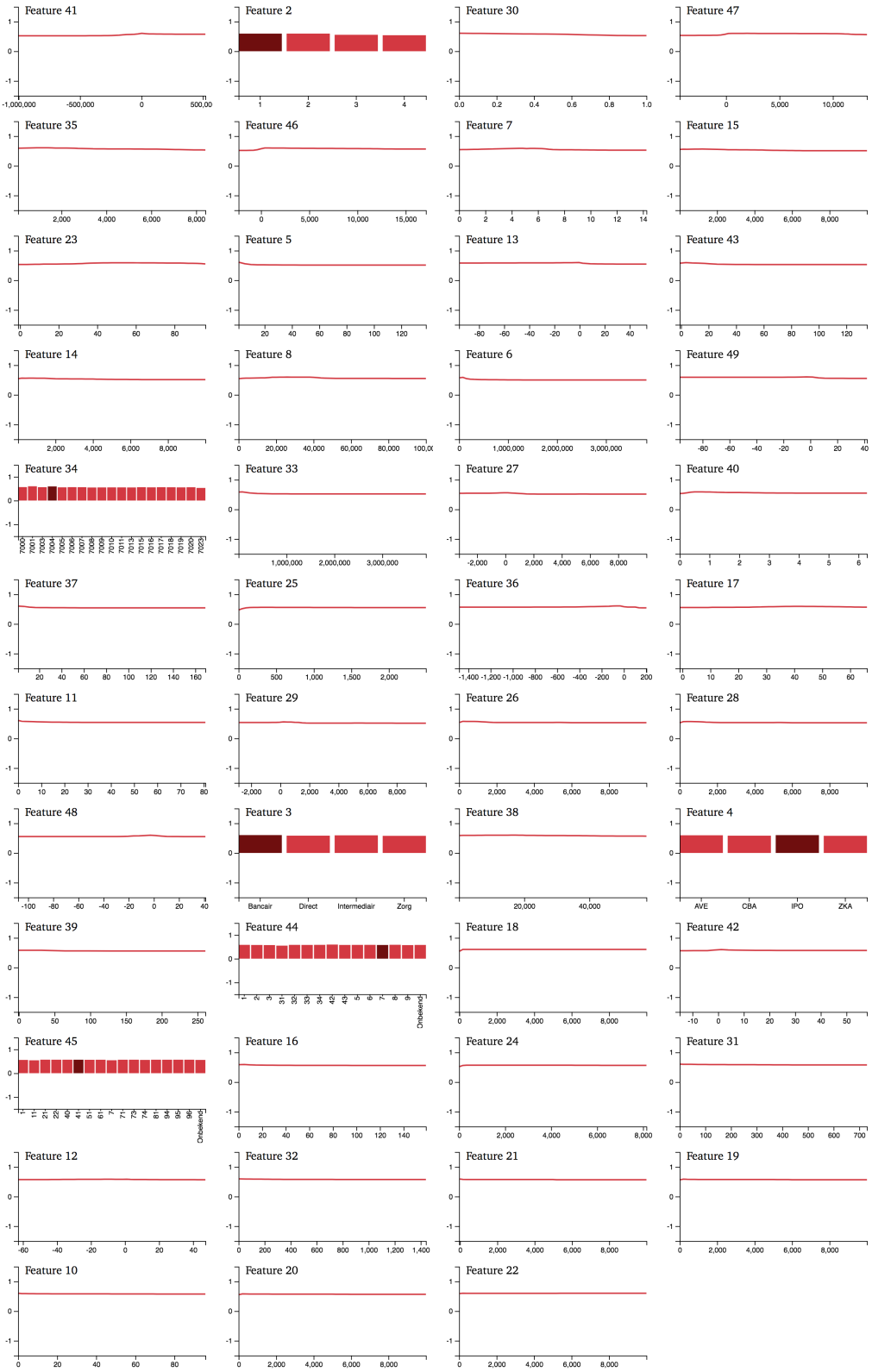


Figure C.1: Global partial dependence plots for the Achmea data set.

Appendix D: Local PDPs

D.1 ANP128

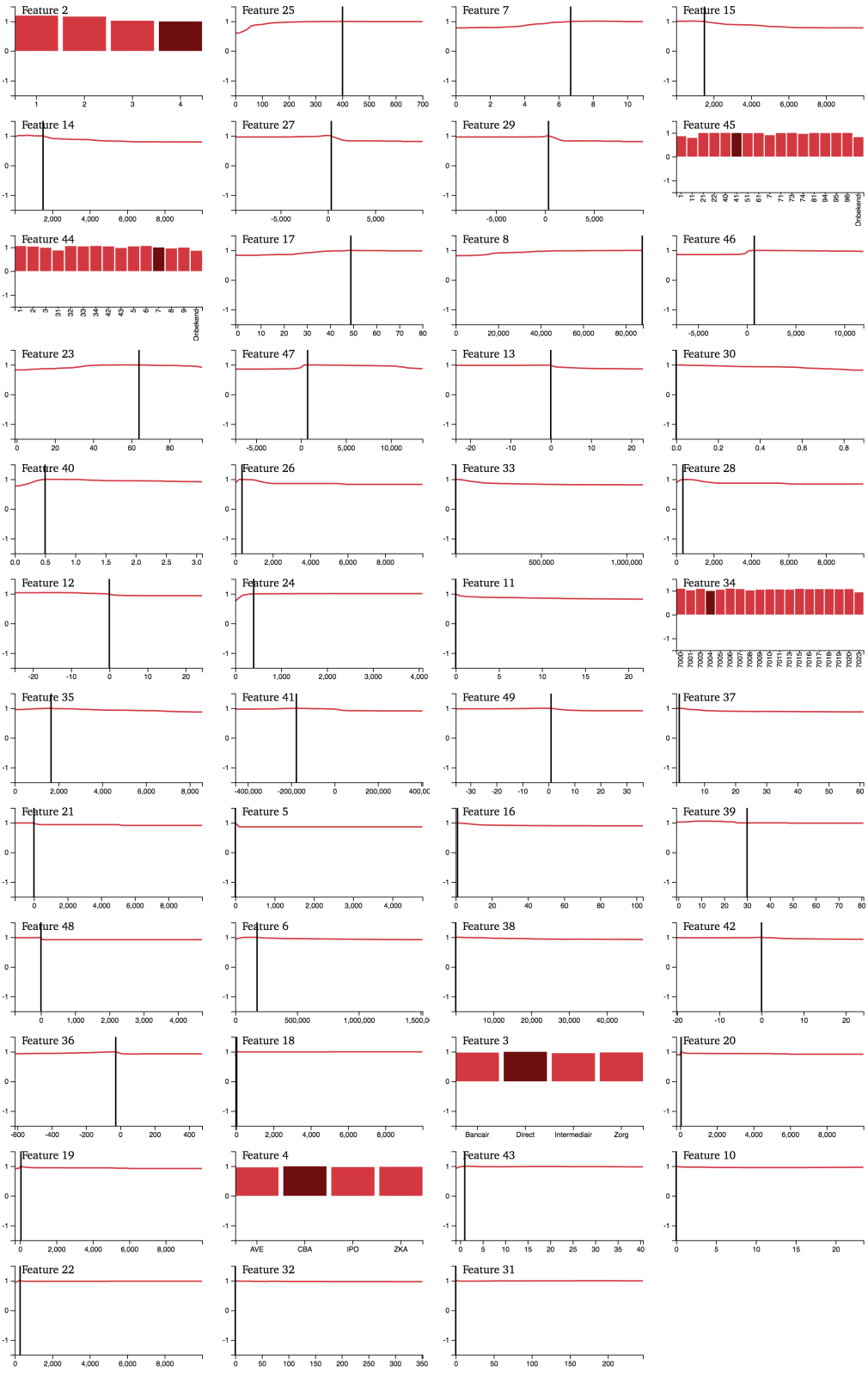


Figure D.1: Local partial dependence plots for the ANP128.

D.2 ANP87

Figure D.2: Local partial dependence plots for the ANP87.

Appendix E: Global InTrees rules

len	freq	err	condition	pred	impRRF
5	0.146	0.274	Feature 7>0.83 & Feature 13<=0.96 & Feature 14<=2301 & Feature 25>60.875 & Feature 33<=148332	Wel fraude	1
6	0.134	0.284	Feature 2 %in% c('1','2') & Feature 7>4.605 & Feature 25>38.0294117647059 & Feature 26<=1698 & Feature 34 %in% c('7000','7001','7003','7004','7005','7006','7007','7008','7009','7010','7011','7013','7015','7016','7017','7018','7019','7020') & Feature 35<=7761.6	Wel fraude	0.512
4	0.022	0.091	Feature 2 %in% c('1') & Feature 4 %in% c('CBA','ZKA') & Feature 24<=44.5 & Feature 39<=20	Wel fraude	0.24
3	0.234	0.128	Feature 7<=6.94 & Feature 14>988 & Feature 22<=127.25	Geen fraude	0.164
5	0.098	0.347	Feature 2 %in% c('1') & Feature 17>25 & Feature 26<=470.5 & Feature 31<=140.868421052632 & Feature 40<=1.43918918918919	Wel fraude	0.132
3	0.792	0.227	Feature 42<=3.5 & Feature 42>-0.5 & Feature 49<=6.5	Geen fraude	0.095
2	0.654	0.135	Feature 15>986.666666666665 & Feature 25<=119.5	Geen fraude	0.093
4	0.4	0.125	Feature 8<=36785.3333333334 & Feature 32 <=175.380952380952 & Feature 33>37334.5 & Feature 34 %in% c('7001','7004','7008','7015','7016','7020','7023')	Geen fraude	0.091
6	0.166	0.301	Feature 2 %in% c('1','2','3') & Feature 11<=0.17 & Feature 20>-2.5 & Feature 30<=0.593103448275862 & Feature 39<=25 & Feature 29<=1376	Wel fraude	0.078
3	0.478	0.096	Feature 2 %in% c('2','3','4') & Feature 15>1234.53409090909 & Feature 24<=1039.5	Geen fraude	0.061
3	0.516	0.19	Feature 33>51151.5 & Feature 41<=89738.5 & Feature 42<=9	Geen fraude	0.052
5	0.144	0.375	Feature 11<=23.995 & Feature 14<=943 & Feature 23<=73.5 & Feature 33<=463184.5 & Feature 40>0.261363636363636	Wel fraude	0.052
4	0.412	0.18	Feature 7>2.205 & Feature 12>-8.35 & Feature 38>8572.33130081301 & Feature 48<=1	Geen fraude	0.043
3	0.104	0.115	Feature 21<=0.0588235294117645 & Feature 34 %in% c('7004','7011') & Feature 35>1579.58617424242	Geen fraude	0.038
2	0.16	0.188	Feature 8<=26316.5625 & Feature 19<=3.21428571428572	Geen fraude	0.036
2	0.658	0.182	Feature 3 %in% c('Bancair','Intermediair') & Feature 34 %in% c('7000','7001','7004','7008','7017','7019','7023')	Geen fraude	0.029
1	0.944	0.239	Feature 7<=8.635	Geen fraude	0.028
3	0.456	0.083	Feature 2 %in% c('2','3','4') & Feature 15>1364.75 & Feature 34 %in% c('7001','7004','7008','7015','7023')	Geen fraude	0.026
7	0.078	0.385	Feature 2 %in% c('1','2','3') & Feature 6<=165121 & Feature 12<=10.99 & Feature 17>39.5 & Feature 18<=16.1818181818182 & Feature 20<=222.044303797469 & Feature 20>0.954545454545454	Wel fraude	0.026
2	0.396	0.141	Feature 34 %in% c('7004','7008','7010','7016','7020') & Feature 35>1545.21875	Geen fraude	0.025
4	0.094	0.383	Feature 2 %in% c('1','2') & Feature 22<=55.25 & Feature 23<=76 & Feature 25>51.5833333333333	Wel fraude	0.023
3	0.036	0.278	Feature 19>1.25 & Feature 37>2.5 & Feature 28<=93.2833333333334	Geen fraude	0.023
2	0.1	0.3	Feature 25>51.1041666666667 & Feature 35<=1455.2	Wel fraude	0.022
3	0.26	0.2	Feature 5>2.5 & Feature 18<=15.0666666666667 & Feature 34 %in% c('7001','7004','7011','7018')	Geen fraude	0.018
1	0.9	0.213	Feature 25<=268.5	Geen fraude	0.017
4	0.684	0.167	Feature 12>-1.395 & Feature 13>-27.015 & Feature 24<=1595.5 & Feature 25<=264.107142857143	Geen fraude	0.016
3	0.264	0.197	Feature 20<=8.05844155844155 & Feature 22<=85.5 & Feature 28>150.584016393442	Geen fraude	0.015
1	0.99	0.257	Feature 29>-705	Geen fraude	0.015

Continued on next page

Table E.1 – *Continued from previous page*

len	freq	err	condition	pred	impRRF
6	0.094	0.128	Feature 15<=5073.75 & Feature 18<=13.1428571428571 & Feature 22<=162.666666666667 & Feature 24>56 & Feature 34 %in% c('7001','7004','7007','7015','7017','7023') & Feature 35>2481.54761904762	Geen fraude	0.014
3	0.482	0.402	Feature 40>0.19634703196347 & Feature 29<=1405 & Feature 29>-314.333333333333	Geen fraude	0.014
3	0.28	0.129	Feature 3 %in% c('Bancair','Direct') & Feature 7<=3.91 & Feature 33>43144	Geen fraude	0.013
3	0.162	0.111	Feature 10>0.5 & Feature 13>-3.075 & Feature 24<=2035.5	Geen fraude	0.013
3	0.24	0.067	Feature 4 %in% c('IPO') & Feature 7<=4.025 & Feature 38>7632.76704545455	Geen fraude	0.013
8	0.158	0.342	Feature 26<=1148.5 & Feature 27<=1026 & Feature 30<=0.0296610169491526 & Feature 40<=2.06512605042017 & Feature 40>0.435606060606061 & Feature 41<=33560.5 & Feature 28>3 & Feature 48>-10	Wel fraude	0.013
4	0.116	0.276	Feature 7>5.4 & Feature 8>9726.20833333335 & Feature 11<=0.06 & Feature 26<=1692.5	Wel fraude	0.012
5	0.296	0.135	Feature 5>1.5 & Feature 12>-6.14 & Feature 18<=13.3729166666667 & Feature 25<=372.083333333333 & Feature 25>2.14285714285715	Geen fraude	0.011
3	0.012	0	Feature 10<=24.015 & Feature 12>8.055 & Feature 18>1.5	Geen fraude	0.011
2	0.892	0.222	Feature 12>-8.57 & Feature 13>-26.49	Geen fraude	0.011
3	0.464	0.164	Feature 2 %in% c('2','3','4') & Feature 6<=139506.5 & Feature 24<=1209.5	Geen fraude	0.01
3	0.414	0.159	Feature 34 %in% c('7004','7007','7011','7016','7023') & Feature 42<=4.5 & Feature 49<=0.5	Geen fraude	0.01
1	0.048	0.208	Feature 36<=-350.416666666667	Geen fraude	0.01

Appendix F: System Usability Scale (SUS) survey

System Usability Scale