Peter SCHEIBELHOFER

Tree-based Methods for Predictive Failure Detection in Semiconductor Fabrication

MASTERARBEIT

zur Erlangung des akademischen Grades einer/s
Diplom-Ingenieur/in

Masterstudium Finanz- und Versicherungsmathematik

Technische Universität Graz

Betreuer/in:
Univ.-Prof. Dipl.-Ing. Dr.techn. Ernst STADLOBER

Institut für Statistik

Graz, im Februar 2011

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am ....
(Unterschrift)

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitely marked all material which has been quotes either literally or by content from the used sources.

....
date (signature)

Danksagung

Ich möchte mich ganz herzlich bei Univ.Prof. DI Dr. Ernst Stadlober sowie bei der Firma austriamicrosystems AG und speziell bei DI Dr. Dietmar Gleispach und DI Dr. Günter Hayderer bedanken, für die Möglichkeit diese Masterarbeit in Zusammenarbeit mit austriamicrosystems AG zu erstellen. Vielen Dank im Besonderen für die immer freundliche, engagierte und motivierende Betreuung bei der Erstellung dieser Arbeit.

Ebenso möchte ich mich bei meinen Freunden und KollegInnen bedanken, die mir während meines Studiums immer mit Ratschlägen zur Seite gestanden sind und von denen ich viel lernen konnte.

Ganz besonders möchte ich mich bei meinen Eltern bedanken, die mir mit ihrer immerwährenden Unterstützung in jeglicher Hinsicht das Studium möglich gemacht haben. Ihnen möchte ich diese Arbeit widmen.

Abstract

The scope of this master thesis is to develop statistical models for different types of faults in a semiconductor fabrication plant. The methodology of *Classification and Regression Trees* (CART) proves to be a suitable method for analyzing the corresponding semiconductor equipment data. Moreover, recent enhancements of CART like *Random Forests* or *Boosting with trees* are highly accurate and useful for modelling purposes. The goal of the thesis is to describe and use these statistical methods in practical classification and regression problems. These problems include the prediction of the maintenance date and time of an implanter tool, the modelling of arcing on the wafer surface as well as predicting the thickness of the film deposited on the wafer. The models allow a more efficient scheduling of maintenance operations, the reduction of both scrap wafers and physical measurements.

The thesis is written in cooperation with austriamicrosystems AG.

Zusammenfassung

Der Aufgabenbereich der vorliegenden Masterarbeit ist es, statistische Modelle für verschiedene Arten von Fehlern in der Halbleiterfertigung zu entwickeln. Die Methodik der *Classification and Regression Trees* (CART) eignet sich zur Analyse der dazugehörigen Maschinendaten. Neuere Erweiterungen von CART wie *Random Forests* oder *Boosting with trees* eignen sich darüber hinaus zur Modellierung. Das Ziel der Arbeit ist es, diese statistischen Methoden zu beschreiben und in praktischen Klassifizierungs- und Regressionsproblemen anzuwenden. Zu diesen Problemen gehört das Vorhersagen des Wartungszeitpunktes einer Implanter-Maschine, das Modellieren des Phänomens von Arcing auf der Waferoberfläche und das Vorhersagen der Dicke der aufgetragenen Schicht auf den Wafer. Die Modelle ermöglichen effizientere Planbarkeit von Wartungseingriffen, die Reduktion von Ausschusswafern sowie das Einsparen physischer Messungen.

Die Masterarbeit wurde in Zusammenarbeit mit austriamicrosystems AG erstellt.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The scope of this master thesis is to develop statistical models for several different types of faults in a semiconductor fabrication plant. Typically, these faults are related to advanced process control (APC) and fault detection and classification on wafers (FDC).

The ability to schedule certain maintenance operations in an early production state yields more efficient work flows, an increase in equipment uptime and cost reduction. Typical goals of APC revolve around the reduction of certain equipment faults that cause scrap wafers or the prediction of process results such that less physical measurements are needed (virtual metrology).

As there are typically a large number of machine parameters that may or may not influence equipment faults, a first step is to find out parameters that are relevant to reduce complexity. Classification and regression trees as they were developed by Leo Breiman and his colleagues in 1984 (see [6]) represent a powerful statistical tool that turns out to be highly suitable for the above situations. On the one hand, they offer the ability to get an intuitive insight in a set of machine data to find out relevant information and on the other hand they act as a foundation to several powerful statistical methods for classification and regression problems.

This thesis deals with the use of classification and regression tree models and their enhancements for several types semiconductor fabrication faults. The chapters 2 and 3 give a theoretical overview of the statistical methods used. Chapter 4 serves as a brief introduction to the wafer fabrication processes that are investigated. Finally, the chapters 5, 6 and 7 show the utilized methodology and results achieved by using tree-based statistical methods. Chapter 5 deals with predictive maintenance on an implanter tool, chapter 6 describes the construction of a statistical model for arcing on a wafer surface and chapter 7 is about virtual metrology on a PVD tool.

Chapter 2

Classification and Regression Trees

Classification and regression tree (CART) models were introduced in 1984 by Breiman, Friedman, Olshen and Stone in their book ([6] Breiman et al. 1984). It is a non-parametric method to model and predict a response variable y by using a set of explanatory variables $x_1, ..., x_p$. The method provides an overview of the structure of the data and dependencies among certain predictors that are otherwise difficult to find out. The response variable y can be either categorical (e.g. binary) or numerical. For categorical responses, the method produces a *classification tree* which can be used for classifying new observations, i.e. assigning them to a predicted class. In case of a continuous response the result is a *regression tree* which assigns prediction values as in ordinary least squares regression.

The basic idea behind a tree model is best introduced by an example as presented in figure 2.1.

The decision tree shown was used in an american hospital to identify high risk heart attack patients. When such a patient is admitted, several ordered and binary variables such as blood pressure and age are measured giving an overview of the patients condition. The classification rule resulting from the decision tree classifies patients as class F for not high risk and class G for high risk depending on their answers of at most three questions. The patient moves in the tree from top to bottom. At each node questions has to be answered with yes or no. Depending on the answer the patient falls in the left or right branch of the tree. After answering the questions every patient falls into one specific terminal node where a respective risk class is assigned.

The goal of this chapter is to describe the mathematical methodology used to construct such a classification rule for the tree. In the following we present a detailed description of the underlying algorithm to build a tree model either for categorical or numerical response variables, its implementation in `R` and some recent improvements.

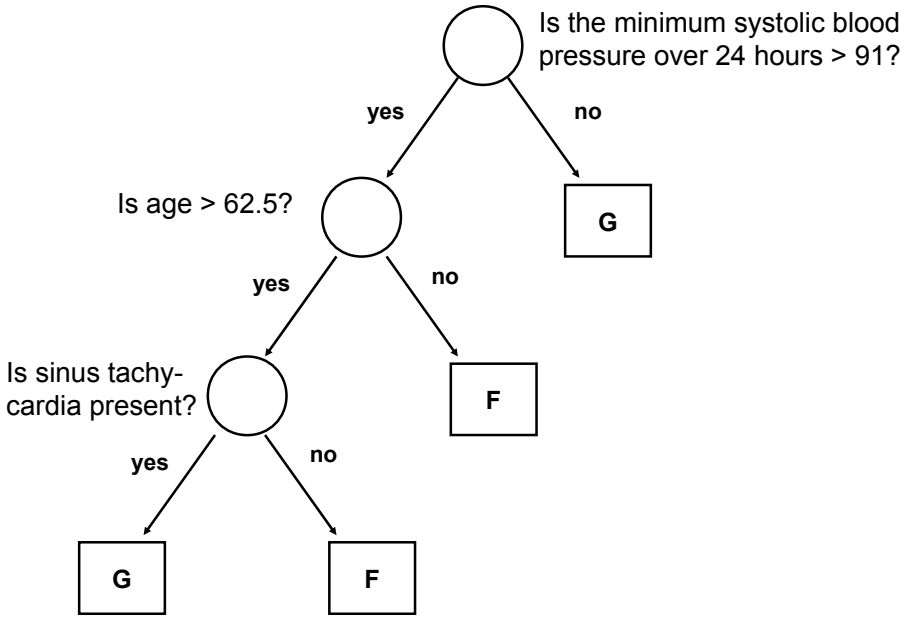

Figure 2.1: A tree structured classification rule for identifying high risk heart attack patients.

2.1 Binary recursive partitioning

We assume that there are n observations, p predictors $x_1, ..., x_p$ and a corresponding response variable y. The algorithm used to build a corresponding classification tree is called *binary recursive partitioning*. Our discussion follows the overview of CART models in [2] Berk (2008) and their introduction in [6] Breiman et al. (1984).

A tree model systematically partitions the space spanned by all the predictors in a stagewise procedure into smaller and smaller subspaces until a certain stopping criterion is met and then assignes a prediction value to the observations in the last subspace. This is either a predicted class for classification or a simple constant value for regression, but we will deal with numerical responses later.

At first we focus on the classification case with two classes, i.e. $y \in \{0, 1\}$. For example for modelling certain kinds of machine problems the class labels can represent erroneous and not erroneous output if no further information about type or degree of the fault is available.

At the beginning, the whole dataset $\mathcal{X}$ is analyzed. The procedure now looks for the best way to split the whole dataset into two subsets $\mathcal{X}_1$ and $\mathcal{X}_2$ so that both are as class homogeneous as possible. We focus on details on this shortly. Here, splitting means finding a treshold in the values of one of the predictor variables and then partition the group of response values into two subgroups

according to that treshold. Figure 2.2 exemplarily shows this idea with two predictor variables x_1 and x_2.

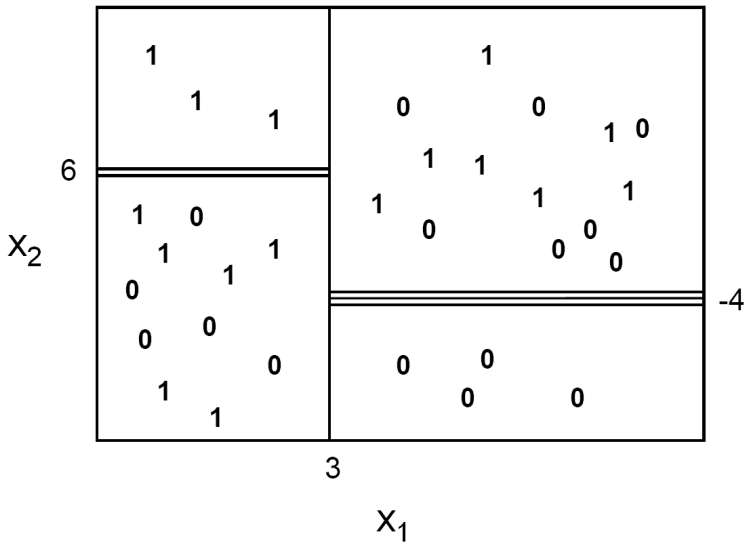

Figure 2.2: Recursive Partitioning of a binary 0/1 outcome and predictor variables x_1 and x_2.

Figure 2.2 shows a three-dimensional scatterplot with predictor variables x_1 and x_2 and the respective binary response. The vertical line at $x_1 = 3$ produces the first partition of the data. Now the values of the response y are partitioned into 2 groups and the partitioning is determined by a value of the predictor variable x_1. The second partition is produced by the double horizontal line at $x_2 = 6$ but only within the group of response values for which the predictor x_1 is smaller than 3. Finally, the third partition is produced by the triple horizontal line at $x_2 = -4$, now only within the group of response values for which $x_1 > 3$ holds.

The CART method recursively constructs a series of partitions with split lines perpendicular to the coordinate axis of the used predictors. But there is more information in figure 2.2.

The partitions in the upper left and lower right corners are fully class homogeneous. So cases with $x_1 \leq 3$ *and* $x_2 > 6$ are always of class 1 and cases with $x_1 > 3$ *and* $x_2 \leq -4$ are always of class 0. Reaching such kinds of homogenity in the constructed partitions is the main goal of the procedure and also the idea behind the partition construction itself.

As shown in the heart attack patient example the achieved partitioning of the data can be illustrated in a tree diagram. A general tree structure is shown in figure 2.3.

One follows the tree diagram from top to bottom, i.e. from the *root node* to the *terminal nodes*. The root node contains the full data sample and then the data is subsequently split using calculated values of certain predictors. Cases

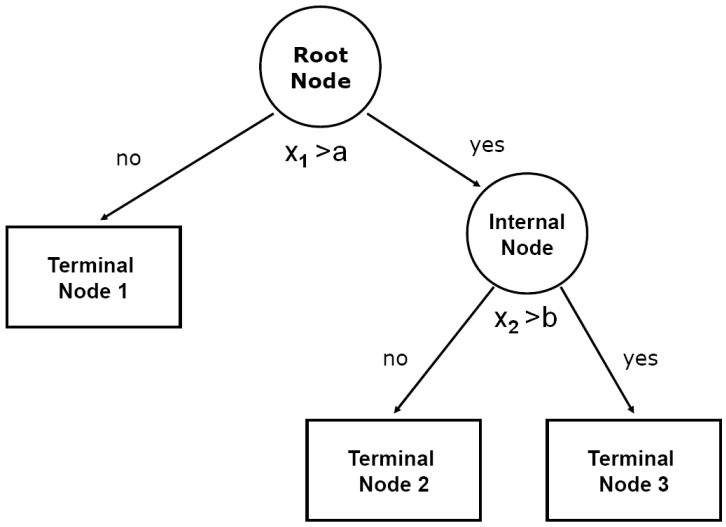

Figure 2.3: General structure of a CART model output.

with $x_1 > a$ go into the right partition and cases with $x_1 \leq a$ go left and so on.

In figure 2.3, every split beyond the first split at the root node means some sort of interaction effect between the predictors as such splits are only performed in a data region where one predictor is restricted to certain values (e.g. $x_1 > a$).

The questions that now remain to fully understand the construction of such a tree are the following:

1. How does one calculate the values of the predictors at which splits are performed?
2. How is homogeneity achieved and maximised?
3. When is a node declared terminal?

2.1.1 Splitting nodes

All of the questions above are linked to the so called *impurity* of a node. The goal is to partition the data such that the resulting partitions are as class homogeneous as possible, i.e. as *pure* as possible. The best case is a node that is fully class homogeneous, e.g. that only contains class 0 observations. The impurity of a node on the other hand is at its maximum if 50% of the observations in a node are class 0 and 50% are class 1 observations. Thus the impurity of a node depends on its class proportions. The fundamental idea behind splitting is to select a split so that the data in the resulting partitions are purer than the data in the parent partition.

Definition 2.1.1. *Let Y be a random variable. The impurity I of a node A is a function Φ of the probability $P(Y = 1|A)$, i.e.*

$$I(A) = \Phi(P(Y = 1|A)),$$

with $\Phi \geq 0, \Phi(P) = \Phi(1 - P)$ and $\Phi(0) = \Phi(1) < \Phi(P)$.

The impurity is a nonnegative symmetrical function with a minimum when the node A is fully class homogeneous and a maximum when half of the observations in A are class 0 and the other half are class 1 observations. There are several types of impurity functions (see [6] Breiman et al. (1984) for comparisons and discussion) but we focus on the *Gini index* (see figure 2.4).

Definition 2.1.2. *The Gini index impurity function is given by*

$$\Phi(P) = P(1 - P).$$

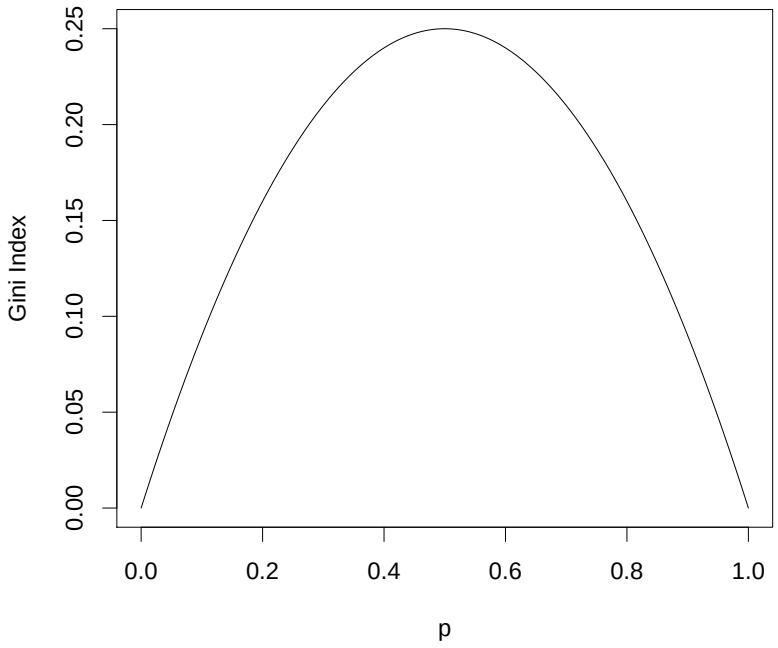

Figure 2.4: The Gini index impurity function.

The goal is to find a value in one of the predictors that partitions the data so that the resulting child partitions have an overall smaller impurity than their parent partition.

When the binary recursive partitioning procedure tries to find a value at which to split a node, it evaluates all possible split of all possible input variables

and takes the one that reduces the impurity the most. The difference in the impurity when a split s is performed on node A,

$$\Delta I(s, A) = I(A) - \Big(P(A_L)I(A_L) + P(A_R)I(A_R)\Big),$$

where $P(A_L)$ is the probability of a case falling in the left child node and $P(A_R)$ is the probability of a case falling in the right child node, is maximised, i.e.

$$\max_{s,A} \Delta I(s, A).$$

The CART procedure takes the variable and split that maximally reduces the impurity to define the new partition of the data.

There are several stop-splitting rules. The initial one is simple: when the procedure reaches a node in which no significant decrease in impurity is possible then do not split this node any further and declare it terminal.
More formal, set a treshold $\beta > 0$ and declare a node as a terminal node if

$$\max_{s,A} \Delta I(s, A) < \beta.$$

Another restriction that implies a stopping criteria is to set a minimum sample size for each terminal node, i.e. a minimum number of cases in a terminal node.

The implementation of the CART procedure in `R` provides several stop-splitting criteria, which will be discussed later when we focus especially on CART in `R`.

2.1.2 Classification and Prediction

When a tree is fully grown the terminal nodes are labeled with classes. The class assigned to a terminal node is determined by majority vote: If the majority of cases in terminal node A are of class $i, i \in \{0, 1\}$, then A is labeled as a class i terminal node.

This assignment points at the advantages of the procedure. First, with a terminal node labeled for example class 1, one can follow the built tree structure from top to bottom and see exactly under which data constellations and interactions between variables, observations are of class 1. Second, the labelling is used to predict the class of new observations with unkown response class. This observation is dropped down the tree structure and after it falls into one distinct labeled terminal node, this label becomes the class prediction of the observation.

But when certain classes are assigned to a terminal node by majority vote, errors are made if the node is not fully class homogeneous. For example if the proportion of class 0 cases in a terminal node is 0.70 this is used as an

estimation of the probability that a new case with unknown response falling into that node is of class 0. The estimated probability for not being of class 0 is 0.30 and this is also the proportion of cases of class 1 being wrongly labeled. We focus on the influence of such errors on the explanatory power of a CART model later when model quality is discussed.

In summary, a CART model is able to model and illustrate the relationship between the response y and a set of predictors $x = (x_1, ..., x_p)$ and, with an established model, it is possible to predict the outcome of a new observation with an unknown response. Furthermore estimated probabilities of prediction errors can be given.

2.1.3 Example

To simply examplify on how the procedure works in a real classification problem we use the data set `frogs` from the `R` library `DAAG`. The data set is the result of a study of ecological factors that may affect the presence of a certain frog species in New South Wales, Australia. It contains 212 rows and 11 columns.

The binary response variable is `pres.abs` where 0 means frogs were absent and 1 means frogs were present. For ease of interpretation we limit ourselves to the following 7 predictors: `altitude` (altitude in metres), `distance` (distance in metres to the nearest extand population), `NoOfPools` (number of potential breeding pools), `NoOfSites` (number of potential breeding sites within a 2 km radius), `avrain` (average rainfall during spring period), `meanmin` (average minimum spring temperature) and `meanmax` (average maximum spring temperature).

The resulting classification tree for `pres.abs` is shown in figure 2.5.

In the resulting model only two out of seven predictors were recognized to be influential, i.e. significantly reduce node impurity, namely `distance` and `meanmin`. The split that partitions the data the best is performed at a `distance` value of 625 metres. For small distances (< 625m, right tree branch), `meanmin` is important. It is split at 2.9 degrees. Two terminal nodes result. One is labeled as class 0 node with 10 observations in the partition of class 0 and 2 observations being of class 1. The node is relatively pure. The resulting misclassification rate of the terminal node is about 17% and its Gini index impurity is

$$\frac{10}{12} \cdot \left(1 - \frac{10}{12}\right) = \frac{5}{36} = 0.13888...$$

Thus, in 10 out of 12 areas (about 83%) with small distances to the nearest extand population and a low average minimum spring temperature frogs were absent. Similar conclusions can be made for the remaining terminal nodes.

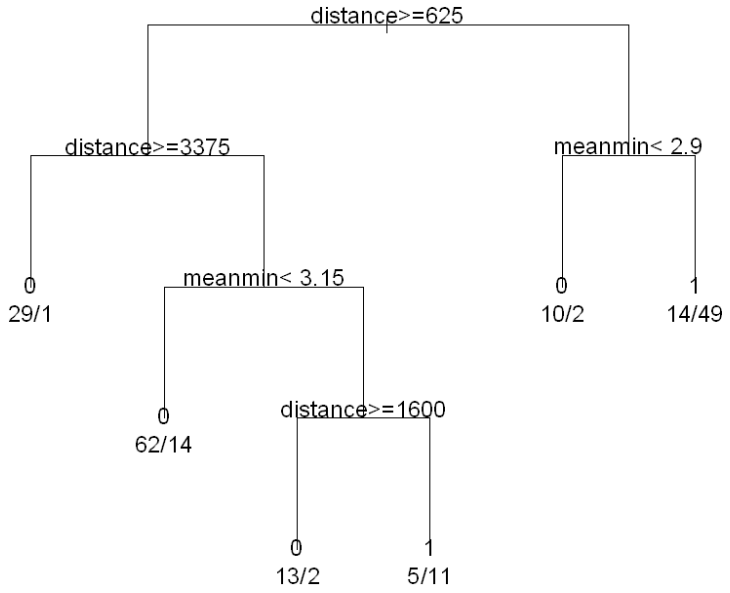

Figure 2.5: Classification tree for the binary response variable `pres.abs`.

2.2 Pruning

An overall too large and complex tree is often not desirable. Sometimes one wants to keep a tree simple because of interpretability issues or for preventing the model to focus too much on idiosyncrasies of the given modelled dataset and thus losing predictive power (this is called *overfitting*). Therefore it is important to constrain the size of a tree by removing branches that do not reduce heterogeneity sufficiently enough for the additional complexity. The strategy to reduce the tree size is called *pruning*.

The following definition mentions costs for misclassifying a case. We will not discuss the use of costs in a CART model in detail (see [2] Berk, section 3.5 for a discussion of costs). For our purposes it is sufficient to say that misclassification weights or costs can be applied to the classes, i.e. for class 0 being the positive class (e.g. not faulty) the cost of misclassifying a class 0 observation as class 1 observation (a so called *false negative*) can be higher or lower than misclassifying a class 1 as class 0 observation (*false positive*). This can be useful in some practical problems. By default, false positives and false negatives have equal costs.

Definition 2.2.1. *Let K be the number of terminal nodes in a tree model, C be the number of classes ($C = 2$ in the binary case), $\tau(A)$ be the class assigned to node A by the model if A is terminal and π_i be the prior probability of a case being in class i, i.e. the proportion of class i in the given data set. Then*

we define

- *L(i,j) as the loss matrix for incorrectly classifying a case of class i as class j. L is a 2×2 matrix in the binary case and the cost for correct classification (i.e. the entries in the main diagonal) are taken to be 0.*
- *the risk of node A as*

$$R(A) = \sum_{i=1}^{C} P(I = i|A)L(i, \tau(A))$$

  with $\tau(A)$ chosen to minimize risk.
- *the risk of the entire tree T as*

$$R(T) = \sum_{j=1}^{K} P(X \in A_j)R(A_j)$$

  with $P(X \in A_j)$ being the probability that case X falls in terminal node A_j.

The risk of a node A in the binary case describes the probability of a class 0 case falling in A times the costs that follow plus the probability of a class 1 case landing in A times the costs. The risk $R(T)$ of the entire tree is also called the *expected cost of the tree T*.

For pruning a tree, a penalty for complexity is introduced. This penalty depends on the number of terminal nodes. It comes to the minimization of

$$R_\alpha(T) = R(T) + \alpha|\tilde{T}|$$

with $\alpha \geq 0$ being the complexity parameter and $|\tilde{T}|$ is the number of terminal nodes of the tree T.

Obviously the larger the value of α the larger is the penalty of complexity, i.e. for each additional terminal node. Hence, α controls the size of a tree.

In [6], Breiman and his colleagues prove the following proposition.

Proposition 2.2.1. *Let T be a tree. For every value of α there exists a subtree $T(\alpha)$ of T with minimal cost complexity, i.e.*

$$R_\alpha(T(\alpha)) = \min R_\alpha(T).$$

2.3 Classification Model Evaluation

After fitting a classification tree model to a given data set one is interested in its performance, i.e. its power to classifiy given observations correctly and the size of the resulting model error.

2.3.1 Confusion Table

A *confusion table* is used to get an overview of how well a CART model fits the data used to build the tree (*training data*). It cross-tabulates the observed real classes of the observations with the classes assigned by the model. Thus it serves as an indicator for the goodness of fit and shows several types of errors. A confusion table can also be used to assess a CART models forecasting power by tabulating a model's results produced with data not used to build the tree but with known response classes (*test data*).

Table 2.1 shows the general structure of a confusion table for the binary classification case.

	Class 0 predicted	Class 1 predicted	Model Error
Class 0	a	b	$b/(a+b)$
Class 1	c	d	$c/(c+d)$
Use Error	$c/(a+c)$	$b/(b+d)$	Overall Error $= \frac{b+c}{a+b+c+d}$

Table 2.1: General structure of a confusion table. The letters in the cells are counts of the respective cases.

First, a confusion table calculates the overall model error as the sum of misclassified cases (off-diagonal entries) divided by the total number of observations. If $b + c = 0$ then the model provides a perfect fit with no cases being misclassified.

Second, the table gives an overview of the number of false negatives (b) and false positives (c). The fourth column shows the proportions of these two types of incorrectly classified cases. Berk [2] calls these proportions the *model error*. When the true class is known, they give information on how common it is for the procedure to fail to identify it.

The entries in the fourth row of the table show the so called *use error*. When an observation was assigned a specific class by the model, these proportions give information on how common it is for this assignment to be wrong.

Furthermore, the proportions that mirror successful predictions can also be viewed. The values $a/(a+b)$ and $d/(c+d)$ are called *sensitivity* and *specificity* (see [1] Altman & Bland, 1994) and are considered in some applications (see for example [13] Hastie et al. 2001, section 9.2.5).

Confusion tables serve as an essential diagnostic tool and should be considered for every classification problem.

2.3.2 Cohen's Kappa

Another possibility to describe the overall rate of class agreement of a classification model is given by *Cohen's Kappa* (see [7] Cohen, 1960). It is used

to measure the agreement of two individual classification procedures. In our case we are interested in measuring the agreement between the observed class labels in our training sample and the class labels assigned by a classification tree model.

The kappa coefficient depends on the values of a confusion table of proportions as shown in table 2.2.

	Class 0 predicted	Class 1 predicted	Total
Class 0	p_{11}	p_{12}	$p_{1.}$
Class 1	p_{21}	p_{22}	$p_{2.}$
Total	$p_{.1}$	$p_{.2}$	1

Table 2.2: Confusion table of proportions. The value p_{ii} is the number of observations in cell (i, i) divided by the total number of observations.

At first the observed level of agreement is computed:

$$p_o = p_{11} + p_{22}$$

This will be compared to

$$p_e = p_{.1}p_{1.} + p_{.2}p_{2.}$$

The Kappa coefficient is then defined as

$$\kappa = \frac{p_o - p_e}{1 - p_e}.$$

The value of κ is always less or equal to 1 where 1 implies perfect agreement. Sometimes it can even be negative which means very poor agreement. There are different ways to interpret the values of Kappa. One possible interpretation that seems useful is provided below (see e.g. [17] Landis & Koch (1977)).

- less than 0.20 = poor agreement
- 0.21 - 0.40 = fair agreement
- 0.41 - 0.60 = moderate agreement
- 0.61 - 0.80 = good agreement
- 0.81 - 1.00 = very good to (almost) perfect agreement

	Absence predicted	Presence predicted	Model Error
Absence	114	19	0.14
Presence	19	60	0.24
Use Error	0.14	0.24	0.18

Table 2.3: Confusion table for modelling the distribution of certain frogs in New South Wales, Australia.

2.3.3 Example

We continue our example using the `frogs` data set to evaluate the classification tree model whose graphical output is shown in figure 2.5. Table 2.3 shows the resulting confusion table.

The model seems to perform quite good with an overall classification error of about 18%. However, if frogs are present the procedure fails to correctly identify them in about 1 out of 4 cases (model error of 0.24). Furthermore, if presence of frogs is predicted, on average about 1 out of 4 predictions is wrong (use error of 0.24).

	Absence predicted	Presence predicted	Model Error
Absence	0.5377	0.0896	0.6274
Presence	0.0896	0.2830	0.3726
Use Error	0.6274	0.3726	1

Table 2.4: Confusion table of proportions for modelling the distribution of certain frogs in New South Wales, Australia.

A look at Cohen's kappa coefficient gives further information. The related confusion table of proportions is shown in table 2.4. This results in a kappa coefficient of about 0.62 which means good agreement.

In summary, the model seems quite acceptable.

2.4 Regression Trees

We will now focus on numerical response variables. The key difference between the classification and the regression case lies in the splitting criterion.

Our data again consists of p input variables and a numerical response variable y for N observations, so we have (x_i, y_i) for $i = 1, ..., N$ and $x_i = (x_{i1}, ..., x_{ip})$. In the classification case we had to maximize the reduction in the impurity, thus CART had to solve

$$\max_s \Delta I(s, A) = \max_{A_L, A_R} \Big(I(A) - \Big(P(A_L) I(A_L) + P(A_R) I(A_R) \Big) \Big).$$

Exactly the same concept applies for the regression case, with the impurity of a node now being the sum of squares in that node, i.e.

$$I(A) = \sum_{i=1}^{N}(y_i - \bar{y}(A))^2.$$

For a node A the deviance is given by

$$D(\tilde{x}, A) = \sum_{i=1}^{n}(y_i - \bar{y}(A))^2 - \Big(\sum_{i=1}^{n_{\tilde{x},A}}(y_i - \bar{y}_{1,\tilde{x}}(A))^2 + \sum_{i=n_{\tilde{x}+1,A}}^{n}(y_i - \bar{y}_{2,\tilde{x}}(A))^2 \Big)$$

with $\tilde{x}$ being the treshold at which a split is performed. So we seek

$$\max_{\tilde{x}} D(\tilde{x}, A).$$

Again each observation is placed in one terminal node and is then assigned with the mean of all values in that node. These conditional means serve as fitted values for observations out of the training data and as predicted values for new observations out of test data.

As there is no confusion table, the quality of fit is based on a summary statistic such as the root mean squared error.

2.4.1 Formal representation

A given regression tree also has a formal representation (see [13] Hastie et al. 2001).

Suppose we have partitioned our data space into J regions $R_1, ..., R_J$, thus R_j describes the parameter space region defined by the jth terminal node, $j = 1, ..., J$. Then a regression tree models the response y as constant in each region. Thus, a tree T can be represented as

$$T(x, \Theta) = \sum_{j=1}^{J} \gamma_j 1_{\{x \in R_j\}}$$

with γ_j being the conditional mean of the response values that is assigned to terminal node j and tree parameters $\Theta = \{R_j, \gamma_j\}$. Let $f(x)$ be the estimated relationship between predictors x and response y, then

$$x \in R_j \Rightarrow f(x) = \gamma_j.$$

With this a regression tree model can be formulated as

$$Y = \sum_{j=1}^{J} \gamma_j 1_{\{x \in R_j\}} + \epsilon,$$

where ϵ is zero-mean noise.

2.5 Advantages and Limitations of CART

One advantage of the CART procedure is the possibility to show its output in a tree diagram. The tree plot delivers a highly intuitive insight in the structure of the data and shows effects of interactions in a natural way. With a given tree structure one is also able to determine the most influential predictors. The higher above in the tree structure a certain predictor is used for a split the more reduction it yields in impurity or deviance and thus the more important it is for describing the relationship with the response.

As outlined in [6] Breiman et al. section 2.7, some particular advantages of the CART procedure are:

- It is very flexible and can be applied to any given data structure, i.e. it does not rely on any assumptions about the distribution of the data as it is non-parametric.
- It handles both numerical and categorical variables in a simple way.
- The final output has a simple and easy to interpret form and can be used for predicting new data.
- The tree structure implicitly gives a variable importance ranking and variable selection.
- The procedure also provides error estimates.
- It is very robust with respect to outliers.

However, there are also some shortcomings to deal with. First, there are no formal mathematical results indicating that a CART model will find the correct relationship between predictors and response from a given sample, even with all predictors provided and well measured. For example a suboptimal split at one level might allow better splits on lower levels that would not be performed otherwise and thus such a split might result in a better overall tree structure.

Also, there is evidence that the procedure selects predictors in a way that injects bias. The method tends to favour predictors with a larger number of distinct predictor values.

Another problem of CART can be *overfitting*. As mentioned earlier, flexible statistical modelling procedures such as CART sometimes tend to focus too

much on idiosyncrasies of the given training data and thus they are too dataset specific. Hence, the outcome often does not generalize well to new data and predictive power is lost. This problem is mostly visible in unpruned trees and trees where every observation has its own terminal node (a tree with as many terminal nodes as observations is called *saturated tree*). Thus pruning and tree size limitations can help to prevent overfitting. Further enhancements of the CART methodology which are discussed later are also able to deal with the problem.

As discussed in [13] Hastie et al. (2001, p. 272-4) a major problem of a tree is its high variance. Small changes in the data can result in different splits and thus in a different tree with different terminal nodes. This should be kept in mind when interpreting tree results. The variance is basically due to the hierarchical tree structure, as changes in data and errors in the top split affect all splits below i.e. are carried down.

Furthermore, a small number of observations in a node can cause instability in the results.

2.6 Unbiased Binary Recursive Partitioning

As mentioned above, one problem of the CART method is its selection bias towards predictor variables with many possible splits. A solution to this problem is presented in the work of Hothorn and his colleagues ([14] Hothorn et al. 2006). They presented a unified framework for an unbiased recursive partitioning procedure based on hypothesis testing (conditional inference).

Their key idea for constructing interpretable tree structures not suffering of bias toward certain predictors revolves around the separation of variable selection for a split and the splitting procedure itself. The selection of a variable at which to perform a split is determined by the results of a hypothesis test, i.e. a test of independence between any of the covariates and the response is performed.

The procedure is summarized as follows (see [2], p. 137-8):

1. Test the global null hypothesis of independence between any of the p predictors and the response.
2. Stop if the test cannot be rejected.
3. If the test is rejected select the predictor with the strongest relationship to the response as the splitting variable.
4. Choose the best split as usual using only the selected predictor.
5. Iterate steps 1-4 until no further splits are indicated.

As mentioned above the variable selection and the actual splitting procedure is separated into steps 3 and 4.

All of the performed hypothesis tests are based on permutation distributions where values of the response variables are randomly shuffled. For each predictor such a *permutation test* is performed under the null hypothesis of no association. Thus a p-value follows for every predictor variable and an overall p-value is also computed using methods of correction of multiple comparisons (e.g. Bonferroni correction).

If the global null hypothesis is rejected the predictor with the strongest relationship to the response is chosen (step 3). Here, 'strongest relationship' means having the smallest p-value out of the single permutation tests.

Once a predictor is selected this way, a split is performed as described in the above chapters or by utilizing the permutation test framework from above (see [14] for details).

Hothorn and his colleagues also showed that such a procedure results in a predictive performance similar to what can be found in optimally pruned trees and therefore they implicitly offer a possible solution to the overfitting problem.

However, there are still some minor concerns. For example, it is still unclear whether the method estimates the real relationship between the response and the predictors, if there is any. Results also depend on the sample size. See [2] for details.

A description of the method in `R` as well as its application on the `frogs` example is presented in the following section.

2.7 CART in R

The initial CART methodology described earlier as well as the idea of unbiased recursive partitioning have its implementations in `R`. The ideas of Breiman and his colleagues were transformed to `R` and can be found in the `rpart` package (see [31] Therneau & Atkinson (1997) for a description). The work of Hothorn et al. is implemented in the `party` package. There are several more implementations of tree models but we focus on these two.

2.7.1 The rpart package

`rpart` is an abbreviation for **r**ecursive **part**itioning and has to be loaded as a separate `R` package. With it the user can fit tree models following the methodology described above.

Again, we use our example of the `frogs` data set and describe how the classification tree shown in figure 2.5 is built using `rpart`. The main command to build a tree model is `rpart`. With

```
> xtrain<-data.frame(altitude, distance, NoOfPools, NoOfSites,
  avrain, meanmin, meanmax)
> library(rpart)
> mod<-rpart(pres.abs~., data=xtrain, method="class")
```

one fits a CART model to estimate the relationship between the response variable `pres.abs` and all of the columns of a previously defined data frame of predictors stored in `xtrain`. The notation is equal to `lm` and other model fitting functions. The resulting `rpart` object is stored in `mod`. Generally, the response can be either categorical (i.e. a factor, which should be declared explicitly using `as.factor()`) or numerical as can be the predictors in `xtrain`. Depending on the type of the response the routine automatically distinguishes between a classification tree or a regression tree. One can also explicitly tell the routine which kind of tree model to fit via `method`.

The `rpart` command takes further arguments, some of which are discussed below (see [31] Therneau & Atkinson (1997) and the description in the `R` help by Ripley for more).

- `method`: explicitly specifies which kind of tree to be build. THe first option `method="class"` builds a classification tree, `method="anova"` results in a regression tree. If not specified, the routine guesses the right method based on the type of the response.
- `parms`: lets the user specify optional parameters for the splitting function. A regression tree does not have further parameters, for classification splitting a list can contain a vector of prior probabilities of a case being in each class (component `prior`). They must be positive and sum up to 1 and the defaults are proportional to the data counts. With the `loss` component one can specifiy a loss matrix which must have zeros on the main diagonal and positive off-diagonal elements. The `split` component specifies the split criterion to be used. It defaults to `gini`.
- `cost`: a vector of non-negative costs for variables in the model. These are scalings for the variables to be applied when splits are determined. The improvement on splitting on a variable is divided by its cost in order to decide which split to choose. With this, the user can specify if certain variables are more important than others based on previous knowledge.
- `control`: here further options that control details of the `rpart` can be given as an object that is specified before one executes `rpart`.

The details that can be specified in the control object are the following:

- `minsplit`: the minimum number of observations in a node for which the routine will even try to compute a split. The default value is 20.

- `minbucket`: the minimum number of observations in a terminal node. It defaults to `minsplit`/3.
- `maxcompete`: controls the number of competitor splits to be printed. For example sometimes it is useful to see not only the variable that gave the best split but also the second, third etc. best.
- `maxdepth`: controls the maximum interaction depth level of a tree with the root node counted as depth 0.
- `cp`: sets the complexity parameter for pruning. A value of `cp`=1 will always result in a tree with no splits. For the classification this corresponds to the complexity parameter α as described above. For regression models it has a direct interpretation: if any split does not increase the overall R^2 of the model by at least `cp` (with R^2 defined as coefficient of determination for linear models) then that split is not considered and that branch is not split any further. The default value is 0.01.

The control object can be passed to the `rpart` command in the following way:

```
> modControl<-rpart.control(minsplit=20,
  minbucket=round(minsplit/3), maxcompete=3, maxdepth=5, cp=0.01)
> mod<-rpart(pres.abs~., data=xtrain, control=modControl)
```

As mentioned above, the parameters `minsplit`, `minbucket` and `maxdepth` have influence on the size of the tree in addition to what is specified in `cp`.

When a `rpart` model object is generated a tree output can be plotted by simply using the plot command. Furthermore a plot needs to be texted with the respective variable names using the `text` command. The plot in figure 2.5 on page 10 is generated with

```
> plot(mod, uniform=TRUE)
> text(mod, use.n=TRUE, all=FALSE)
```

With this a tree is plotted in a separate `R` plot window as usual. Basic plot parameters like changing text size or adjusting margins can also be applied as usual by using `par` commands.

2.7.2 The party package

The second `R` package suitable for fitting tree based classification and regression models is the `party` package for constructing conditional inference trees and more. The core function of the package is `ctree` for constructing conditional inference tree models according to the unbiased recursive partitioning methodology described earlier. It basically has the same syntax as `rpart`. With

```
> library(party)
> mod<-ctree(pres.abs~., data=xtrain)
```

a conditional inference tree model object for the `frogs` data example is created (with `xtrain` as before). The `plot` command plots the model.

```
 > plot(mod)
```

Here no texting is required. The resulting plot for our `frogs` example is shown in figure 2.6.

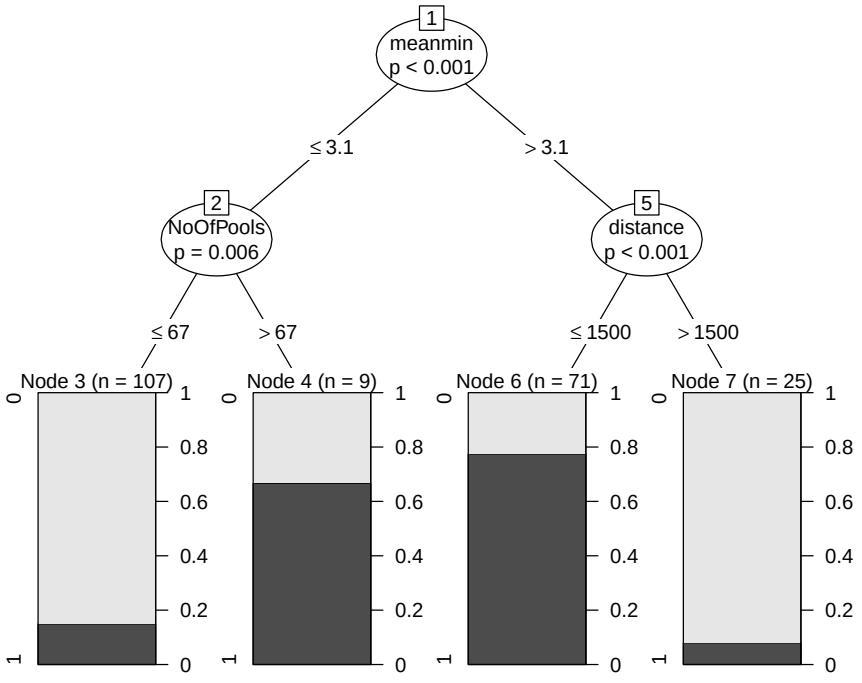

Figure 2.6: Classification tree using unbiased binary recursive partitioning for `pres.abs`.

The variable ranking resulting from the `party` procedure differs slightly from its `rpart` counterpart. It also recognizes `NoOfPools` as important. Furthermore, the corresponding p values are shown in every node.

The plot also graphically shows the distribution of the response values in every terminal node by various plot types. Therefore the package's plot method can be extended by so called panel functions. The most important panel methods are:

```
> plot(mod, terminal_panel = node_terminal(mod))
```

```
> plot(mod, terminal_panel = node_boxplot(mod))
> plot(mod, terminal_panel = node_hist(mod, freq=T,
  horizontal=F))
> plot(mod, terminal_panel = node_density(mod, rug =T,
  horizontal=F))
```

From top to bottom they are: an ordinary terminal node representation with the number of observations and the mean of each node shown (`node_terminal`), a boxplot of the observations in the terminal node (`node_boxplot`), a histogram ((`node_hist`)) either with frequency or proportions as in the usual `hist` command and either rotated 90 degrees (`horizontal=T`) or as usual (`horizontal = F`) and an estimated density curve (`node_density`) either horizontal or not and with (`rug=T`) or without (`rug=F`) rug representation showing the distribution of the response values as dots on a horizontal axis at the bottom of the plot.

For controling the size of a `ctree` tree model, `minsplit`, `minbucket` and `maxdepth` can be used as in `rpart` with a `ctree_control` object. In addition, the parameter `stump=T` returns a tree with only 3 nodes (a so called "stump", a root node and one split). With parameters `teststat` and `testtype` one can also specify the type of the test statistic and how to compute its distribution. With `mincriterion` one can then specify the value of the test statistic or $1 - p$–value that must be exceeded in order for a split to be performed. See [14] Hothorn et al. (2006) for details on this.

2.8 Further Literature

After the initial publication of the theoretical framework of CART in [6] research on the introduced models started slowly. A new approach to the idea was given in 1993. The procedure of recursively partitioning the observations by univariate splits is also used to derive a procedure different to CART called C4.5 (see [21] Quinlan, 1993). It also performs an exhaustive search over all possible splits and maximizes some information measure to achieve the best partitioning but unlike in CART a C4.5 model is not limited to binary splitting and therefore also allows multiple splits in the data.

As computer power generally increased so did research on the matter with particular focus on node splitting and variable selection. The problem of bias in variable selection was already known and discussed by Loh and Shih in their work on split selection methods (see [20] Loh & Shih (1997) and [19] Loh (2002)). Algorithms for tree construction called QUEST (for classification) and GUIDE (for regression) are proposed to specifically minimize or eliminate variable selection bias.

For classification trees, [26] Shih (1999) summarizes splitting criteria in families and shows that the best splits based on split criteria are based on parameters of the proposed families.

Furthermore, [30] Su (2002) published a method for growing regression trees under maximum likelihood aspects. Standard likelihood methods such as likelihood ratio tests are incorporated into each stage of the tree procedure.

Following the earlier work of Loh and Shih, the group around Torsten Hothorn published several articels on their ideas of conditional inference trees and tree-based models. Hothorn and colleagues [33] (2008) published a discussion on model based recursive partitioning, linking tree models with parametrical models. In newer versions of the `party` package, these ideas are also implemented. Furthermore, [28] Strobl (2006) focuses specifically on unbiased variable selection in classification trees with the Gini impurity function and discusses some distributional characteristics.

Chapter 3

Tree-based Methods

We now turn to methods that use the results of several classification or regression tree models in order to classify or regress. Therefore we make a transition from focussing on procedures that produce a single set of results to methods which use many sets of results to produce their output.

The idea of aggregating results of several models does have benefits. For example the problem of overfitting the data can be averaged out as averaging tends to balance out effects that come from idiosyncratic features of a given data set. Furthermore it can reduce the variance that comes with a single tree model and thus can produce more accurate and stable models. Thus this approach also offers a solution to CART's instability problems while building on the flexibility of the method. This is particularly beneficial if one is interested in predicting the outcome of new data.

An alternative approach to enhance the predictive power is by constantly improving the results of a single model over a series of steps, i.e. using some sort of additive model. This approach can also take advantage of the flexibility of a CART model and deal with some of its problems.

For modelling maintenance problems and defects occuring during wafer production we use two such non-parametric methods based on the outcomes of several CART models, namely *Random Forests* introduced by [4] Breiman (2001) and *Boosting* whose idea goes back to [24] Schapire (1990). These methods have proven to be very accurate for modelling both categorical and numerical response variables while having the above discussed advantages over CART.

3.1 Random Forests

A Random Forest consists of an ensemble (or a *forest*) of CART models. The method builds on the idea of Bagging (see [3] Breiman, 1996) which stands for "**B**ootstrap **Agg**regat**ing**". With Bagging Breiman first used the approach

of aggregating results of several tree models. It attempts to reduce the variance of CART's fitted values in that it takes bootstrap samples of the given observations, fits a large number of tree models from each sample and then aggregates the results. For classification the result for one case is achieved using majority vote with one vote per tree over all constructed trees in the forest. For regression, a result for one case is achieved by averaging over the outcomes of all trees. With this the prediction error is usually improved over CART. We will not discuss Bagging any further. For our purposes it is enough to say that Bagging forms the ideological foundation of Random Forests.

3.1.1 The basic algorithm

The basic idea of a Random Forest is in part already outlined above. The procedure builds a large number of tree models, either classification trees for classification problems or regression trees for regression problems. As in Bagging, a Random Forest does not use all of the data cases for constructing each tree but only a bootstrap sample for each tree. This is one reason for the term "random". The randomness comes on the one hand from taking not all but only bootstrap samples of the data to build each tree and on the other hand from using not all but only a random sample of predictors for determining each split in each tree.

The method is also able to measure its own performance by using the samples held back from constructing or "training" a tree, i.e. the data cases not selected by the bootstrapping. It uses them as "test" data or *out-of-bag* (OOB) samples to test the model's predictive power and calculate error rates (*OOB error*).

In the following we will focus on the regression case but, as with CART, the classification methodology is quite similar.

The basic steps of the Random Forest algorithm for the regression case are the following (see [2] Berk, 2008):

Let the response be numerical, N be the number of given observations and m_{try} be the number of predictors used for each split in each tree.

1. Draw a bootstrap sample of size N from the data (random sample drawn with replacement).

2. Take a random sample of size m_{try} without replacement of the predictors.

3. Construct the first regression tree partition of the data, i.e. the first split and repeat step 2 for each subsequent split in the tree. Do not prune.

4. Drop the OOB data down the tree and store the assigned value, i.e. the mean of the terminal node in which the observation falls.

5. Iterate the steps 1 to 4 a large number of times, e.g. 500.

6. Use only the predicted values assigned to each observation when that observation was an OOB observation (i.e. not used to build the tree) to calculate the MSE.

Because of aggregating results the procedure reduces variance and produces more stable models. Furthermore the method does not overfit due to the law of large numbers as is proved in [4] Breiman (2001).

Using not all but only a random sample of predictors to determine each split is the main difference between Random Forests and Bagging. The additional randomness yields fitted values that are more independent (see [2] Berk, 2008, p. 194-5). When averaging results over a large number of more independent trees, the accuracy of the outcome can be higher and thus the performance gains can be more dramatic.

The random selection makes Random Forests be able to deal with a large number of predictors. As it works with a large number of trees they all have a chance to contribute anyway. It is even possible to have more predictors than observations in a Random Forest model, i.e. $p > N$. Furthermore the randomness allows that more different predictors are able to contribute to splits than in CART. With a large number of trees each predictor is evaluated at least several times and therefore will have several opportunities to define a split with relatively few competitors compared to a single CART procedure. Variables that play a certain role and create somewhat different partitions that otherwise would not have been considered are able to participate. Thus more information from the given predictors is brought into the fitting process.

A big disadvantage of the method is that compared to CART there is no graphical output that visualizes all the results and lets the user identify variable importance ranking, possible interactions and the general data structure in one interpretable tree. Although there are several graphical methods that aim to compensate this drawback, the procedure remains a black box.

3.1.2 The out-of-bag data

When drawing a bootstrap sample of size N with replacement from the data, on average about one third of the samples are left out, i.e. not used to build the corresponding tree as stated in [4] Breiman (2001). Each of the left out observations or OOB samples is then used to test each tree to get internal estimations of the model error. On average each data point is among the out-of-bag sample around 36% of the time as mentioned in [18] Liaw and Wiener (2002).

With these OOB cases there is no immediate need for cross-validation or a separate set of test data to get an estimate of the test set error. For each tree the respective OOB cases are dropped down and the estimated response is compared to the observed response value. With this one gets an unbiased internal estimate of the prediction error. Breiman argues that the OOB test

set error is as good as the error estimation produced with a separate test data set (see [4] p. 11). Furthermore, Breiman explains that the prediction error observed using OOB cases approaches the true prediction error as the number of trees goes to infinity.

3.1.3 Variable Importance

There are several different approaches as to how to measure variable importance in a Random Forest. Following the discussions in [2] Berk (2008), [23] Sandri & Zuccolotto (2006) and [4] Breiman (2001, p. 23-4) the following measures for determining variable importance in the regression case are common.

- **Measure 1**: Use the decrease in the fitting measure to determine the contribution to a model, i.e. measure the reduction in the deviance, each time the predictor is used to define a split. The sum of these reductions can then be used as importance measure for that particular tree. Then the average of all reductions over all tree models in the random forest describes the predictor's importance. The importance measure $\text{Imp}^{(1)}$ for the predictor x_i is therefore defined as

$$\text{Imp}_{x_i}^{(1)} = \frac{1}{k} \sum_{A} d(x_i, A) 1_{\{x_i \in A\}}$$

  where A is a node in each tree, $d(x_i, A)$ is the reduction in the deviance induced x_i at node A and $1_{\{x_i \in A\}}$ is an indicator function which is equal to 1 if x_i is selected for a split at node A.

- **Measure 2**: In every grown tree in the forest, the out-of-bag data cases are dropped down and the mean squared error is computed to assess the model error. Then the values of predictor i are randomly shuffled and the out-of-bag cases are dropped down again. The shuffled predictor should now be on average unrelated to the response. Iterate this procedure for each of the p predictors and compute

$$\text{Imp}_{x_i}^{(2)} = \frac{1}{K} \sum_{k=1}^{K} (\text{MSE}_i^{(k)} - \text{MSE}^{(k)}), \quad i = 1, ..., p$$

  with K being the number of trees, $\text{MSE}_i^{(k)}$ is the mean squared error of the kth tree calculated using out-of-bag data when the ith predictor values are shuffled and $\text{MSE}^{(k)}$ is the general mean squared error of the kth tree without shuffling.

  If desirable, one can normalize $\text{Imp}_{x_i}^{(2)}$ with the standard deviation of the differences:

$$\text{Imp}_{x_i}^{(2)*} = \frac{\text{Imp}_{x_i}^{(2)}}{sd(\{MSE_i^{(k)} - MSE^{(k)}\}_{k=1}^{K})},$$

with the division not being done if the standard deviation is 0.

As observed in [29] Strobl et al. (2007), the first measure $\text{Imp}^{(1)}$ is biased towards predictors with many possible splits while the permutation measure $\text{Imp}^{(2)}$ is a more reliable indicator. However, in general it is recommended to look at both of the measures, keeping the shortcomings in mind.

3.1.4 Partial Dependence

When the most important predictors are identified via variable importance analysis, one is interested in the relationship between each predictor and the response variable. A useful way to examine this are so called *partial dependence plots* or *marginal effect plots* as discussed for example in [13] Hastie et al. (2001, section 10.13.2), [2] Berk (2008, section 5.7) or [27] Siroky (2009, section 2).

A partial dependence plot for a model with continuous response is constructed as follows:

1. Grow a random forest as usual.
2. Let x_1 be the predictor of interest with ν distinct values ($\nu \leq N$) in the training set.
3. Now for each of the ν values of x_1 construct a new data set where x_1 only takes on that value while leaving all other predictors untouched.
4. Predict the response using random forests for each of the ν data sets. Average over each of the ν sets of response predictions to get ν single predicted values.
5. Plot the average prediction against the ν values of x_1.
6. Repeat the steps 2 to 5 for each predictor of interest.

The result of this procedure is a two-dimensional plot that shows how the given predictor is related to the response averaged within the values of the other predictors. A drawback of this approach is that interaction effects can not be represented unless the corresponding interaction variable is constructed in advance and used as a predictor.

3.1.5 Example of Random Forests in R

For fitting Random Forest models for both classification and regression the above methodology has its `R` implementation in the `randomForest` package by Andy Liaw and Matthew Weiner (see [18] for an overview). The package is based on the contribution of [5] Breiman and Cutler (2005).

The main command for fitting a model and creating a Random Forest object is `randomForest`. We again use the `frogs` data example to outline the `R` approach.

```
> library(randomForest)
> set.seed(101)
> mod<-randomForest(pres.abs~., data=xtrain)
```

saves a Random Forest model for `pres.abs` and predictor data frame `xtrain` containing 7 predictors under the name `mod`. As in `rpart` the routine automatically fits a classification or a regression model, based on the type of the response variable. The above `randomForest` object has the following output.

```
Call:
 randomForest(formula = pres.abs ~ ., data = xtrain)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 20.28%
Confusion matrix:
    0  1 class.error
0 116 17   0.1278195
1  26 53   0.3291139
```

In this case, the results of the random forest model are similar to the comparable `rpart` model which leads to the matrix values $(114, 19, 19, 60)$.

The most important additional arguments are:

- `ntree`: specifies the number of trees in a forest. The default value is 500. The number of trees necessary for good performance grows with the number of predictors and predictions should be compared with models of different size to determine the right value of `ntree`.
- `mtry`: the number of variables randomly selected as candidates for each split in each of the `ntree` trees. The default value is $\sqrt{p}$ for classification and $p/3$ for regression with p being the number of predictors (i.e. the number of columns in `xtrain`).

- `importance`: a logical argument, specifies if importance should be calculated for each predictor during the fit. A ranking plot of the variable importance can then be viewed with the `varImpPlot()` (e.g. `varImpPlot(mod)`) which shows the measures $\text{Imp}^{(2)*}$ and $\text{Imp}^{(1)}$ in a dotchart.

- `nodesize`: specifies the minimum size of the terminal nodes of each tree. The default is 1 for classification and 5 for regression. With a larger value, trees in the forest will get smaller.

- `maxnodes`: the maximum number of terminal nodes of trees in a forest. If not specified, every tree is grown as large as possible (without pruning).

A created object of class `randomForest` then has several components that can be addressed with `$` (e.g. `mod$...`):

- `type`: the type of the response variable as determined by the procedure i.e. `regression`, `classification` or `unsupervised` (if a response is omitted).

- `oob.times`: number of times each case is used as out-of-bag data and thus used in computing the OOB error estimate. The number should be around 36% of `ntree` as stated in [18].

- `mse`: for the regression case only, this gives a vector of length `ntree` of the mean squared errors for each tree.

- `rsq`: for the regression case only, this gives an R^2 value where

$$R^2 = 1 - \frac{\text{MSE}}{\text{Var(y)}}$$

- `confusion`: for the classification case only, this gives a confusion matrix of the prediction based on OOB data.

- `err.rate`: for the classification case only, this gives a matrix with `ntree` rows and #`classes`+1 columns with error rates of the prediction on the input data. The ith row describes the OOB error (column 1) and the error rates in classifying each class.

Regarding the default values for `mtry` and `ntree`, [12] Genuer et al. (2008) present some important observations made with a number of different datasets. As for standard regression with $N >> p$, the OOB error is maximal for `mtry`=1 and then mostly decreases quickly and as soon as `mtry` $> \sqrt{p}$ it remains the same. The choice of `mtry` $= \sqrt{p}$ always gives a lower OOB error than `mtry` $= p/3$, thus the default value for the regression case proposed by the `R` package seems to be suboptimal for some data sets. Furthermore, the default value of `ntree`=500 turns out to be convenient but a much smaller value of

`ntree`=100 often leads to comparable results. For high dimensional problems ($N < p$) the default of `mtry` $= p/3$ always gives better results than `mtry` $= \sqrt{p}$.

For the standard classification case with $N >> p$ the default values seem quite optimal but for the high dimensional case ($N < p$) it may be worthwhile to choose `mtry` larger than the default $\sqrt{p}$.

The optimal value of `mtry` that yields the lowest OOB error estimate can be found with the command `tuneRF`. With

```
> mtry.array<-tuneRF(xtrain, pres.abs, mtryStart=1,
  stepFactor=2, ntreeTry=500, improve=0.01)
> best.m<-mtry.array[mtry.array[,2]==min(mtry.array[,2]), 1]
> best.m
 [1] 1
```

the OOB error estimates for different random forest models are calculated. It starts with `mtry`=1, inflates `mtry` by `stepFactor` at each iteration and generates a forest with `ntreeTry` trees. The `improve` value then specifies the relative improvement in the OOB error that must be reached at least in order for the search to continue. Here, `tuneRF` saves a 2 column matrix of the results in `mtry.array` where the values of `mtry` are in the first and the corresponding error rates in the second column. The optimal value is stored in `best.m`. In our `frogs` example, the best value of `mtry` is 1. This leads to a slightly improved error rate of 19.8%.

The plot command for a `randomForest` object plots the error rates as stored in the `err.rate` component. In the binary classification case, three curves are presented, one for each error rate (OOB and class errors). In regression, the mean squared errors for each tree are plotted as they are stored in the `mse` component. With this plot one can determine the optimal number of tree models to use in a random forest fit. The resulting plot is sometimes called *performance plot*.

As already mentioned, the command `varImpPlot()` plots two different variable importance measures of a `randomForest` object in a dotchart while the command `importance()` prints the numerical values of the importance measures. The variable importance plot for our frog population example is shown in figure 3.1.

A partial dependence plot is generated with `partialPlot()`. With the command

```
> partialPlot(mod, xtrain, x1)
```

a two-dimensional plot is generated showing the marginal effect of the predictor `x1` out of the data frame `xtrain` (usually the training data used to construct the model) for the previously generated random forest model `mod`.

For our `frogs` example, the partial plots for `distance` and `meanmin` are shown in figure 3.2.

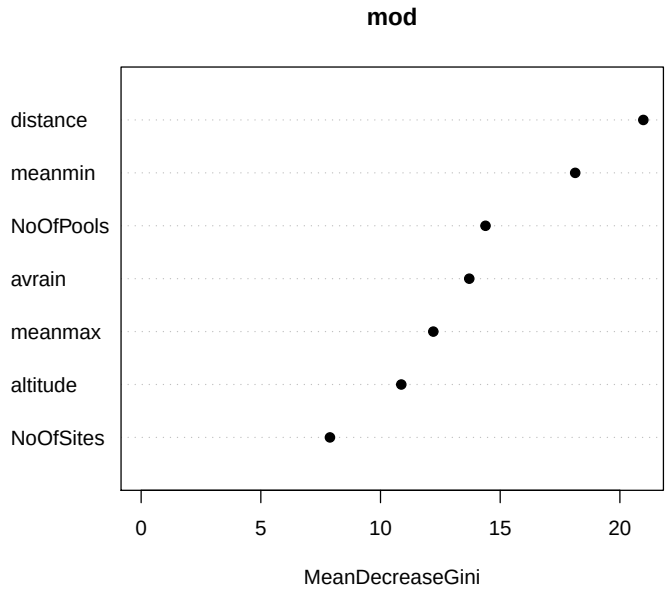

Figure 3.1: Variable importance plot of a random forest model for `pres.abs`.

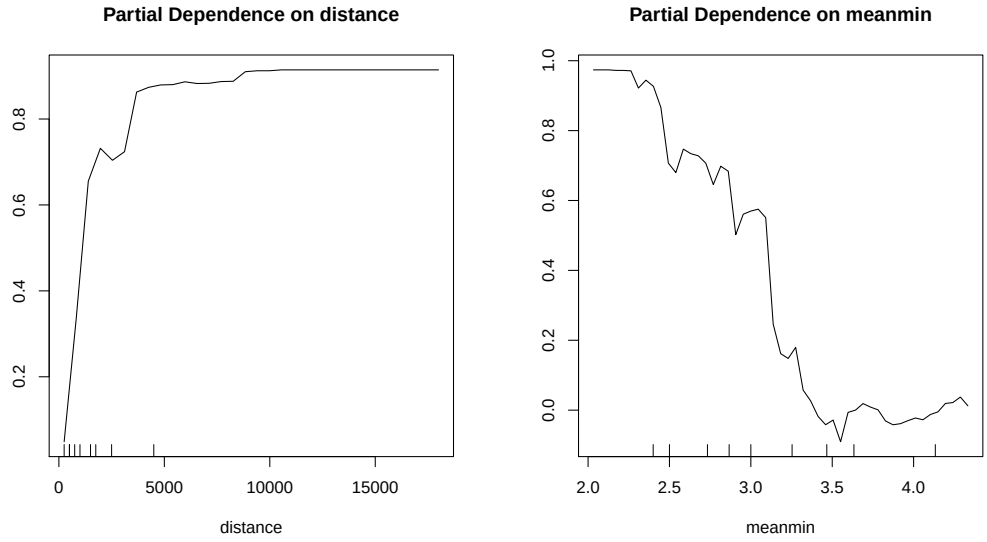

Figure 3.2: Variable importance plot of a random forest model for `pres.abs`.

3.2 Boosting

We now turn to another tree-based procedure called boosting. The idea behind boosting in general is the construction of additive regression functions by taking a simple base learning algorithm or *weak learner* which performs mediocrely satisfying and *boosting* it into an arbitrarily *strong* learning algorithm ([25] Schapire, 1999: p. 1). The procedure aims to "combine the output of many *weak* classifiers to produce a powerful *committee*." ([13] Hastie et al. 2001: p. 299).

In case of regression problems, it does so by computing fitted values and their errors using the base learner and then fits the errors in a subsequent model. The achieved error estimations are then added to the estimated response values from the previous iteration. For classification, the key lies in giving observations different weights based on their respective model errors from the previous iteration. This results in focusing more on observations with greater errors.

Any learning functions can be used as base learner but we focus on tree models. As Random Forests, boosting builds on the flexibility of CART models but unlike Random Forests it does not require them to be the underlying base procedure. The fundamental difference to Random Forests now is that Boosting does not average results from several *parallel* CART models but uses the tree results *in series*.

Boosting was originally designed for classification problems but can as well be extended to regression. In the following we will discuss two different implementations of the boosting framework and their `R` implementations.

3.2.1 AdaBoost

There are several different approaches to the boosting idea, therefore there is not one distinct procedure as in Random Forests. In its earliest stage, boosting has its implementation in AdaBoost which dates back to the work of Schapire (1990) (see [24]) and [9] Freund and Schapire (1995). In its original form, it works for classification problems.

A "weak" classifier is one that is only slightly better than random guessing. AdaBoost now sequentially applies the weak classifier to repeatedly reweighted data points and hence produces a series of weak classifiers $G_m(x)$ for every iteration m with $m = 1, ..., M$. The final prediction is then achieved by combining all M.

We consider a binary response variable Y coded 1 or -1 given a vector of predictor variables X with N observations. As stated in [13] Hastie et al. (2001) the algorithm has the following general structure.

1. For each of the N observations initialize observation weights $w_i = 1/N, i = 1, ..., N$.

2. For $m = 1$ to M do

   (a) Fit a classifier $G_m(x)$ to the training data using the weights w_i.

   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i 1_{\{y_i \neq G_m(x_i)\}}}{\sum_{i=1}^{N} w_i}$$

   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.

   (d) Set $w_i \leftarrow w_i \cdot \exp\{\alpha_m G_m(x)\}, \ \ i = 1, ..., N$.

3. Output $G(x) = \text{sign}[\sum_{m=1}^{M} \alpha_m G_m(x)]$.

In theory, G could be any classification procedure but classification trees are common and give favourable results. In 1996, Leo Breiman even called AdaBoost with trees the "best off-the-shelf classifier in the world" (see [3] Breiman, 1996).

The classification error from pass m, err_m, is logarithmically transformed into α_m which is then used to calculate the new case weights w_i. The algorithm "up-weights" or "boosts" only those observations that were misclassified in the previous pass. Hence, in every iteration AdaBoost focuses more on the cases that were misclassified in the previous iteration. The final prediction then results from a weighted vote of the M classifiers with weights α_m.

3.2.2 Stochastic Gradient Boosting

The second implementation of boosting that is presented is more complex. Before describing the algorithm we have a look at its structure in general. Following [15] Kriegler (2007, page 21) the typical structure of a boosting algorithm for regression problems is generally the same and is given below. The framework is presented for regression problems but the type of boosting that will be discussed applies this general structure for both regression and classification problems.

1. For all observations initialize the fitted values to a scalar.

2. For each observation and a given loss function compute the error between fitted and actual values.

3. Fit a model of these errors against the predictors and get fitted values of the errors.

4. Compute the boosted fitted values by adding the predicted values from the previous iteration to the fitted errors from step 3 generating a new vector of predictions.

5. Iterate a large number of times.

The difference to AdaBoost is that there is no actual "reweighting" involved as in iteration m the weak learner is fitted to the current residuals. This principle also works for classification where a binary response is treated as a continuous one.

With this in mind we are now able to approach a boosting method called *stochastic gradient boosting*. It was introduced in 1999 by Jerome Friedman and published in the two papers [10] Friedman (2001) and [11] Friedman (2002).

The following discussion focuses on the classification case with binary response values y_i coded as 0 or 1. However, the regression case essentially works the same way and a description for continuous problems can be found in [15] Kriegler (2007). Here, we follow [2] Berk (2008, section 6.4) and [11] Friedman (2002).

Let $x = \{x_1, ..., x_p\}$ be a set of explanatory variables and let n be the number of observations in the training data sample. A given regression tree can be written as

$$T(x, \Theta) = \sum_{j=1}^{J} \gamma_j 1_{\{x \in R_j\}}$$

with tree parameters $\Theta = \{R_j, \gamma_j\}$ and j being an index of the terminal node of the tree, R_j representing the parameter space region defined by the jth terminal node and γ_j being the value assigned to each observation of the response falling into the jth terminal node, i.e. the class label achieved by majority vote. As already stated before, it follows

$$x \in R_j \Rightarrow f(x) = \gamma_j$$

with $f(x)$ being the estimated relationship between x and y. Regression trees now look for the choice of parameters that minimizes the loss:

$$\Theta^* = arg \min_{\Theta} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j)$$

for a given loss function L which describes the error between the observed response values y_i and their corresponding estimates γ_j. Now, the key is to minimize this loss function over several trees generated in a stagewise manner using information from the previous iteration. So in every iteration m we seek

$$\Theta_m^* = arg \min_{\Theta_m} \sum_{i=1}^{n} L(y_i, f_{m-1}(x_i) + T(x_i, \Theta_m))$$

with $f_{m-1}(x_i)$ being the results from the previous iteration. Hence, in the current iteration the new tree parameter set Θ_m^* is calculated by minimizing the loss between the response values and the "boosted" fitted values. These

are the fitted values from the previous iteration plus the current terminal node predictions. Thus in every iteration a new regression tree is built using this minimum loss aspect.

Friedman's idea that leads to *gradient boosting* is to formulate the change in the loss function as the fitting function f gets changed from one iteration to the next as the negative gradient of the loss function:

$$g_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]$$

The g_{im} are called *pseudo-residuals*. During the procedure they are used as response and a regression tree is fit to them in every iteration. The terminal node predictions of these are again calculated by minimizing the loss and are then used to update the fitted values.
After initializing $f_0(x) = arg\min_\kappa \sum_{i=1}^{n} L(y_i, \kappa)$, where κ is a constant, the procedure works as follows:

For $m = 1, ..., M$ iterate the following steps:

1. Out of the full training data sample $\{y_i, x_i\}_{i=1}^{n}$ select a random subsample $\{y_{\pi(i)}, x_{\pi(i)}\}_{i=1}^{\tilde{n}}$ of size $\tilde{n} < n$ with $\{\pi(i)\}_{i=1}^{n}$ a random permutation of $1, ..., n$. This is the *stochastic* part of the method.

2. For observations $i = 1, ..., \tilde{n}$ compute the above gradient g_{im} as the working response.

3. Fit a regression tree $\{R_{km}\}_{k=1}^{K}$ with K terminal nodes to the gradients using the selected observations.

4. For every terminal node $k = 1, ..., K$ compute γ_{km}, the optimal node prediction of terminal node k from iteration m, such that

$$\gamma_{km} = arg\min_{\gamma} \sum_{x_{\pi(i)} \in R} L(y_{\pi(i)}, f_{m-1}(x_{\pi(i)}) + \gamma)$$

   where the $f_{m-1}(x_{\pi(i)})$ are the response estimates from iteration $m - 1$. Thus, the errors that minimize loss between the observed response and the previous fitted values are sought.

5. Finally calculate the new fitted values as

$$f_m(x) = f_{m-1}(x) + \nu \cdot \gamma_{km} 1_{\{x \in R_{km}\}}$$

   with $0 < \nu < 1$.

The final fitted values will be achieved from the final vector of predictions. In the binary response case, a 0 is assigned if the final predicted value $\hat{y}_i = f_M(x) < 0.5$, otherwise a 1 is assigned.

The step size for updating the fitted values from one iteration to the next, $0 < \nu < 1$, is called *shrinkage* and describes the learning rate of the algorithm. In [10] Friedman found out that a smaller value ($\nu < 0.1$) leads to better results in terms of the generalization error. The purpose of including ν is to prevent the method from over-shooting a more optimal solution.

The stochastic component from step 1 of the algorithm was introduced in [11] Friedman (2002) and resulted in a marked improvement in predictions. Without the stochastic part (and with $\nu = 1$), the algorithm is simply called *gradient boosting* (see [10] Friedman, 2001). In his work, Friedman states that the introduced randomization substantially improves the non-randomized version "through the simple expedient of training the base learner on different randomly selected data subsets at each iteration" ([11], p. 9). It also helps to control overfitting. But he admits that the reason for the achieved improvement is not clear. The improvements above are substantial for small samples and with high variance base learners such as CART models. Small subsamples generated through the randomization cause the variance of the base learner to increase further and the correlation between the base learners at different iterations to decrease. With this the variance of a combined model tends to be reduced as the combined model "in effect averages the base learner estimates".

3.2.3 The Loss Function

In case of a continuous regression problem the usual loss function is the squared error or *Gaussian* loss function:

$$L(y_i, f(x_i)) = \sum_{i=1}^{n} (y_i - f(x_i))^2.$$

Another possible loss function for continuous response variables is the *Laplace* loss function:

$$L(y_i, f(x_i)) = \sum_{i=1}^{n} |y_i - f(x_i)|.$$

In case of a categorical response variable the loss function used is the so called *Bernoulli* loss function. For such a response variable it is common to model the class proportions. Thus, a logistic regression problem with categorical response is written as

$$\log\Big(\frac{p_i}{1 - p_i}\Big) = f(x_i)$$

where f is the estimated relationship between predictors and response, x_i is the ith observation and p_i is the class proportion for every observation

$(i = 1, ..., n)$. The left hand side in the above equation is called *logit*. This results in the transformation

$$p_i = \frac{1}{1 + \exp(-f(x_i))}.$$

With this, the values on the right hand side are constrained to $[0, 1]$ and one is able to model the class proportions.

The above relationship can also be written as

$$\frac{1}{1 - p_i} = 1 + \exp(f(x_i)).$$

Using this and the calculation rules for logarithms one can rewrite the log likelihood function of the Bernoulli distribution:

$$\begin{aligned}
\sum_{i=1}^{n} \Big(y_i \log(p_i) + (1 - y_i) \log(1 - p_i) \Big) &= \sum_{i=1}^{n} \Big(y_i \log\Big(\frac{p_i}{1 - p_i}\Big) + \log(1 - p_i) \Big) \\
&= \sum_{i=1}^{n} \Big(y_i f(x_i) - \log\Big(\frac{1}{1 - p_i}\Big) \Big) \\
&= \sum_{i=1}^{n} \Big(y_i f(x_i) - \log(1 + \exp(f(x_i))) \Big)
\end{aligned}$$

Thus, the Bernoulli loss function or *Bernoulli deviance* is given by

$$L(y_i, f(x_i)) = \sum_{i=1}^{n} \Big(y_i f(x_i) - \log(1 + \exp(f(x_i))) \Big).$$

3.2.4 Variable Importance

The variable importance measure for boosted tree models is essentially based on the same idea as the one for single trees. For boosting, the importance for every single tree is just summed up over each iteration. In the classification case importance of a variable is based on the amount of reduction of a node's impurity it yields when used for splitting that node. In the regression case the reduction in the deviance is the key factor. So after a boosting model is constructed the error reduction statistics are summed up over each tree model for each predictor variable. Then this procedure yields a variable importance ranking. In addition one can normalize the resulting values to place the measures of importance on a percentage scale.

3.2.5 Problems of Boosting

Boosting is a very powerful method but the method also has some serious drawbacks.

First, like random forests, a boosting model lacks a single interpretable output. Single decision trees are very intuitive and highly interpretable. The model can easily be visualized by its tree representation whereas a combination of several such tree models can not. The main output just consists of model quality values or cross-validation results. Graphical outputs of a boosting model are limited to error plots, variable importance ranking plots or partial dependence plots. For a description of such alternative approaches of interpretation, see [13] Hastie et al. (2001, section 10.13). Thus, boosting remains a black box method.

Second, overfitting remains a problem. As stated in [2] Berk (2008, section 6.5.1), overfitting is present in boosting as there is no mechanism for averaging out this issue as in a random forest. Cross-validaton results or the use of a separate test data set can make the user aware of this problem and give guidance on when to stop the boosting process.

A problem for classification with stochastic gradient boosting is that there is no distinction between false positives and false negatives because classification problems are transformed into regression problems when residuals are defined. Positive and negative residuals are treated equally. For regression problems, weights can be applied to capture asymmetric costs as suggested in [15] Kriegler (2007).

Finally, as with random forests, there are no satisfying proofs of consistency. It is not clear if the functional relationship estimated by boosting also estimates the true mechanism that links inputs and outputs.

3.2.6 Example of Boosting in R

There are several implementations of boosting in `R`. Out of the numerous packages we chose `gbm` for our analysis as it implements Friedman's stochastic gradient boosting machine. A detailed overview is given in [22] Ridgeway (2007). In the following we give a brief introduction to the package.

The main command for fitting a stochastic gradient boosting model for both classification and regression is `gbm`. It has the same syntax as `lm` and the other modelling functions discussed above but also needs a loss function to be specified. Some possible options for the loss function are `gaussian` for the square error, `laplace` for the absolute error and `bernoulli` for classification problems.

In case of the `frogs` data classification example, a stochastic gradient boosting model is fit with the following commands.

```
> library(gbm)
> mod<-gbm(pres.abs~., data=xtrain, distribution="bernoulli")
```

Here, the model is fitted with Bernoulli loss as loss function L. In this case one has to keep in mind that `pres.abs` must not be declared as `factor` but has to be numeric. The problem type is specified solely via the `distribution` argument.

The most important additional optional parameters are

- `n.trees`: The total number of constructed trees in the model. This is equivalent to the number M of iterations. The default is 100.
- `interaction.depth`: The maximum depth of variable interactions in each tree. For example, a value of 2 yields a model with 2-way interactions. The default is 1.
- `n.minobsinnode`: Specifies the minimum number of observations in each tree's terminal nodes. The default is 10.
- `shrinkage`: The shrinkage parameter ν which describes the stepsize for updating the fitted values between iterations. The default value is 0.001.
- `bag.fraction`: Specifies the fraction of the training set observations randomly selected for each iteration. This controls the stochastic part of the algorithm. A value < 1 yields slightly different results with each function call. It uses the `R` random number generator, thus `set.seed` ensures that one run can be reconstructed. The default value is 0.5.

As with random forests there is not one interpretable graphical output as in CART models. Therefore the `plot` method for a `gbm` object yields a partial dependence plot as was described earlier. With

```
> plot(mod, i.var=1)
```

one produces a partial dependence plot of the first variable in the training set against the response. As with a `randomForest` object, one can also produce a performance plot. The command

```
> gbm.perf(mod, method="OOB")
```

produces an error rate plot for determining the optimal number of iterations using generated OOB observations. Depending on the distribution, the resulting plot shows the number of boosting iterations against the corresponding value of the respective loss function.

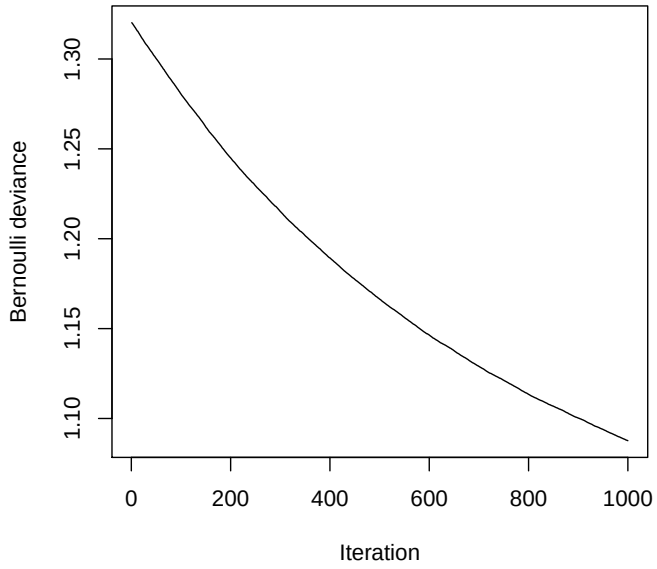

Figure 3.3: Performance plot of a stochastic gradient boosting classification model for `pres.abs`.

In our `frogs` example, it shows the number of iterations against the corresponding Bernoulli deviance. In figure 3.3 a performance plot is shown with a boosting model for `pres.abs` and $M = 1000$ iterations.

For our later analysis we use the `caret` package as a wrapper for `gbm`'s functionality as it is more comfortable to use. See the appendix for a desription on how to use boosting in the `caret` package.

Chapter 4

Process Characterization

4.1 The CVD and PVD Process

Chemical vapor deposition (CVD) is a single process that can be found multiple times during wafer fabrication. Typically, a CVD process is used to deposit a thin film of oxid or other materials on the surface of a wafer. To achieve such a film deposition reactive carrier gases are used to transport precursors of a desired material to the substrate. Chemical reactions with other gases or decomposition lead to reaction products which are then deposited on the surface. It is a highly versatile method because it can utilize a wide range of different chemical reactants and reactions to achieve deposition of different types of films.

A usual CVD system consists of a reaction chamber, thermal heating and plasma energy sources to provide the energy needed for the desired chemical reactions to occur, gas carrier and source materials. Normally the process is performed at vacuum level.

At the beginning the used carrier gas picks up a quantity of the source materials while flowing through them. Thereby materials are mixed and then injected into the reaction chamber via a manifold. In the chamber the energy provided by the external energy sources excites a chemical reaction. Thereby a uniform flow over the substrate is mandatory to ensure a uniform film deposition.

In the case of austriamicrosystem's HDP CVD the gas *silan* reacts with plasma to achieve deposition. This type of CVD process is called *plasma enhanced CVD* (PECVD) as it uses a plasma discharge to provide the excitation needed for the chemical reactions. The plasma is produced in the chamber and is brought to a certain level of density by using electromagnetic waves. These waves are generated in a radio frequency generator (RF generator) and directed into the chamber via multiple ways (top or side). Thereby the power from the generator (RF power, in W) is crucial. Part of the power is reflected and can not be used to enhance the plasma. This so called reflected power should be kept as low as possible.

In turn with the repeated deposition of oxid films, the edges of every film are sputtered using *argon ions* such that a desired shape of every deposited film is reached. RF bias power is used to excite and accelerate these argon ions. The higher the RF bias power the more intense the sputtering is.

Figure 4.1 shows a cross-section of a CVD chamber as it is in use at austriamicrosystems.

The physical vapor deposition (PVD) process is another method to deposit a thin film on a wafer surface. In contrast to the CVD process, it involves purely physical processes rather than chemial reactions. Typically, a solid cathode of aluminium is used as carrier for the material needed. The deposition is achieved by bombarding the cathode with charged argon ions emitted from a gas discharge. When these ions impinge on the cathode the collision sputters substrate atoms. Subsequently, these atoms are deposited on the wafer surface and create a metal film.

For more detailed information about the CVD and PVD sputtering processes in general see [8] Elshabini-Riad & Barlow (1998).

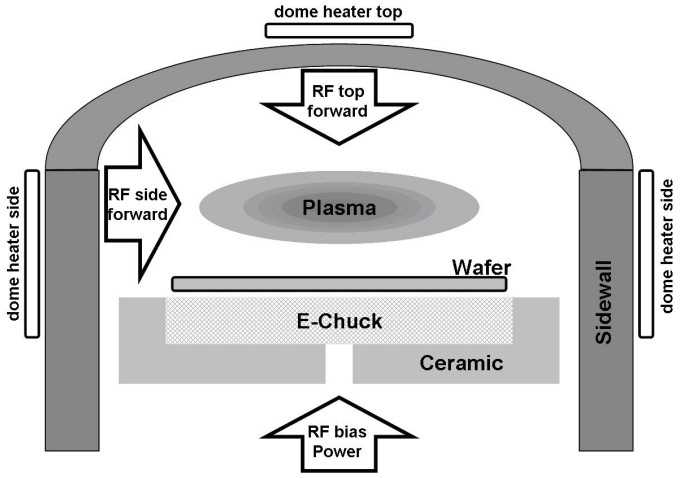

Figure 4.1: Cross-section of the chamber of a CVD tool as it is in use at austriamicrosystems.

4.2 The Implanter Process

Ion implantation is one of the most sophisticated processes during wafer fabrication and the required Ion-Implanter is a highly complex tool. Every transistor in an integrated circuit is formed by the results of ion implantation.

During the process, charged atoms or molecules of a certain species are created in the ion source of an implanter. These ions are accelerated up to a few hundred keV and impinged upon the wafer surface. The reason for this is that the implantation of certain ions changes the physical and electrical qualities

of crystals. The ions can be implanted with high accuracy of dose and energy over many orders of magnitude. This can be controlled by the acceleration voltage and a dose controller.

The tool consists of an ion source where the ionization takes place, a beam line through which the ions are accelerated and directed, and an end station chamber where the wafer is positioned. Part of the ion source is the *filament*. This mechanical part is stressed during operation and has to be changed after several days of usage. Figure 4.2 shows a diagram of an ion implanter as used by austriamicrosystems.

A detailed description of the process of ion implantation can be found in [32] Wolf (2003, chapter 12).

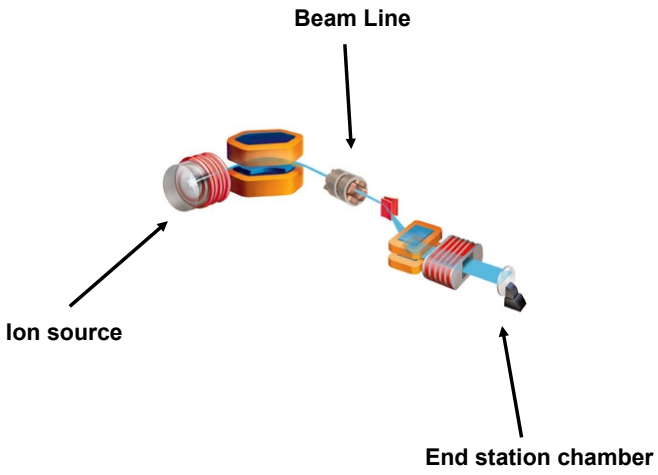

Figure 4.2: Diagram of an ion implanter tool.

Chapter 5

Prediction of Implanter Maintenance

5.1 Introduction

Part of the ion source of an ion implantation tool at austriamicrosystems is the filament. This mechanical part is stressed during the implanter operation and breaks approximately every 5 to 14 days. When a filament break occurs, a tool engineer has to change the broken part. This is a problem if the breaking happens during a weekend. Knowledge of the exact date and time of this moment in advance, i.e. before a weekend, would solve the problem. The part would then simply be changed in advance. This would save costs and increase the throughput of the ion implanter as idle times would be reduced. Therefore we try to find a statistical model for predicting the occurrence of the filament break.

The response variable is created using the exact dates of historical filament maintenance occurrences. Thereby, in the given set of machine data every row of the data set is assigned the exact number of hours until the next filament break, thus our response variable `NextPM` (*next preventive maintenance*) is "hours until the next maintenace occurs". This is highly intuitive and predicted values are easy to interpret for engineers. As these are continuous values, this is a regression problem.

5.2 Data analysis

The initial data set of given implanter machine data consists of $n_{initial} = 6781$ rows and $p_{initial} = 18$ machine parameters that were measured during the actual ion implanting. The values are aggregated automatically by the machine over the current beam set up and the beam set up changes every time a new recipe is started on the machine. Thus, one entry in the data represents an

automatically aggregated value of one of the 19 machine parameters measured during one recipe run.

Figure 5.1 shows the response created from the initial data set plotted over approximately 5 months or 7352 data points.

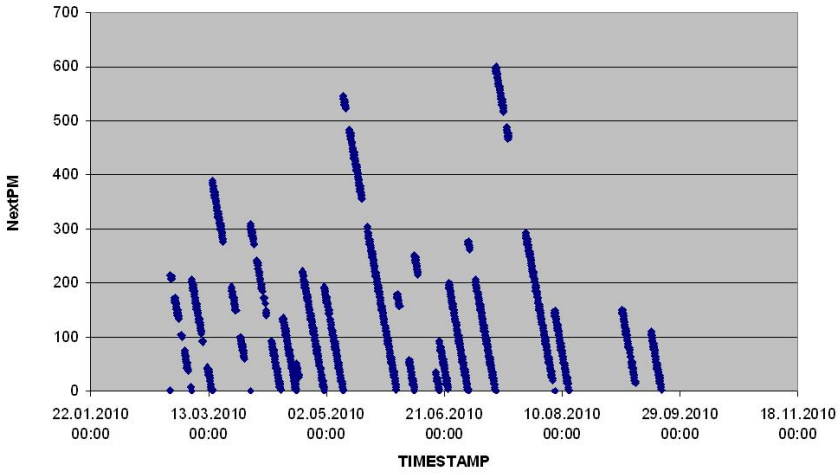

Figure 5.1: Hours until filament break over time.

The hours until the next maintenance decrease strictly monotonic until a filament break occurs and then jump up for the next filament lifetime cycle. Obviously, there are gaps in some of the cycles. These gaps can either come from the machine being shut down completely, i.e. no beam at all, or from the machine being idle, i.e. with a beam running but without actual production. The gaps that result from the machine being shut down affect the response because the response values should mirror the hours of the filament being healthy with the beam in use up to the point where it breaks. Therefore the response values have to be adjusted such that these shut down times are factored out.

Furthermore, we are only interested in a timeframe of about 150 hours before a filament break for fitting a prediction model as there is no need for response prediction further than that. The actual critical phase in which the model should perform accurately then is 72 hours before the break as a prediction is needed on fridays for the coming weekend. Therefore this is also considered when cleaning the values of `NextPM`.

Figure 5.2 shows the response values with these adjustments factored in. Then, $n = 1812$ data points remain. Figure 5.3 shows a histogram of the response.

The predictor matrix contains 18 non-constant predictors. In table 5.1 they are listed along with a short description.

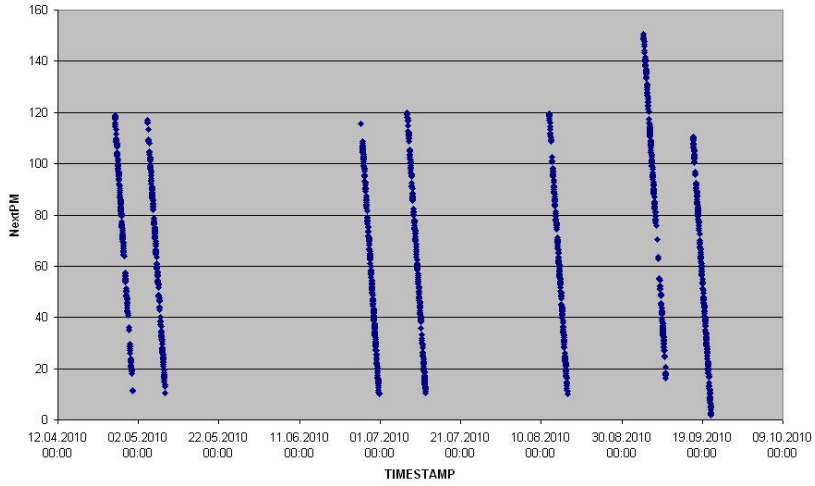

Figure 5.2: Hours until filament break over time with adjustments factored in.

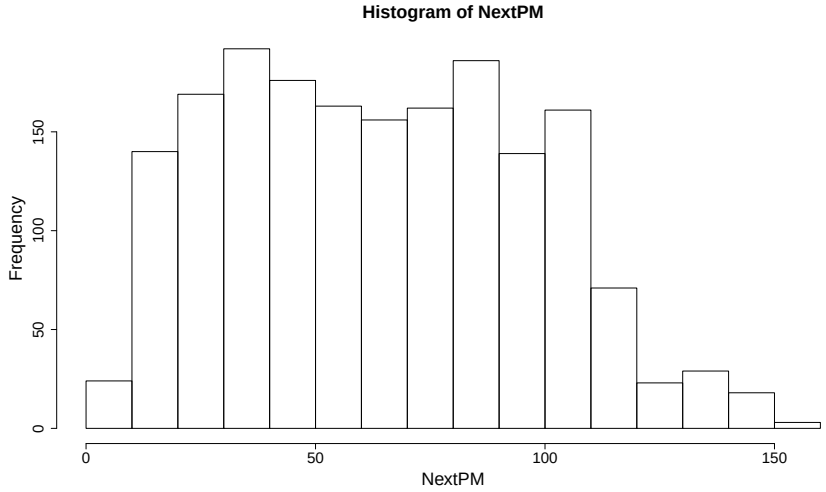

Figure 5.3: Histogram of 1812 values of the response variable `NextPM`.

Out of discussions with the engineers, `FIL_I` should be the most important predictor as it provides direct information about the filament. All power parameters (`EXT_I`, `SUP_I`, `ARC_I`, `BEAM`) are also considered to be influential. On the other hand, `BEAM_ENERGY`, which is measured on the front end of the implanter tool, should not have as much explanatory power. `BEAM_I_RANGE` only gives information on how `BEAM` was measured (milli-, micro- or nano-ampere) and is not relevant after normalizing the values of `BEAM` accordingly. `SUP_VOLTS` is a constant with value 1.2 and `SCANNER_PRESSURE` is nearly constant with only 20 different values in the data set, thus we can omit them in our set of predictors. The time elapsed between two different data points (`Delta`) was suggested by engineers to also consider for modelling. A summary of the relevant predictors and the response `NextPM` follows.

Parameter	Description
ION_NAME	Specifies the used element to implant
GAS	Pressure of gas bottle for storing the elements
FIL_I	Power streaming over the filament
EXT_I	Power coming out of the source
EXT_VOLTS	Voltage of the extraction power
EXTRACTION	Target value of EXT_VOLTS
ARC_I	Power between filament and source chamber
ARC_VOLTS	Voltage of the arc power
SUP_I	Power filtered out of extraction power
SUP_VOLTS	Voltage of suppression power
X_AXIS	Geometry value of source and extraction blind
Y_AXIS	Geometry value of source and extraction blind
Z_AXIS	Geometry value of source and extraction blind
SOURCE_PRESSURE	Vacuum value, must correspond with GAS
SCANNER_PRESSURE	Pressure of the scanner
CHAMBER_PRESSURE	Pressure in the source chamber
BEAM	Power of the ion beam
BEAM_I_RANGE	Specifies unit in which BEAM is measured
BEAM_ENERGY	Total energy that reaches the wafer
ION_NAME	Specifies the target atomic mass unit
DELTA	Time elapsed since last data point

Table 5.1: Tabular overview and descriptions of the variables measured on an implanter tool.

```
      GAS             FIL_I            EXT_I           EXT_VOLTS
 Min.   :1.539   Min.   : 98.9   Min.   : 0.000   Min.   :20.07
 1st Qu.:2.378   1st Qu.:122.9   1st Qu.: 0.310   1st Qu.:49.98
 Median :3.049   Median :133.0   Median : 0.690   Median :49.98
 Mean   :3.252   Mean   :133.3   Mean   : 1.785   Mean   :45.71
 3rd Qu.:3.925   3rd Qu.:143.7   3rd Qu.: 2.670   3rd Qu.:49.98
 Max.   :6.398   Max.   :173.0   Max.   :17.720   Max.   :65.03

   EXTRACTION         ARC_I            ARC_VOLTS       SUP_I
 Min.   :200.0   Min.   :0.00271   Min.   : 44.90   Min.   :0.1200
 1st Qu.:500.0   1st Qu.:0.07500   1st Qu.: 48.90   1st Qu.:0.1400
 Median :500.0   Median :0.24300   Median : 64.90   Median :0.1900
 Mean   :457.3   Mean   :0.64568   Mean   : 62.47   Mean   :0.3602
 3rd Qu.:500.0   3rd Qu.:0.87900   3rd Qu.: 70.00   3rd Qu.:0.3600
 Max.   :650.0   Max.   :8.51000   Max.   :100.00   Max.   :5.3100

    X_AXIS          Y_AXIS          Z_AXIS        SOURCE_PRESSURE
 Min.   :428.8   Min.   :428.8   Min.   : 15.3   Min.   :2.000e-06
 1st Qu.:467.1   1st Qu.:464.9   1st Qu.:382.2   1st Qu.:3.300e-06
 Median :482.7   Median :478.6   Median :919.4   Median :5.000e-06
 Mean   :493.0   Mean   :486.4   Mean   :735.1   Mean   :5.149e-06
 3rd Qu.:518.8   3rd Qu.:503.7   3rd Qu.:989.4   3rd Qu.:6.400e-06
 Max.   :598.1   Max.   :602.3   Max.   :999.0   Max.   :1.600e-05

  CHAMBER_PRESSURE         BEAM           BEAM_ENERGY
 Min.   :1.700e-07   Min.   :7.069e-06   Min.   : 24.9
 1st Qu.:1.700e-07   1st Qu.:7.339e-05   1st Qu.: 49.8
 Median :9.700e-07   Median :1.743e-04   Median : 79.8
 Mean   :1.511e-06   Mean   :2.783e-04   Mean   :102.3
 3rd Qu.:2.700e-06   3rd Qu.:3.008e-04   3rd Qu.:122.5
 Max.   :5.700e-06   Max.   :2.500e-03   Max.   :500.5

 Delta                 ION_NAME          NextPM
 Min.   :0.00000       11: 598       Min.   :  1.95
 1st Qu.:0.06667       31: 678       1st Qu.: 36.42
 Median :0.27000       40: 27        Median : 62.52
 Mean   :0.43511       49: 479       Mean   : 64.22
 3rd Qu.:0.55000       75: 30        3rd Qu.: 89.74
 Max.   :7.51667                     Max.   :150.70
```

`ION_NAME` is a categorical variable indicating which atom species is used for implantation, thus it is included as a `factor` variable. It has classes 11 (for Bohr), 31 (Phosphor), 40 (Argon), 49 (Bohr-Flurid) and 75 (Arsen). The summary statistic shows the number of data points for each class.

The next step is to further reduce the set of predictors and filter out the subset that is most important for modelling the response. At first we look at the correlation matrix of predictors and the response. Figure 5.4 shows a `lattice` plot to visually identify high correlated predictors.

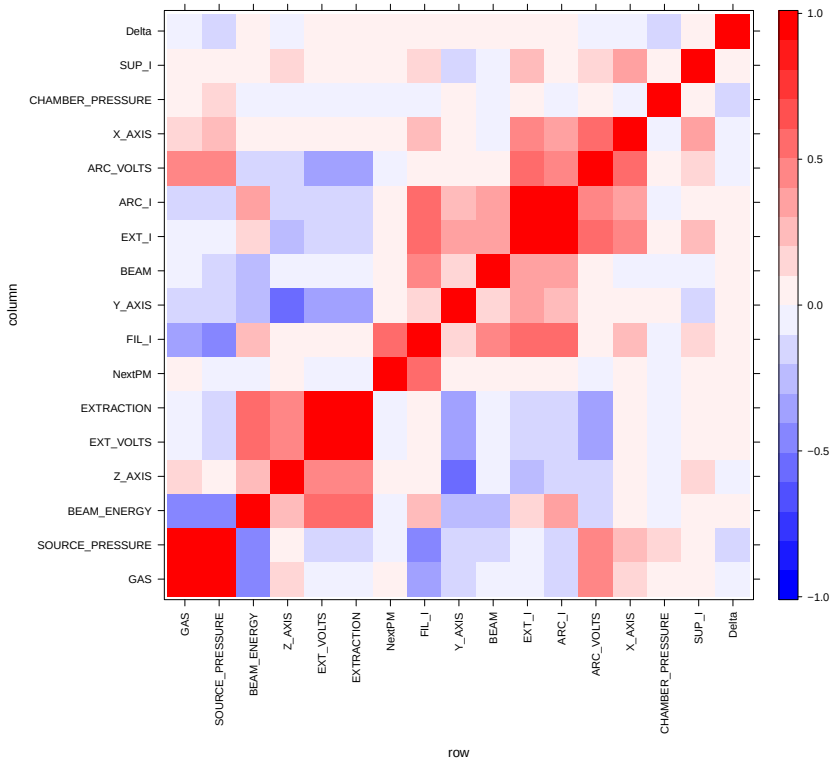

Figure 5.4: Correlation matrix of the 18 non-constant parameters of ion implanter machine data. Red means high positiv linear correlation, white means no linear correlation and blue means high negative linear correlation.

`FIL_I` is positively related to the response with a correlation of 0.54 and therefore seems important as was assumed by engineers. Three parameters are redundant as they are highly correlated with others. The relationship of `SOURCE_PRESSURE` with `GAS` is known. `EXT_VOLTS` and `EXTRACTION` obviously have to show high positive correlation as `EXTRACTION` is just the target value of `EXT_VOLTS`. The high positive correlation of `ARC_I` and `EXT_I` is also clear as they measure similar power streams. Therefore we can omit `SOURCE_PRESSURE`, `EXTRACTION` and `ARC_I` from further modelling.

Thus, our remaining set of predictors contains 13 continuous and 1 categorical variables. A look at the scatterplots in figure 5.5 of the remaining predictors

versus the response shows no other dependencies besides the positive relationship with FIL_I.

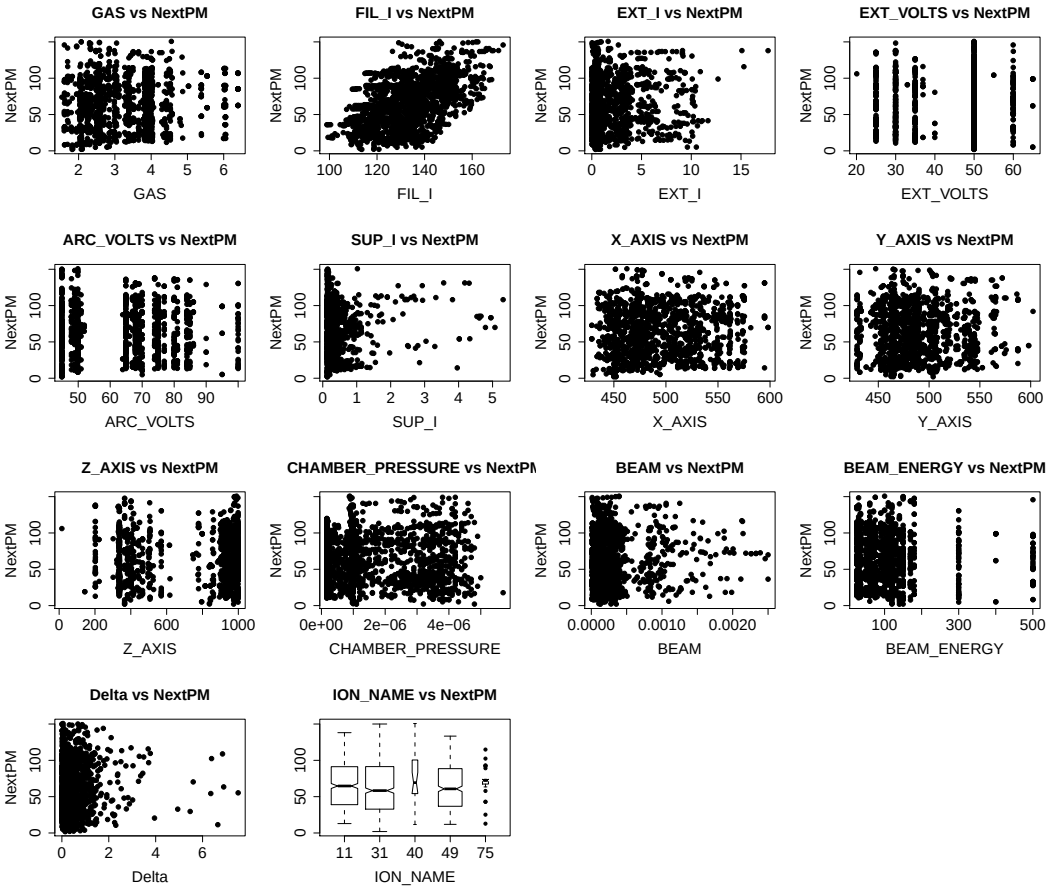

Figure 5.5: Scatterplots of the remaining parameters against the response NextPM.

The boxplots of the different species of atoms against the response (last plot in figure 5.5) does not show significant difference of the first 4 species. The class Arsenic (ION_NAME class 75) shows significant differences, but as there are only 30 data points among the 1812 observations and a few outliers, we do not consider the difference as significant. Argon (ION_NAME class 40) also has very few observations, the most frequently used elements are Boron, Phosphorus and Boron-Fluoride.

5.3 Regression Tree models

As next step we fit regression tree models to have a first look at variable importance and the data structure itself. Figure 5.6 shows a rpart regression tree model generated with our 14 uncorrelated predictors. A maximum tree depth of 4 is used.

Clearly, FIL_I is the most important parameter. Even both of the second level partitions are constructed using FIL_I as split variable. The first split at FIL_I splits the data in two thirds where FIL_I is smaller than 139.1 and one third where it is larger. For the filament power larger than 139.1, the power parameters EXT_I and SUP_I are eventually important. Based on the positive relationship between NextPM and FIL_I as observed in figure 5.5, the average response values in this branch are larger. For FIL_I < 148.4, EXT_I splits 336 data points into partitions with an average response of 60.99 (156 cases) and 90.09 (180 cases). Thus, a smaller value of EXT_I in this particular partition relates to a longer filament lifetime.

Among values of FIL_I smaller than 139.1 we eventually find EXT_I and GAS to be important, with GAS being important for data points where FIL_I is smaller than 115.2. Furthermore, if GAS is smaller than 6.22, the partition consists of 183 values of the response with a mean of 31.4, so most of our observations measured directly before a filament break come to rest in this terminal node. On the other hand, if GAS is larger than 6.22, a terminal node results with a clearly larger average NextPM value of 80.21. However, with only 14 observations in the node the conclusions are not as meaningful.

In the partition where EXT_I is used for splitting 995 data points, the resulting partitions are relatively different. For EXT_I smaller than 0.3, a 270 data point partition results where the average response value is clearly higher than in the EXT_I ≤ 0.3 branch. Thus this parameter constellation is related to a longer filament lifetime.

In summary, judging from the rpart regression tree model, the most important variables seem to be FIL_I, EXT_I and GAS.

A regression tree generated using unbiased recursive partitioning ideas is shown in figure 5.7 with a maximum tree depth of 3.

In general, the results seem rather similar to figure 5.6. Again, FIL_I is the top predictor to perform a split on and EXT_I and GAS are important in relevant data regions. The plot again shows that a longer filament lifetime can be achieved for smaller EXT_I values in the respective data region (terminal nodes 11 and 12).

Furthermore, for values of FIL_I between 115.1 and 139, ION_NAME can be found as important. In this partition, classes 40 (Argon) and 49 (Boron-Fluoride) result in slightly larger values of NextPM than the other 3 classes (terminal nodes 7 and 8). For very large values of FIL_I, the voltage parameter EXT_VOLTS seems important and results in the largest response values (terminal node 14).

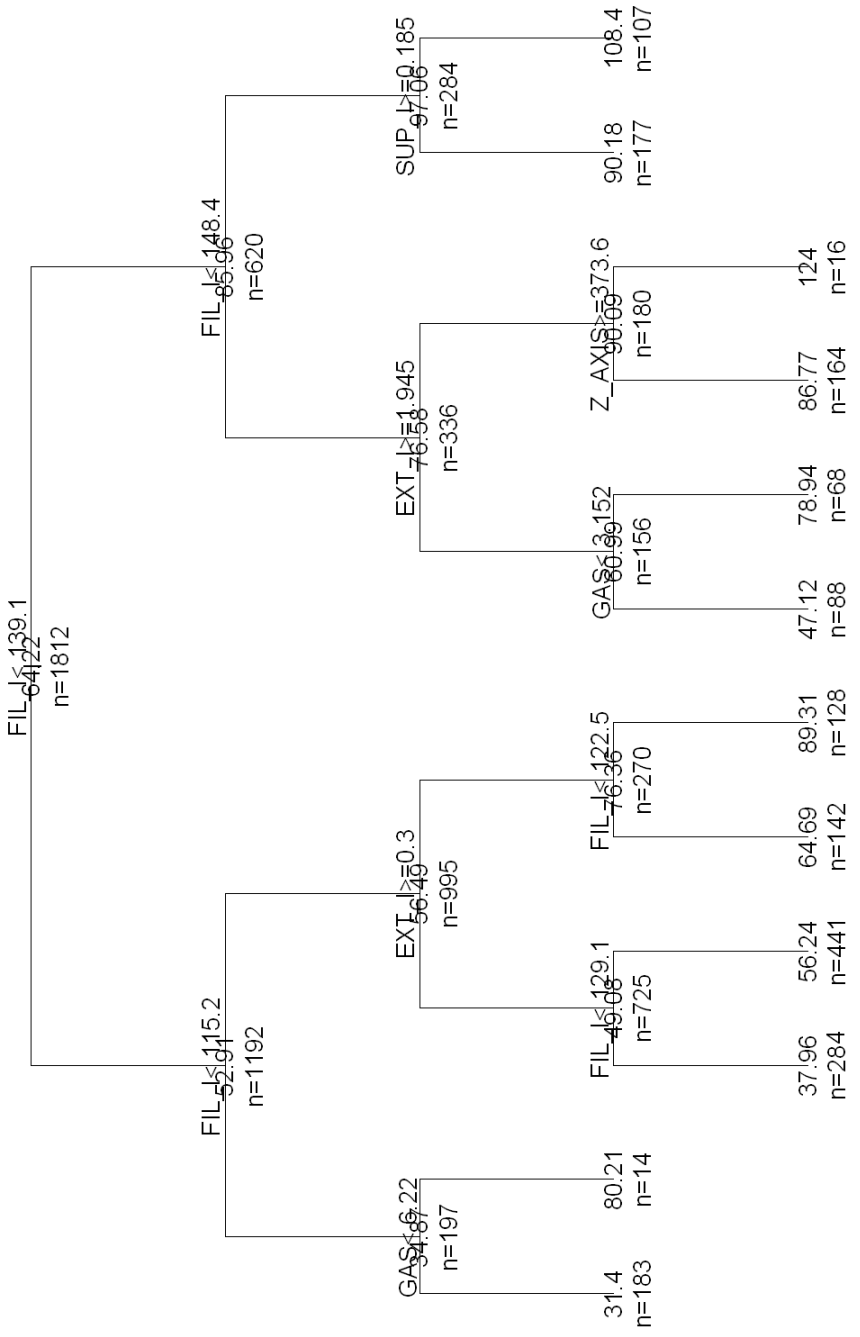

Figure 5.6: Regression tree model created using the `rpart` R routine with a maximum tree depth of 4.

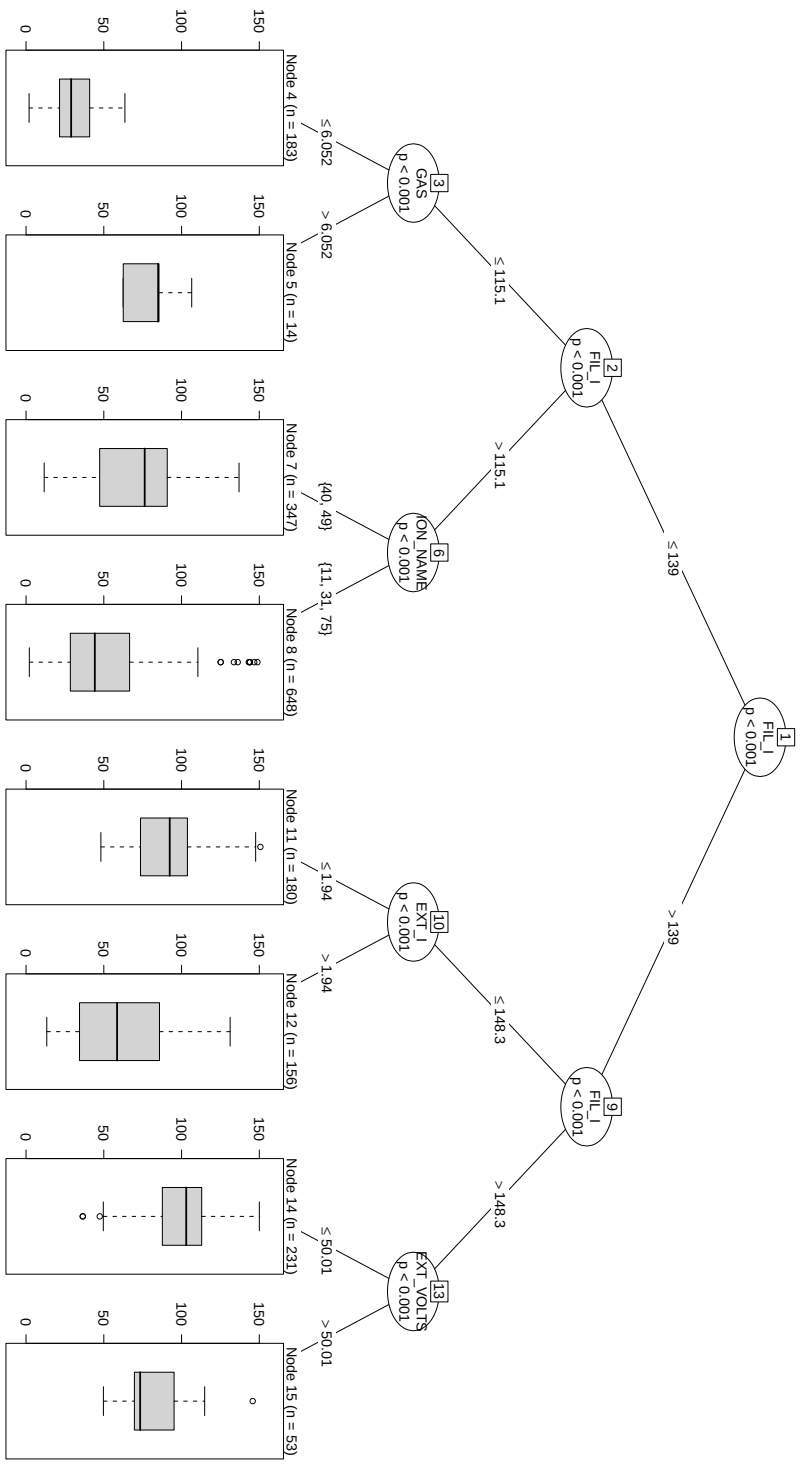

Figure 5.7: Regression tree model created using unbiased binary recursive partitioning ideas with a maximum tree depth of 3.

5.4 A Random Forest model

We are now ready to fit a model for `NextPM` with all 14 predictors and further reduce our set of predictors by using the built in variable importance measures. In the following we fit a random forest model. The predictor matrix is saved in the data frame `xtrain`.

```
> library(randomForest)
> set.seed(1)
> mtry<-tuneRF(xtrain, NextPM, mtryStart=1, stepFactor=2,
  ntreeTry=1000, improve=0.01)
> best.m<-mtry[mtry[,2]==min(mtry[,2]), 1]
> best.m
[1] 14
> mod.rf.1<-randomForest(NextPM~., ntree=1000, data=xtrain,
  importance=TRUE, mtry=best.m)
```

The command creates a `randomForest` object with all of the 15 predictors against `NextPM`. The fitted values are the result of averaging over 1000 regression tree models. The value of `mtry` is determined by the `tuneRF` command and chosen to be the one that yields the smallest oob error. In the case above it is 14 with the default value of `mtry` being $p/3 = 4$. The plot generated by `tuneRF` in figure 5.8 shows that in terms of the oob error we are better off choosing `mtry` $= 14$ or `mtry` $= 8$ as utilizing the default value.

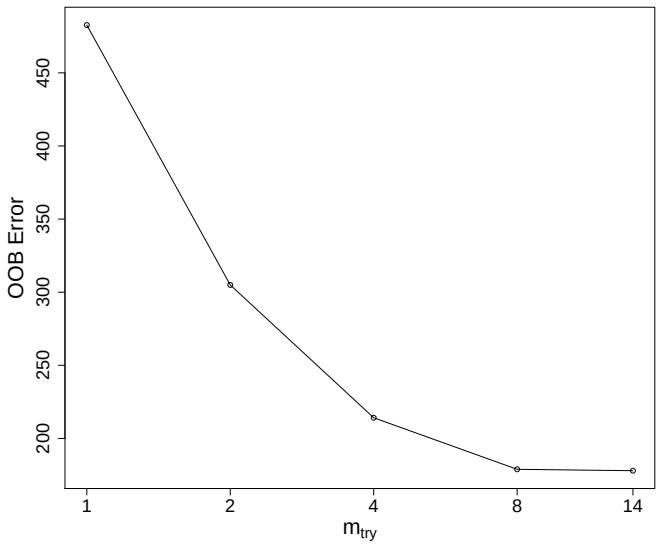

Figure 5.8: Numbers of variables to choose randomly for determining each split in each tree model of the random forest and their corresponding OOB error rate. A value of 14 gives the lowest error.

The resulting model object is saved in `mod.rf.1`. The output following the above commands is given below.

```
Call:

 randomForest(formula = NextPM ~ ., data = xtrain, ntree = 1000,
              importance = TRUE, mtry = best.m)

               Type of random forest: regression
                     Number of trees: 1000
No. of variables tried at each split: 14

          Mean of squared residuals: 178.4557
                    % Var explained: 83.58
```

The model explaines nearly 84% of the variance of `NextPM`. Furthermore the model yields a RMSE value of $\sqrt{177.35} = 13.3$ hours. An error of this magnitude is adequate for the purpose of predicting the filament lifetime for a weekend.

However, in order to reduce the overall complexity of the model but keeping the results similar to the full model we try some further analysis and parameter reduction. Following for example [27] Siroky (2009) we look at some diagnostic plots available for a `randomForest` object.

Figure 5.9 shows the plot of the `randomForest` object where the number of trees is plotted against the error rate. It is observable that a random forest containing 1000 regresson trees is not necessary to obtain a small oob error. In fact, even the default value of 500 is too much and about 300 trees seem reasonable.

The variable importance plot is crucial for further reducing model complexity. Figure 5.10 shows the variable importance plot with the two different variable importance measures $\text{IMP}^{(1)}$ and $\text{IMP}^{(2)*}$.

Out of figure 5.10 the top 5 variables are `FIL_I`, `EXT_I`, `SUP_I`, `GAS` and `BEAM`.

This basically coincides with the results from the regression tree models in figure 5.6 and figure 5.7 as well as with the engineers' assumptions as the power parameters were supposed to be important in the first place. The pressure of the gas valve as described by `GAS` is also among the top contributors. Based on our observations from the regression trees, `GAS` is important for small values of `NextPM`. On the other hand, the `Delta` as well as the voltage variables turn out to make a small contribution. As expected, the contribution of `BEAM_ENERGY` is negligible. The different chemical elements used for implantation which were factored as categories also do not play a major role as `ION_NAME` is found in the middle of the ranking.

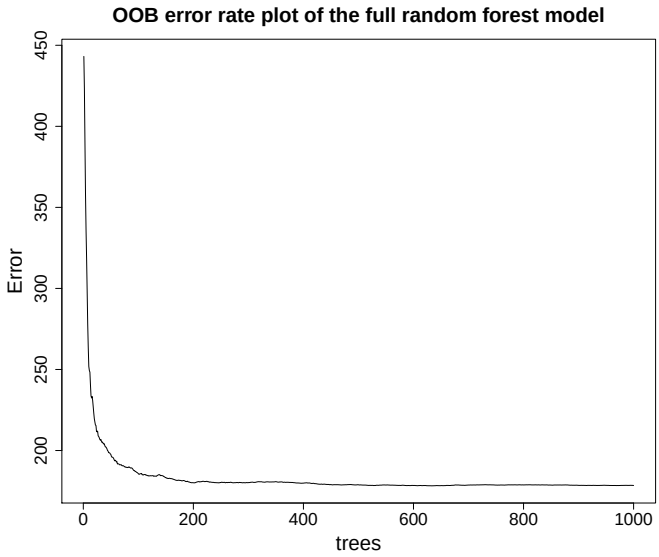

Figure 5.9: The number of trees in the random forest model and the corresponding out-of-bag error rate of the full model `mod.rf.1`.

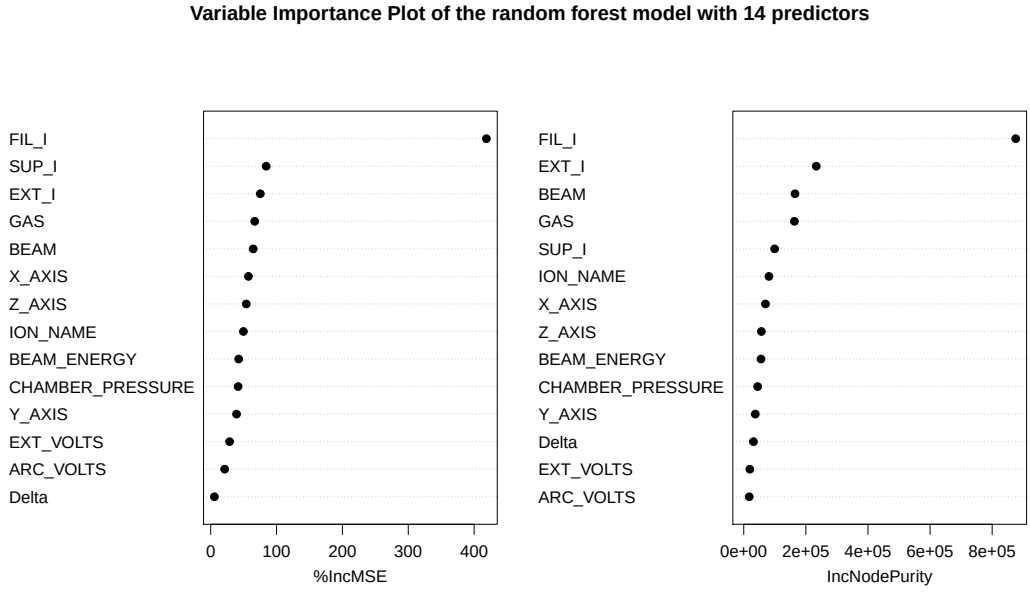

Figure 5.10: Variable Importance plot for the `mod.rf.1` random forest object with all 15 predictors. The variable importance measures $\text{IMP}^{(1)}$ and $\text{IMP}^{(2)*}$ are shown.

We are now interested in the relationship between the most important predictors and the response. Therefore we have a look at the partial dependence plots. Figure 5.11 shows the plots generated by the `partialPlot` command.

As already observed, the smaller `FIL_I` the shorter the lifetime of the filament. There seems to be a linear relationship between `FIL_I` and the response. Thus

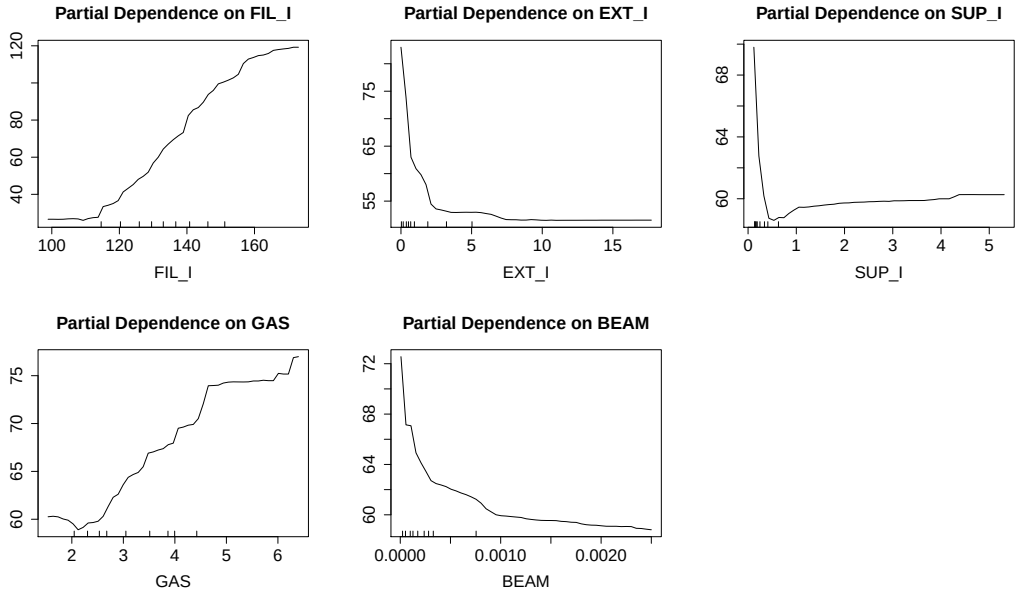

Figure 5.11: Partial dependence plots of the top 5 contributing predictors out of the full random forest model with 15 predictors.

small values of `FIL_I` are the strongest indicator that a filament break is imminent. `GAS` basically shows the same kind of dependence, but is far less relevant. `EXT_I` and `BEAM` show a decreasing dependence but as these values strongly depend on different production recipes, interpretation is not as straight forward as with `FIL_I` and `GAS`. Finally, for values of `SUP_I` between 0 and 0.5, `NextPM` rapidly decreases, but as `SUP_I` increases, the dependence changes and a slight increase in `NextPM` is observable.

A random forest model fitted only with these 5 predictors and the default of 500 regression trees is stored in `mod.rf.2`. The best value for `mtry` is calculated to be 4. The output is shown below.

```
Call:
 randomForest(formula = NextPM ~ ., data = xtrain, importance = TRUE,
              mtry = best.m, ntree = 500, na.action = na.omit)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 4

          Mean of squared residuals: 187.6866
                    % Var explained: 82.73
```

Obviously the result has not changed much which is very desirable. The amount of variance explained is only 1% smaller than in the full model and the RMSE is $\sqrt{187.68} = 13.7$ hours. This model is much less complex and is therefore to prefer over the full model. The oob error plot in figure 5.12

shows that the default value of `ntree` = 500 tree models is enough as a slight increase in the oob error can be observed at about 300.

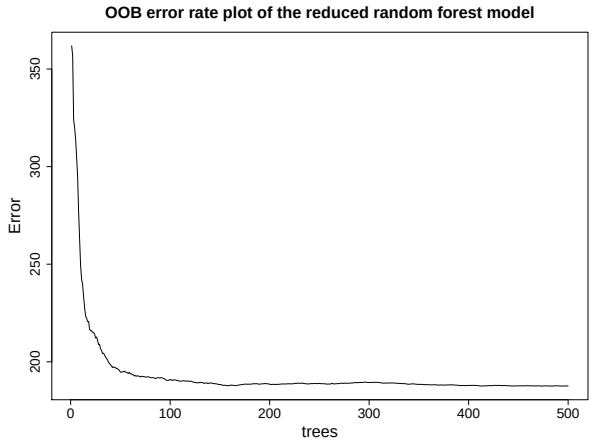

Figure 5.12: The number of trees in the random forest model and the corresponding out-of-bag error rate of the reduced model `mod.rf.2`.

The resulting variable importance plot in figure 5.13 followed by the numerical values of the importance measures show a slightly different order.

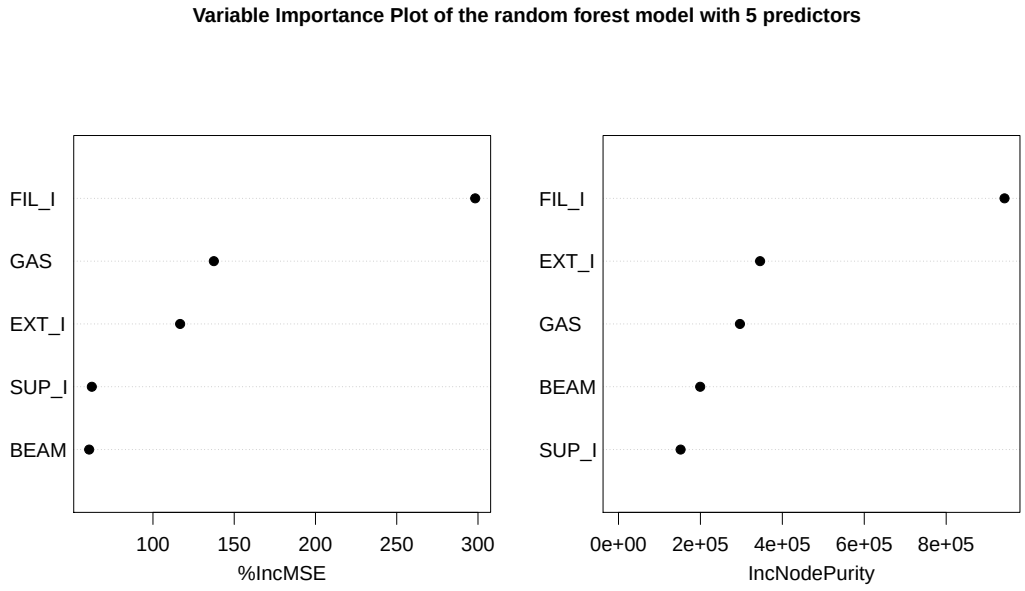

Figure 5.13: Variable Importance plot for the random forest object `mod.rf.2` with the top 5 predictors. The variable importance measures $\text{IMP}^{(1)}$ and $\text{IMP}^{(2)*}$ are shown.

```
        %IncMSE IncNodePurity
FIL_I 298.27989      942453.3
EXT_I 116.71254      345661.7
SUP_I  62.37922      151629.6
GAS   137.46331      296511.6
BEAM   60.74335      199447.4
```

$IMP^{(1)}$ (`%IncMSE` in the table of numerical values) suggests that the model could be further simplified by removing `SUP_I` and `BEAM`. This yields a RMSE of 15.6 hours and an explained variance of `NextPM` of 77%. Therefore the main contributors are obviously `FIL_I`, `EXT_I` and `GAS`.

Figure 5.14 shows a trellis plot of these three predictors. The plot is read from the lower left to the upper right plot. Six scatterplots of `EXT_I` versus `GAS` for different ranges of corresponding `FIL_I` values are shown. The positive relation between `FIL_I` and `EXT_I` is also observable in the linear correlation coefficient of about 0.53 (see also figure 5.4) as is the slightly negative relationship of `FIL_I` and `GAS` (correlation coefficient of -0.37).

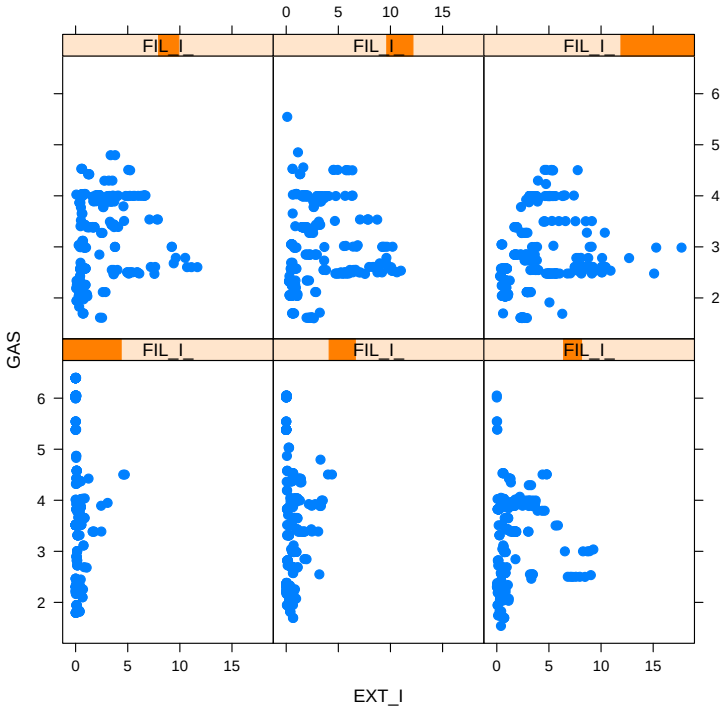

Figure 5.14: Trellis plot of the 3 most important predictors of an implanter.

However, we keep our model with 5 predictors as a typical error rate of 13.7 hours is desirable.

5.4.1 Test data results

With the built random forest model we are now able to test the model using a test data set, i.e. a data set not used to build the model. This gives information about the model's real predictive power.

We use a test data set with implanter machine data from late September and October 2010 with 674 data points which equals 2 filament lifetime cycles. As our model is trained on lifetime cycles of 150 hours at most, we concentrate on a period of 120 hours before a filament break as a prediction for longer periods is not relevant. The response `NextPM` of the test data is shown in figure 5.15.

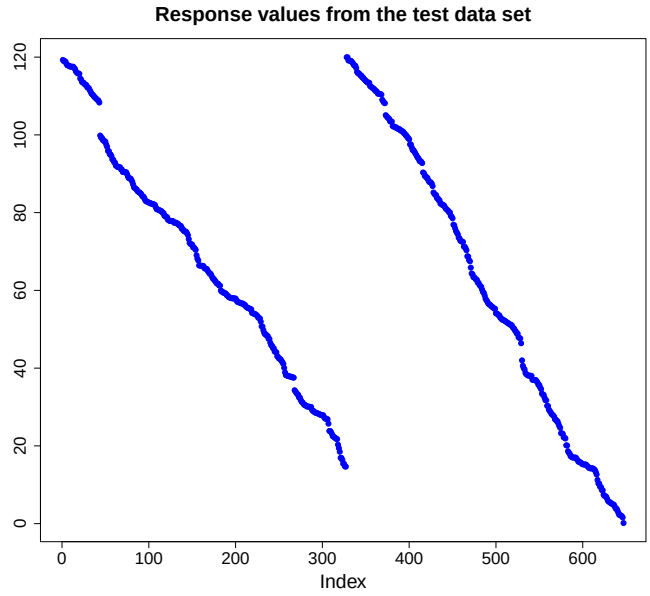

Figure 5.15: Response variable `NextPM` of the test data set of 674 data points or 2 filament lifetime cycles.

The test data is now used to predict the response while keeping the real test response unknown to the model. We then compare the observed response to the corresponding predicted values and calculate the test error. The command

```
pred<-predict(mod.rf.2, newdata=test, type="response")
```

calculates the predicted values to the test data saved in the data frame `test` using the random forest model `mod.rf.2` with the 5 most important predictors. Figure 5.16 shows a graphical comparison of the observed response and the fitted values generated by `mod.rf.2`.

The RMSE of the test data fit is 14.5 hours which is slightly larger than the RMSE of the training data fit of 13.7. The R^2_{pseudo} value of the test data fit is 80.5% where

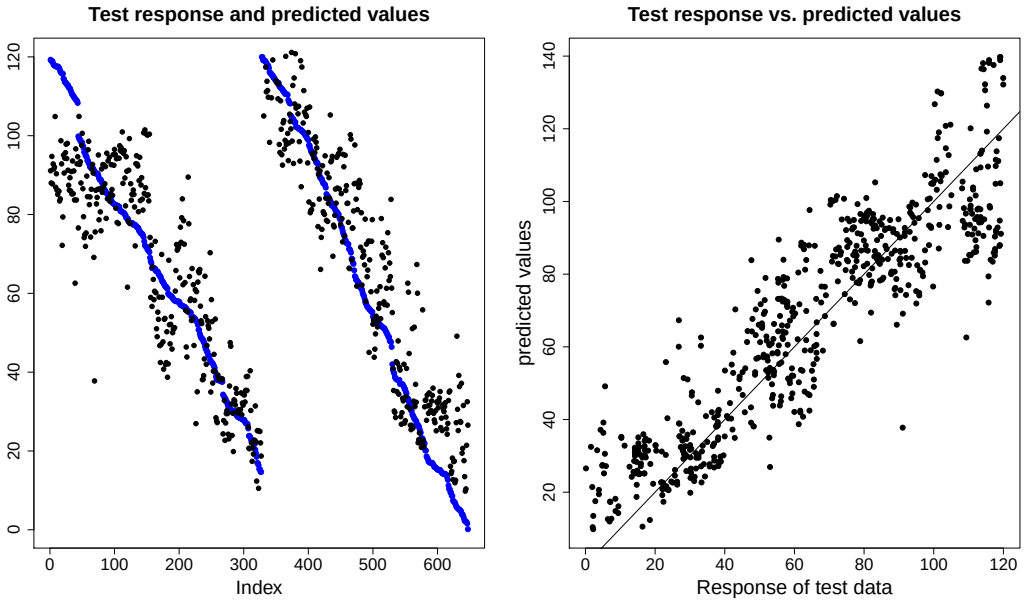

Figure 5.16: Graphical comparison of the real test response and the predicted values, on top of each other (on the left, real response is blue) and against each other (on the right).

$$R^2_{pseudo} = 1 - \frac{\frac{1}{n_{test}} \sum_{i=1}^{n_{test}} (\texttt{test\$NextPM[i]} - \texttt{pred[i]})^2}{\mathrm{Var}(\texttt{test\$NextPM})}$$

with n_{test} being the number of observations in the test data set.
The error can be further decreased by taking moving average values over the last 5 fitted values instead of the fitted values itself. The resulting RMSE reduces to 12.7 hours and $R^2_{pseudo} = 84.7\%$. In the critical period of 72 hours before the actual filament break the RMSE is 12 hours only. The respective plots can be seen in figure 5.17.

In summary, we have found a random forest model for predicting the remaining lifetime of the filament in an implanter tool with a typical error of 12.7 hours when taking moving averages over the last 5 fitted values and with a typical error of 12 hours in the critical 72 hours time frame before the filament will break.

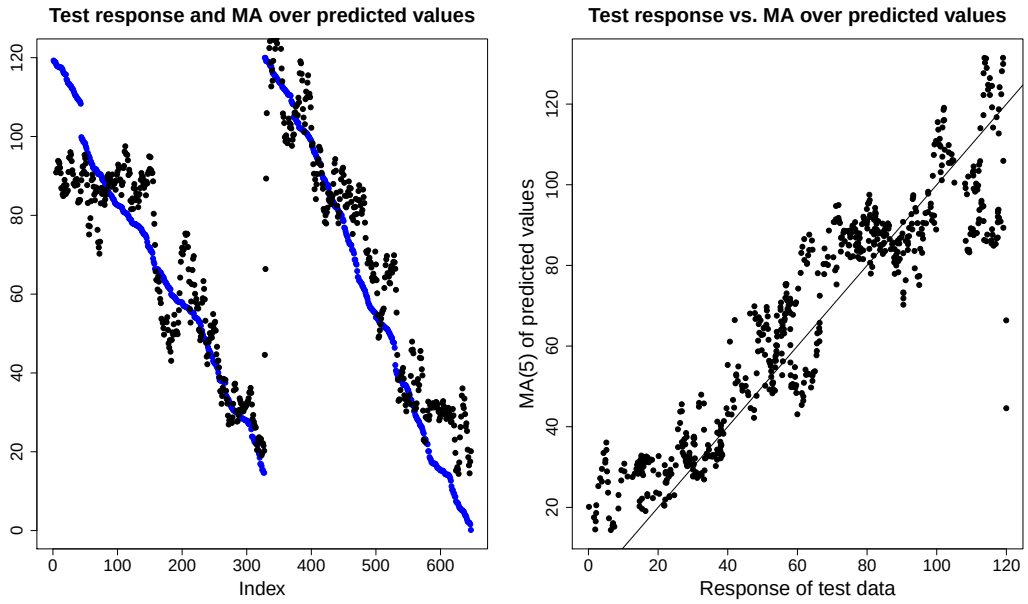

Figure 5.17: Graphical comparison of the observed test response and the moving average over the last 5 predicted values, on top of each other (on the left, observed response is blue) and against each other (on the right).

Chapter 6

Arcing on the Wafer Surface

6.1 Introduction

The problem of arcing on a wafer surface has increasingly become a critical one since it can result in both damage on the wafer surface itself and on the tool chamber. Arcing is mainly due to electrical discharging during certain production steps but the main influential parameters are typically unknown. Therefore the detection of arcing as fast and as reliable as possible is very important for avoiding damages and cost intensive maintenance activities. Figure 6.1 shows a magnification of an arcing defect on a wafer surface.

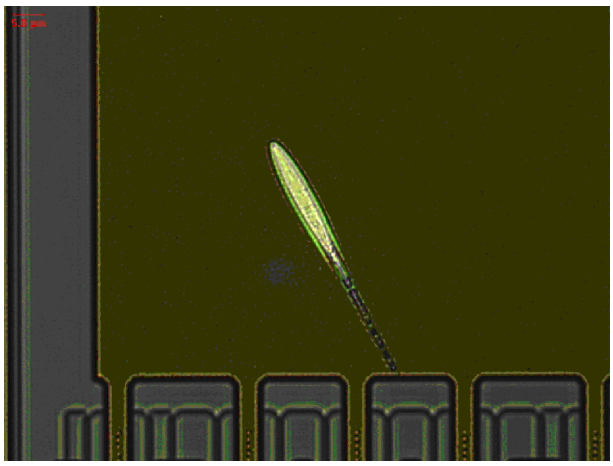

Figure 6.1: Arcing on a wafer surface as occured during a CVD process.

We focus on arcing occuring in a process chamber of a CVD tool. Since there are no shades of arcing as it can only occur or not occur we model it as a classification problem. A wafer in class 1 shows arcing after defect measurement and a wafer in class 0 does not, thus we have a binary response. In addition, when creating a training data set for eventually fitting a classification model, we tried to take only wafers as class 0 which do not show any other kind of defect. This results in an inhomogeneous training data set.

The data set consists of $n = 106$ data points.

The main problem with the data set is the data quality on austriamicrosystems' CVD tool. Before the 25 tool variables are aggregated and the 40 predictors are generated the tool variables are measured during several process steps. Often, these measurements are inaccurate. For example, if the number of samples collected is smaller than the number of samples that should have been collected during the process window or if the time stamp for sample n is lower than the time stamp for sample $n-1$. These and other problems in collecting the data naturally have a negative influence on the aggregated values, resulting in less accurate data. Information about the quality of measurement is summarized in the data quality value. It is calculated for every aggregated data sample. It is a value between 0 and 1 where 0 means the worst possible data quality and 1 means best possible quality. In case of negative values, some entries in the data sample are not available at all. However, the exact method of how this value is calculated is confidential.

Naturally, a fluctuating data quality has a negative influence on modelling both classification and regression problems. Therefore, one has to keep these issues in mind when modelling the arcing problem.

6.2 Data analysis

We start with $p_{initial} = 40$ predictor variables and $n = 106$ data samples in our training data set with 28 arcing (class 1) and 78 non-arcing (class 0) wafers. This inbalanced proportion is due to the irregular occurence of the problem. The data comes exclusively from the CVD tool's chamber A as it is the only chamber where arcing is a problem.

At first we have a look at the predictors. The tables A.1 and A.2 in the appendix give a tabular overview.

Out of the CVD engineers' experience with arcing, information about the dome heater, especially `DHside`, and information about the radio frequency power are considered influential for the problem. However, the reasons for arcing remain largely unknown.
At first we remove high correlated predictors and therefore we have a look at the correlation matrix. Figure A.1 in the appendix visualizes linear correlations in a `lattice` plot.

There are no constant predictors in the data set. We remove predictors that show an absolute value of pair-wise correlation of 0.8 or greater. 29 predictors remain.
Out of these 29 we graphically compare non-arcing and arcing observations for different predictors by using boxplots with confidence intervals for the medians. 5 predictors show an obvious significant difference between the classes (see figure 6.2).

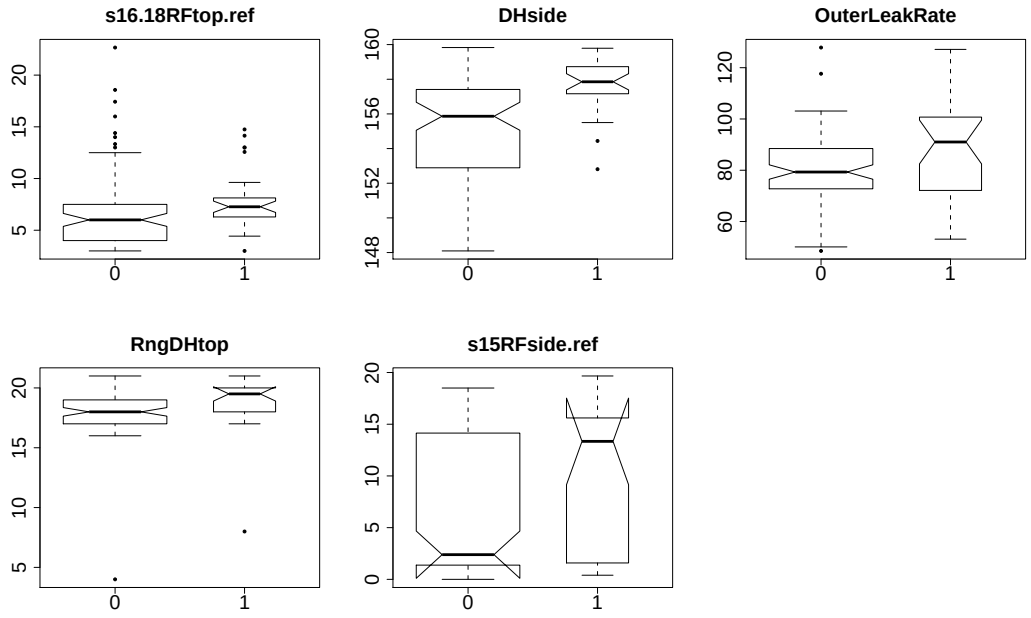

Figure 6.2: Boxplots of the 5 predictors that show an obvious significant difference of non-arcing (left) and arcing (right) wafers in their medians.

All 5 predictors show a higher median in the arcing case. Clearly, `DHside` and `s15RFside.ref` show the largest differences between classes.

By looking at the plot in figure 6.3 of the dome heater side indicator over time of 35 wafers with a total of 8 arcing wafers, it is observable that when arcing occured the value of the dome heater side temperature was at least above 157, in 6 out of 8 cases even above 158. It is suggested by engineers that the reason for the numerous up-and-downturns are so called *3x-clean* procedures of the chamber. It seems, arcing is more likely to occur after a chamber clean procedure which is done after 3 processed wafers. A similar conclusion can be made by looking at the plot of `RngWaferTemp` over time of the same wafers. In all arcing cases, the values of the predictor were at or above 15 after 3x-cleans. Currently, the exact reasons behind arcing occurence after 3x-cleans remain unclear. Figure 6.4 shows the corresponding plot of `RngWaferTemp` where the effect is visualized. The arcing cases in both plots are marked as black dots.

Plotting both predictors against each other using all of the 106 data points leads to the same conclusion. Figure 6.5 shows the plot with the arcing wafers marked red as 1 and the non arcing wafers marked blue as 0. Arcing is more likely to occur if both the dome heater side temperature and `RngWaferTemp` (or `WaferTemp` as they are highly correlated) are large. If both are small, arcing seems unlikely. Thus, `RngWaferTemp` also seems to contribute to the problem.

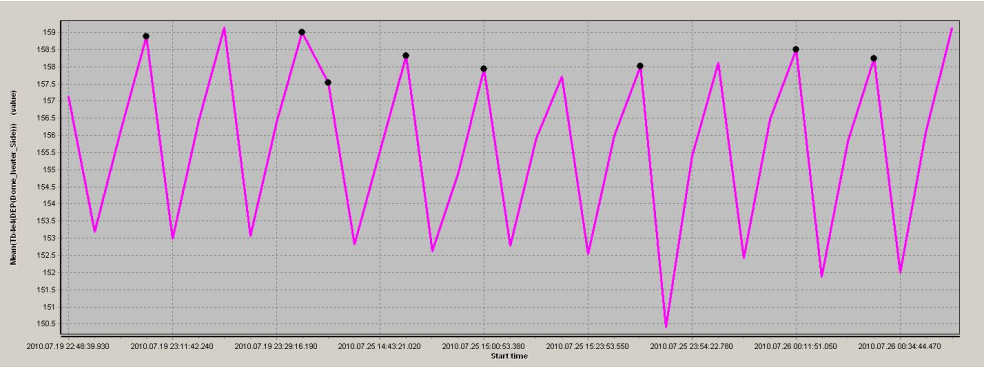

Figure 6.3: Values of `DHside` over time of 35 wafers processed in chamber A of the CVD01 tool during July 2010 with 8 cases of arcing (marked as black dots).

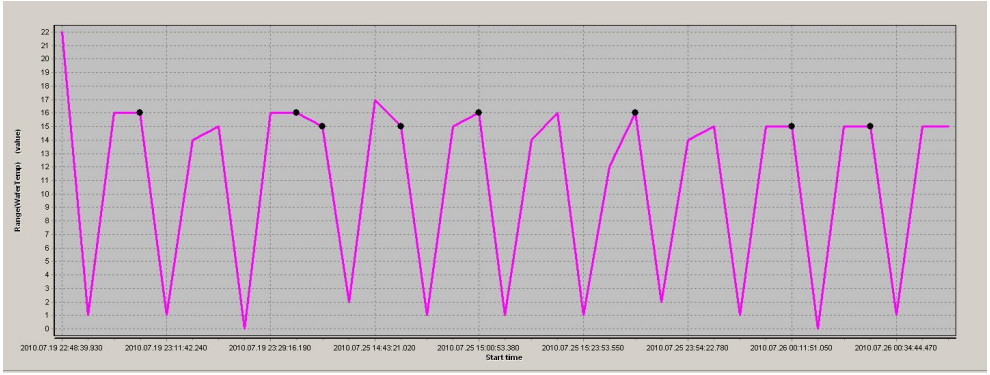

Figure 6.4: Values of `RngWaferTemp` over time of 35 wafers processed in chamber A of the CVD01 tool during July 2010 with 8 cases of arcing (marked as black dots).

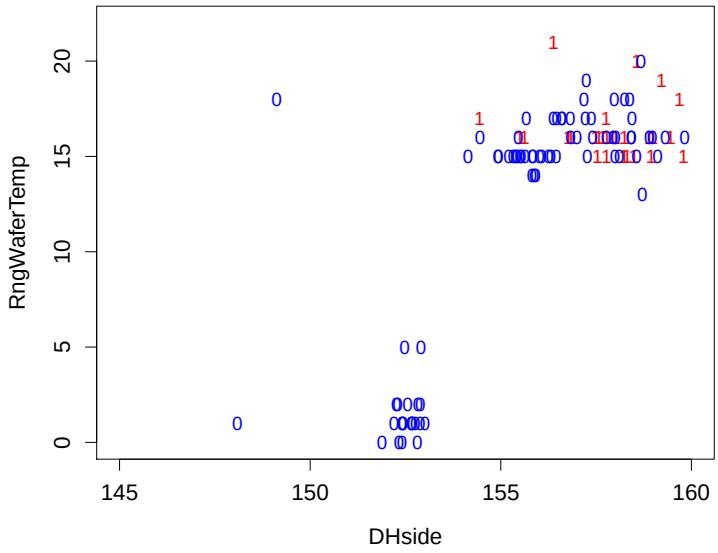

Figure 6.5: The dome heater side temperature plotted against `RngWaferTemp` with arcing wafers marked red as 1 and non arcing wafers marked blue as 0.

6.3 Classification trees

The next step is to fit classification tree models to the data to get an insight in data structure and predictor relationships. Figure 6.6 shows the graphical output of an `rpart` model object.

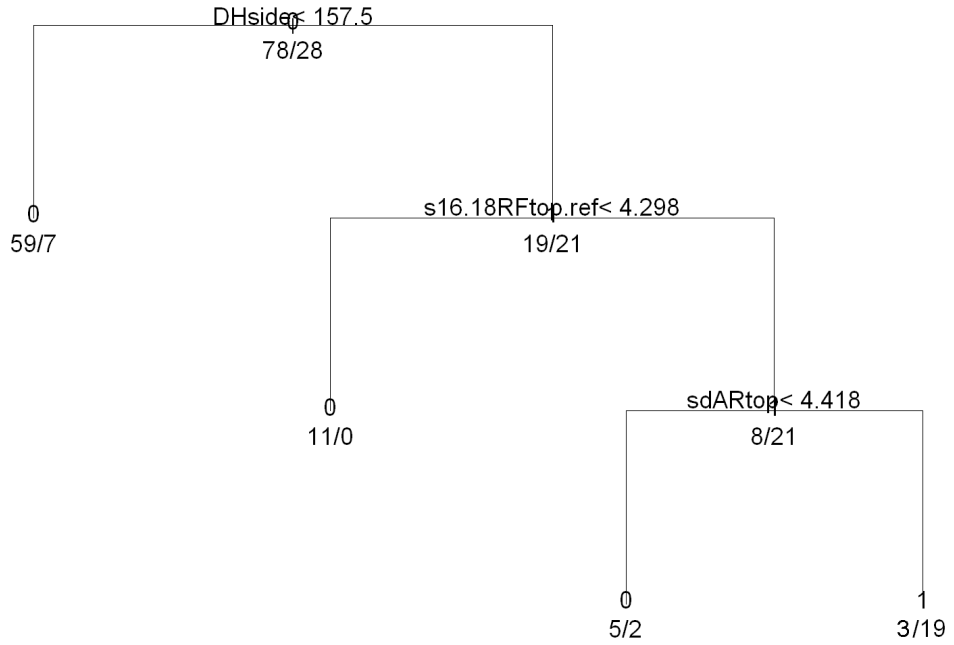

Figure 6.6: Classification tree model created using the `rpart` R routine with 29 uncorrelated predictors.

As already supposed, `DHside` seems to be the most important predictor. The split treshold of 157.5 is also observable in figure 6.3. All observations corresponding to dome heater side temperature values smaller than this treshold fall into the terminal node in the left branch. Obviously, no further reduction in the node impurity is possible in this case. The proportion is 59 class 0 observations to 7 class 1 observations. Thus the node is clearly labeled 0 with a misclassification error of about 10.6%. Furthermore we can observe that 7 out of the 28 arcing wafers, i.e. 25%, have a dome heater side temperature value smaller than 157.5.

On the right hand side of the tree the reflected radio frequency power yields the biggest reduction in impurity. For `s16.18RFtop.ref` parameter larger than about 4.3, the standard deviation of the Argon flow seems to be important. The remaining 19 arcing cases fall in the terminal node corresponding to a gas flow value of 4.418 or larger. The terminal node is labeled as class 1 node with a misclassification rate of 13% as 3 non-arcing wafers also fall in this data partition.

For values of `s16.18RFtop.ref` smaller than 4.3, we find a terminal node labeled class 0 with only non-arcing wafers and thus 0% misclassification rate.

The plots in figure 6.7 and 6.8 give an additional graphical overview of the data situation and further visualize the above tree structure.

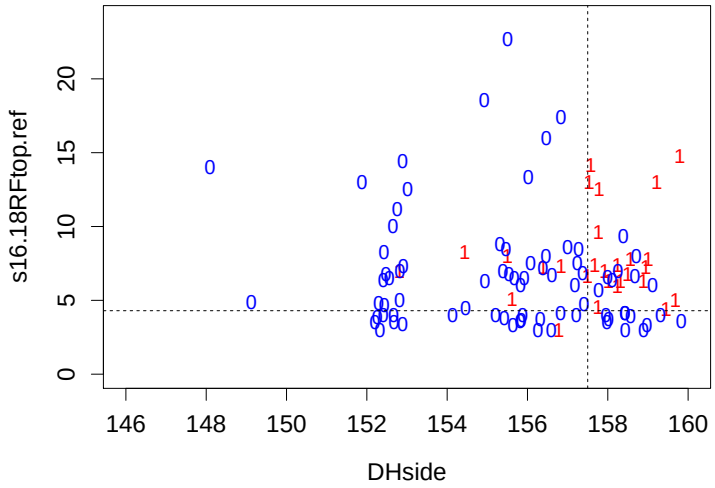

Figure 6.7: The dome heater side temperature plotted against `s16.18RFtop.ref` with arcing wafers marked red as 1 and non arcing wafers marked blue as 0. The dashed lines show the tresholds at which splits are performed in the classification tree model.

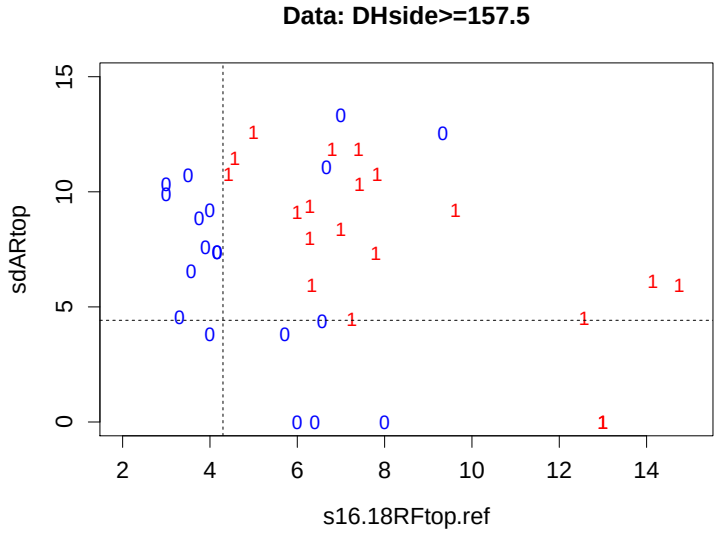

Figure 6.8: Further visualization of the splits in figure 6.6. Only data points in the `DHside` ≤ 157.5 data region are considered. Arcing wafers marked red as 1 and non arcing wafers marked blue as 0.

The classification tree model yields the confusion table printed in table 6.1.

	Arcing predicted	No Arcing predicted	Model Error
Arcing	75	3	3.8%
No Arcing	9	19	32.1%
Use Error	10.7%	13.6%	11.3%

Table 6.1: Confusion table of the `rpart` classification tree model.

An overall model error of 11.3% seems acceptable but there is a problem. The model misclassifies non arcing wafers in nearly one third of the cases, i.e. if a wafer of class 0 is given to the model there is a probability of 32.1% that the model wrongly classifies it as showing the arcing problem. In practical usage, this leads to an increased number of false alarms. The remaining error rates seem acceptable.

Furthermore, the model has a kappa coefficient of about 0.68 which is a good agreement rate.

The classification tree generated by means of unbiased recursive partitioning is presented in figure 6.9.

Again, the dome heater side variable and the reflected power variable are in the tree with similar split tresholds. Terminal node 6 has the same class proportion and cases as in the first tree model. However, now `RngWaferTemp` plays a role for values of the dome heater side smaller than 157.41. A further split is performed at a `RngWaferTemp` treshold of 15 as is already observable in figure 6.4. This results in an entirely pure terminal node 3 of only class 0 cases and another terminal node labeled class 0 with a misclassification error of about 30%. The only terminal node labeled as class 1 is the same as in the `rpart` tree after splitting `s16.18RFtop.ref` at 4.16 (terminal node 7). The further split at the gas flow parameter is omitted by the `party` method and thus a terminal node with 21 arcing wafers and 8 non arcing wafers follows.

The results from the model evaluation change slightly. Table 6.2 shows the confusion table.

	Arcing predicted	No Arcing predicted	Model Error
Arcing	70	8	10.2%
No Arcing	7	21	25.0%
Use Error	9.09%	27.58%	14.14%

Table 6.2: Confusion table of the `rpart` classification tree model.

The overall error has increased to about 14.1%. The misclassification of non arcing wafers has imporved but the use error became larger. The error that is made when a wafer is predicted non arcing has nearly doubled to about 27.6%. In practical usage, this means that the prediction of non arcing wafers is less reliable.

The value of Cohen's kappa is 0.64 and thus only slightly smaller than in the `rpart` model. However, it still means good agreement.

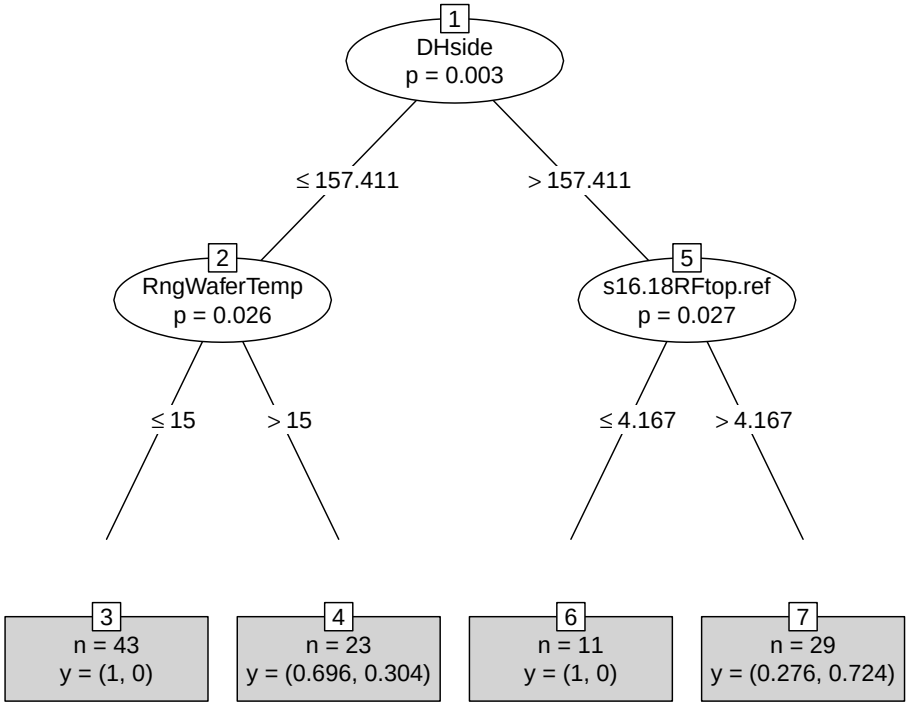

Figure 6.9: Classification tree model created using the `party` R routine with 29 uncorrelated predictors.

In summary, the classification tree models offer an acceptable fit but there is room for improvement.

6.4 A Stochastic Gradient Boosting model

We are now ready to apply stochastic gradient boosting to the arcing classification problem. For constructing a corresponding R model we use the `caret` package (see [16] Kuhn, 2008) as a wrapper for the `gbm` package. A description of the functionality of the `caret` package can be found in the appendix.

We start with a full model using all 29 predictors to get an overview of variable importance measured with boosting's built in method. The 20 most important predictors are listed below with their importance values scaled between 0 and 100.

```
DHside                    100.000
OuterLeakRate              77.588
s16.18RFtop.ref            61.232
RngWaferTemp               46.315
DHtop                      26.598
DEPRFside                  24.016
s16.18RFtop                23.913
DEPRFbias.ref              23.839
sdDEPRFbias                20.960
s15RFside.ref              19.884
sdARtop                    19.420
DEPCHPress                 16.322
DEPRFside.ref              15.664
DEPRFtop                   13.213
InnerLeakRate              12.810
s15RFside                  12.114
s15RFbias.ref              10.942
DEPRFtop.ref               10.856
s15RFtop                    9.392
DurDep                      9.151
```

As supposed, the dome heater side temperature is the most important predictor. `s16.18RFtop.ref` as well as `RngWaferTemp` are also important as supposed from the classification tree models. Furthermore, the outer helium leak rate indicator has influence. A difference between classes is already observable in figure 6.2.

Now we fit a model with the 4 most important predictors found above and `sdARtop` as observed to be important in the classification tree. An excerpt of the output resulting from the `caret` package's `train` function with `gbm` as the underlying boosting method is shown below.

```
106 samples
5 predictors

  interaction.depth  n.trees  Accuracy  Kappa  Accuracy SD  Kappa SD  Selected
  1                  100      0.841     0.564  0.0953       0.25
  1                  500      0.841     0.572  0.0686       0.174
  2                  100      0.823     0.508  0.0613       0.15
  2                  500      0.84      0.56   0.0435       0.14
  3                  100      0.869     0.671  0.0484       0.0991
  3                  500      0.84      0.575  0.0579       0.148
  4                  100      0.87      0.675  0.081        0.162     *
  4                  500      0.831     0.556  0.0501       0.129
```

The stochastic gradient boosting model uses a shrinkage parameter of 0.075 as this value turned out to give adequate results better than the default value of 0.001. For calculating accuracy and Cohen's kappa values we use 8 fold cross-validation. We tried interaction depths of 1 to 4 and 100 or 500 iterations. The `train` function now finds the model parameter constellation that yields the highest average classification accuracy over all cross-validation passes. The

best overall model uses a variable interaction depth of 4 and 100 tree models, i.e. 100 boosting iterations.

A resulting average accuracy of 0.87 (i.e. 13% model error) calculated over all cross-validation runs seems highly satisfying as does the average kappa of 0.675. The standard deviations of these values also seem acceptable, although the kappa values are somewhat instable over different cross validation passes. However, using an interaction depth of 3 and 100 iterations yields nearly the same accuracy (0.869) and lower standard deviations but is less complex. Thus an interaction depth of 3 seems recommendable.

The boosting performance plot on the left hand side in figure 6.10 shows the change in the deviance (Bernoulli deviance) over iterations.

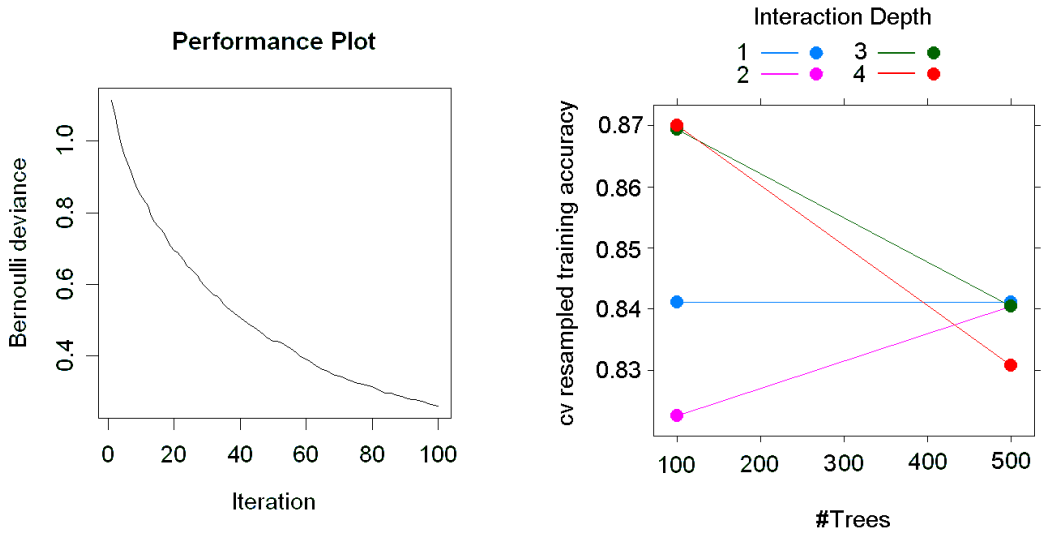

Figure 6.10: Performance plot and plot of model accuracy values for different interaction depths and iterations of a boosting classification model for arcing with 5 predictors.

Clearly, the remarkable improvements in the deviance come early. The less complex model with only 100 iterations seems suitable. The second plot in figure 6.10 shows the change in accuracy for different model parameters.

As seen before, the less complex model with an interaction depth of 3 yields nearly identical results at 100 iterations but with more stability in both accuracy and kappa values.

For evaluating the model fit itself, we have a look at the confusion table in table 6.3.

The confusion table of the boosting model shows a clear improvement over the single classification tree model. The overall error of 1.88% seems highly adequate with only 2 false negatives, i.e. wafers predicted as showing arcing when they do not in truth. Thus, the problem of false alarms improved a lot. The variable importance plot of the model object is given in figure 6.11.

	Arcing predicted	No Arcing predicted	Model Error
Arcing	78	0	0%
No Arcing	2	26	7.1%
Use Error	2.5%	0%	1.9%

Table 6.3: Confusion table of the stochastic gradient boosting model with 5 predictors.

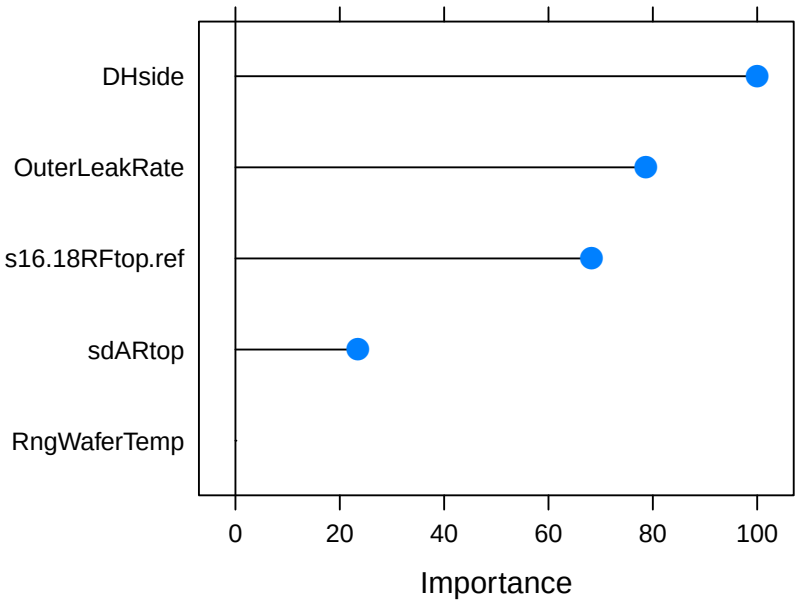

Figure 6.11: Variable importance plot of the stochastic gradient boosting model with 5 predictors.

Obviously, `RngWaferTemp` does not contribute at all in this reduced model so the model can be further simplified by omitting this predictor. Thus, the 4 most important predictors are

```
DHside
OuterLeakRate
s16.18RFtop.ref
sdARtop
```

The boosting model using only these 4 predictors has cross validation results similar to the model using 5 predictors. The best model uses an interaction depth of 4 and 500 tree models. This results in an accuracy of about 0.85 (i.e. model error of 15%) and a slightly lower kappa value of 0.58. Thus, the model is slightly weaker, but the difference is not dramatical. However, the best model needs more iterations to achieve its results (500 compared to 100) which yields a higher complexity. It seems using 100 instead of 500 iterations does not result in a dramatic decrease in model quality, although the performance plot of a 100 iteration model compared to a 500 iteration model exhibits a difference. See figure 6.12 and figure 6.13.

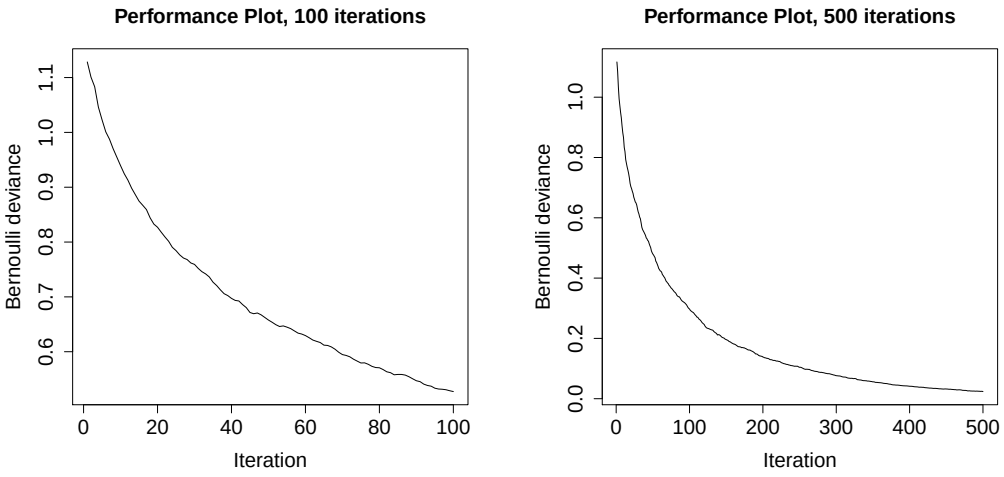

Figure 6.12: Performance plot comparison of boosting classification models for arcing with 4 predictors.

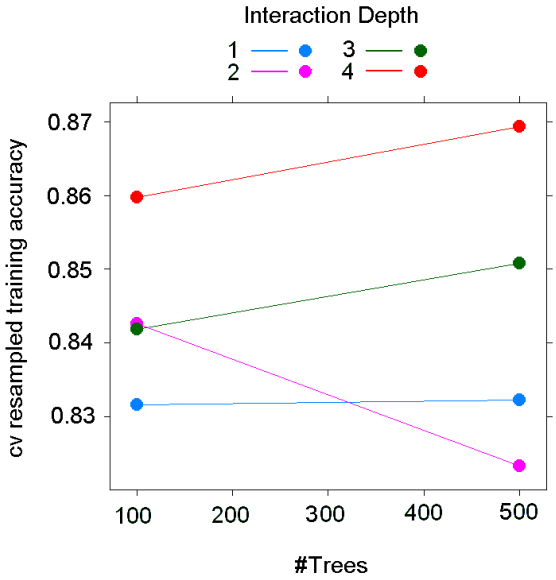

Figure 6.13: Plot of model accuracy values for different interaction depths and iterations in a boosting classification model for arcing with 4 predictors.

	Arcing predicted	No Arcing predicted	Model Error
Arcing	78	0	0%
No Arcing	0	28	0%
Use Error	0%	0%	0%

Table 6.4: Confusion table of the stochastic gradient boosting model with 4 predictors.

The confusion table can be seen in table 6.4. The model yields a perfect confusion table and thus a perfect fit of the data as all wafers from the training data set are classified correctly. However, the cross-validation results suggest an overfitting of the data as the calculated accuracy and kappa statistics are not as favourable.

6.4.1 Test results

Results from applying the model on real test data are not as good as the model quality values and confusion tables suggest. The main problem of the model is the high proportion of false alarms (i.e. false negatives) for arcing as already assumed.

For testing the arcing classification model we used a test data set containing 76 data points (6 lots of wafers) where only 6 wafers show the arcing defect. Classifying these data yields the confusion table shown in table 6.5.

	No Arcing predicted	Arcing predicted	Model Error
No Arcing	60	10	14.0%
Arcing	2	4	33.3%
Use Error	3.2%	71.4%	15.8%

Table 6.5: Confusion table of generated using a test data set with 76 data points: 70 class 0 and 6 class 1 wafers.

As mentioned earlier, the unbalanced data set is due to the infrequent occurence of arcing. Clearly, the biggest problem are the false negatives. If arcing is predicted by the model, about 7 out of 10 predictions are false alarms. On the other hand, 2 arcing wafers are not predicted as such by the model (false positives). It is worth to mention that in 2 out of the 6 tested lots the model works perfectly, predicting every wafer correctly. In two lots, the arcing wafers are predicted correctly while other wafers were false alarms (1 and 2 false alarms) and in the remaining two lots the model does not recognize the arcing wafers at all while producing false alarms (4 and 3 false alarms).

The overall model error is not very meaningful as the data situation is too imbalanced. In part, the problems are clearly related to the data quality issue. An improvement in data quality should generally improve the model and increase accuracy. The highly imbalanced data situation may also have a negative influence. However, the problems may also originate from the boosting's problem with overfitting as cross-validation and test data results barely mirror the goodness of fit of the training data. Compared to the boosting model, a random forest classification model fitted to the same data yields somewhat different results with the test data.

6.4.2 Comparison with a Random Forest model

In theory, the problem of overfitting is not as visible in random forest as it is in boosting (e.g. see [4] Breiman, 2001). Thus, we fit a random forest classification model for comparison.

Again, we use the top 4 most important predictors, i.e.

```
DHside
OuterLeakRate
s16-18RFtop-ref
sdARtop
```

The confusion table constructed using the training data to evaluate the model fit itself is shown in table 6.6.

	No Arcing predicted	Arcing predicted	Model Error
No Arcing	78	0	0%
Arcing	0	28	0%
Use Error	0%	0%	0%

Table 6.6: Confusion table of the random forest classification model with 4 predictors constructed using the training data.

Like the stochastic gradient boosting model it yields a perfect confusion table, i.e. it perfectly fits the training data. However, a better information about model quality gives the confusion table constructed using the out of bag data given in table 6.7.

	No Arcing predicted	Arcing predicted	Model Error
No Arcing	72	6	7.7%
Arcing	13	15	46.4%
Use Error	15.3%	28.6%	17.9%

Table 6.7: Confusion table of the random forest classification model with 4 predictors constructed using the out of bag data.

The model error of about 18% calculated using out of bag cases is slightly higher than the 15% achieved in the boosting model with the same 4 predictors using cross-validation. Furthermore, in addition to the 4 predictors used, `RngWaferTemp` does not give any further improvement as the resulting confusion table is the same.

Judging from the out of bag error estimates, the false alarms are reduced as only 6 wafers are classified as having arcing when they do not. The main problem are the false positives, i.e. the model does not recognize arcing wafers as such. Out of the 28 arcing wafers, the model does not recognize 13 arcing wafers (46.4%).

However, the two models are best compared using the test data set. Table 6.8 shows the confusion table.

The overall test error is better than for the boosting model, but the imbalanced class proportions have to be kept in mind. More importantly, the random forest model has less false alarms as a non arcing wafer is wrongly classified as having arcing only 5 times compared to 10 times in the boosting case. Thus, it has fewer false negatives. However, there is still a problem with the false positives.

	No Arcing predicted	Arcing predicted	Model Error
No Arcing	65	5	7.1%
Arcing	3	3	50%
Use Error	4.4%	62.5%	10.5%

Table 6.8: Confusion table of the random forest classification model with 4 predictors constructed using test data.

The model fails to recognize an arcing wafer in 3 cases, i.e. 50%. Here the random forest model performs worse than its boosting counterpart.

In summary, the random forest model has a slightly lower overall model error than a stochastic gradient boosting model with the same predictor set. Furthermore, it produces less false alarms. However, it has a higher proportion of false positives.

6.5 Summary

In summary, we found classification models for the problem of arcing on a wafer surface that perfectly classify the data used to construct the model where the outcome is known. Through statistical analysis, the main influential predictors were found. The models are able to recognize wafers showing arcing but in test phases they are either prone to producing false alarms or fail to recognize defect wafers. Clearly, one reason for these problem are data quality issues and a highly imbalanced data situation but methodical issues such as overfitting may also play a certain role.

Chapter 7

Modelling Film Thickness on a PVD tool

7.1 Introduction

Equipment qualification and tool monitoring is a cost and time consuming procedure for semiconductor manufactoring and especially for metal film deposition with a PVD tool. Therefore an accurate knowledge of the thickness of the deposited film without actual measurement (*virtual metrology*) is highly desirable.

After the PVD film deposition process the wafers are moved to metrology tools. The measurements were done over the course of several weeks. The thickness of the film deposited on the wafer surface is measured on five sites. Every wafer is assigned its mean film thickness. We are now interested in modelling this mean film thickness using tool parameters. The film thickness is measured in nanometre (10^{-9} metres).

7.2 Data analysis

For modelling we use a set of $n = 209$ data points. During the process, 17 predictors are calculated out of tool parameters. A table of all predictors (table B.1), summary statistics as well as a correlation matrix (figure B.1) can be found in the appendix.

First we have a look at the response variable `Thickness`. Its left skewed distribution is shown in figure 7.1.

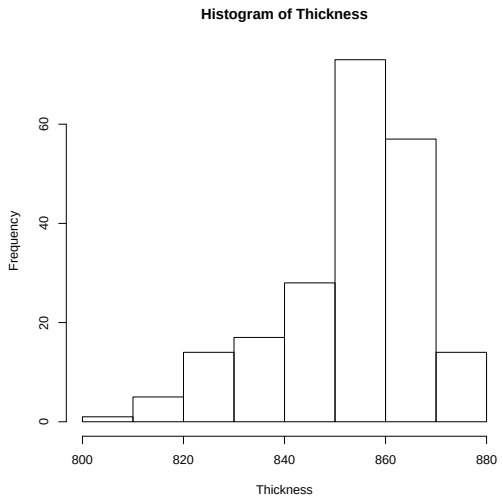

Figure 7.1: Histogram of the mean thickness of film deposited in two different process chambers of a PVD tool.

The data set contains data from two different PVD process chambers (chamber 3 and chamber 4). Figure 7.2 shows a comparison of `Thickness` for the different chambers.

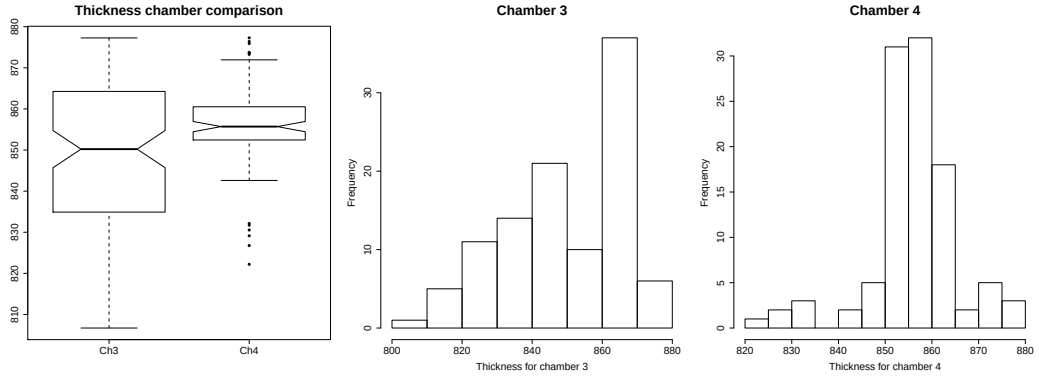

Figure 7.2: Comparison of the mean deposited film thickness between two different process chambers.

The plots show a difference between the two process chambers in both the boxplot comparison and the distributions shown in the histograms. Thus we add `chamber` as a categorical predictor.

7.3 Regression Trees

First we look at regression tree models to get an impression of the structure of the data and variable importance. Figure 7.3 shows the graphical output of an `rpart` object. The model uses 1 categorical and 10 continuous predictors.

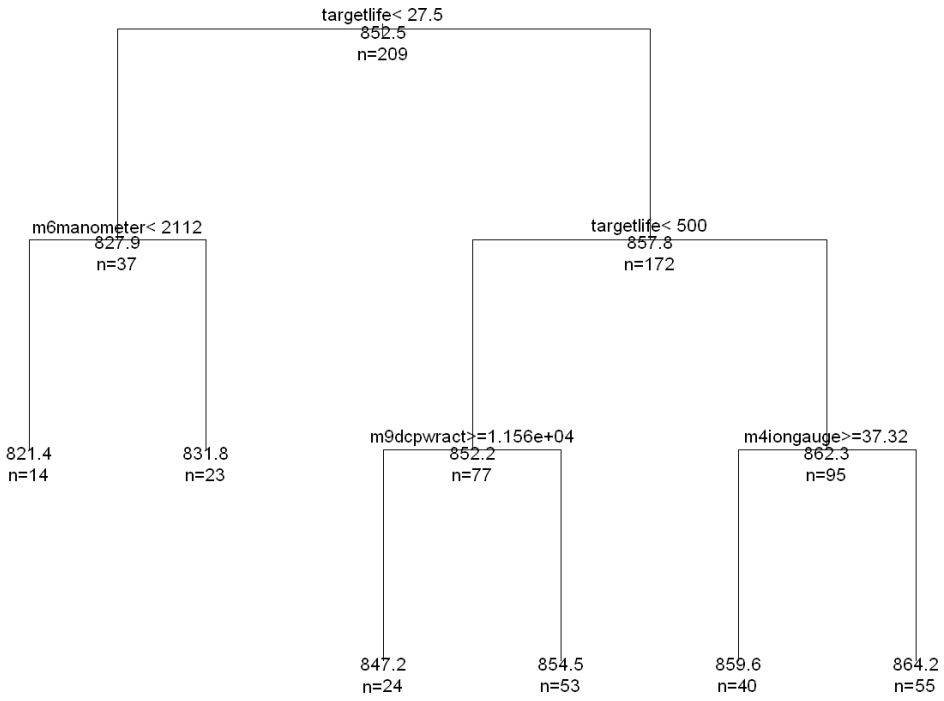

Figure 7.3: Regression tree model using 11 uncorrelated predictors for the mean deposited film thickness on a PVD tool.

`targetlife` contributes the most. For very small values (< 27.5, left branch) the deposited film is thinner. In this data partition, the capacitive manometer parameter `m6manometer` is important. However, only 37 cases fall in this branch. The bigger part of the data (172 cases) is found in the right tree branch. For values of `targetlife` between 27.5 and 500, i.e. typically small values to values slightly above average (see the summary statistics in the appendix), the film thickness is around its mean of 852.4 (77 cases). The remaining greater thickness values can be found in the right-most branch (95 cases). About half of all cases (105) show a `targetlife` greater than 500 and `m4iongauge` ≤ 37.3. The film thickness is at its highest level with this data constellation.

The tree in figure 7.4 created using the `party` algorithm also recognizes `targetlife` as the main contributor in first and second level splits.

The 37 data cases with a thickness value well below average are again separated from the rest (terminal nodes 3, 5 and 6). Among these cases, the predictor `m6dccurr` is used for further splitting. However, it splits only 26 cases.

For thickness values around average in the right branch of the tree the procedure separates the data by the process chamber (61 cases in node 8). In chamber 4, the deposited film is thicker than in chamber 3. Finally, if `targetlife`

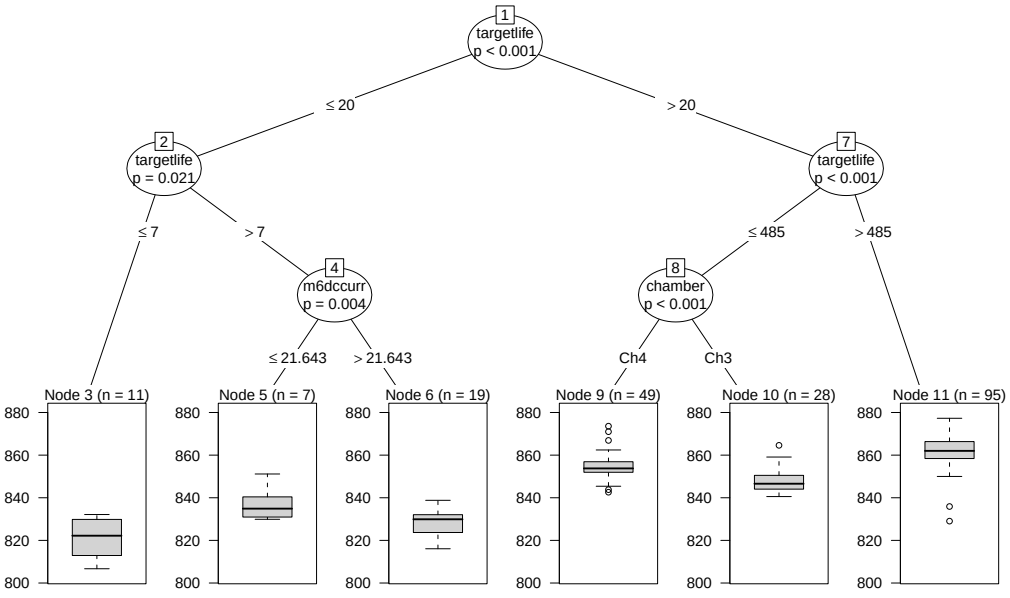

Figure 7.4: Regression tree model using the `party` routine and 11 uncorrelated predictors for the mean deposited film thickness on a PVD tool.

is slightly above average (greater than 485), the resulting film thickness is the greatest (117 cases, terminal node 11).

7.4 Random Forest Model

We continue with a random forest model. The output of a random forest using all 11 predictors is shown below.

```
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 2

          Mean of squared residuals: 56.22696
                    % Var explained: 72.63
```

The model has an amount of explained response variance of about 73% and a resulting RMSE of 7.5 nanometres which seems acceptable. A linear model using the same set of predictors yields a coefficient of determination (R^2) of 59% and a root mean squared error of about 9 nanometres. Thus the random forest model seems to perform better than a comparable linear model.

The model can be further simplified by using only 200 tree models instead of 500. The resulting change in the model quality is not significant as suggested by figure 7.5.

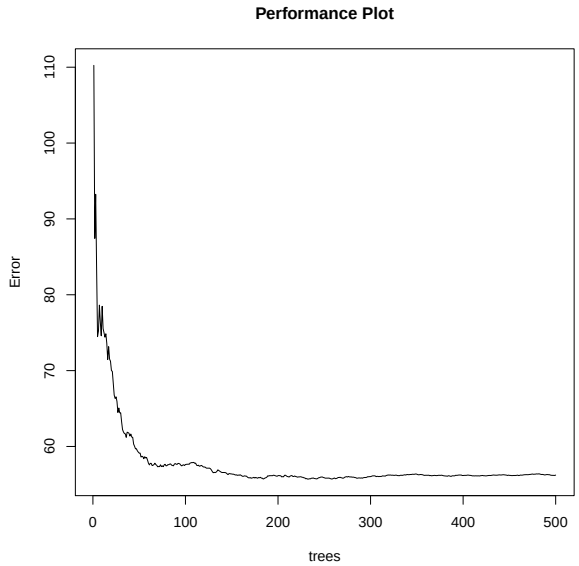

Figure 7.5: The number of trees against the corresponding OOB error rate of a random forest model for the mean film thickness.

The partial dependence plots of the random forest model are shown in figure 7.6.

The partial dependence plots show the positive relationship of `targetlife` and the response. `m6dccurr` is negatively related, however when interpreting the

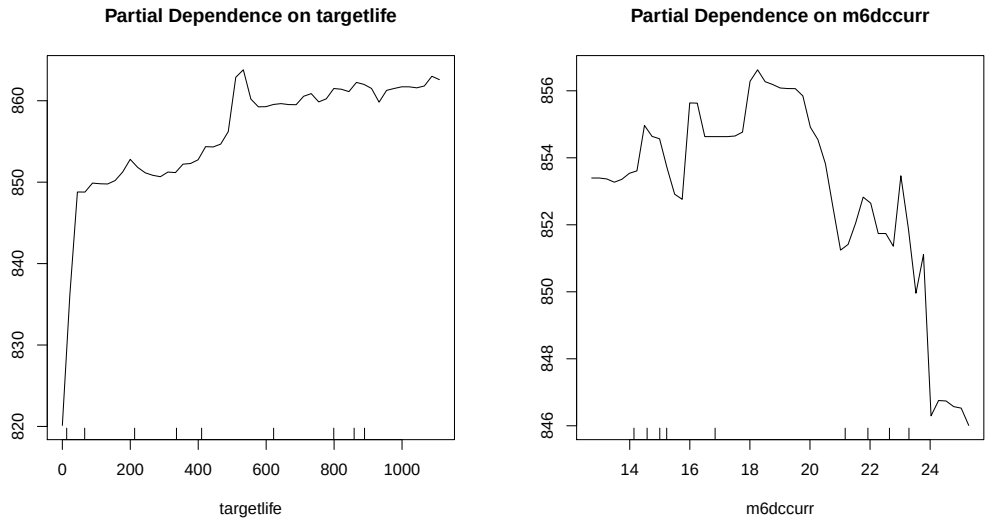

Figure 7.6: Partial Dependence Plots of the 2 most important continuous predictors of a random forest model for the mean film thickness.

plot one has to keep in mind the difference between the two process chambers. The values of `m6dccurr` are larger for process chamber 3.

However, judging from the variable importance plot in figure 7.7, the influence of the categorical predictor `chamber` is insignificant.

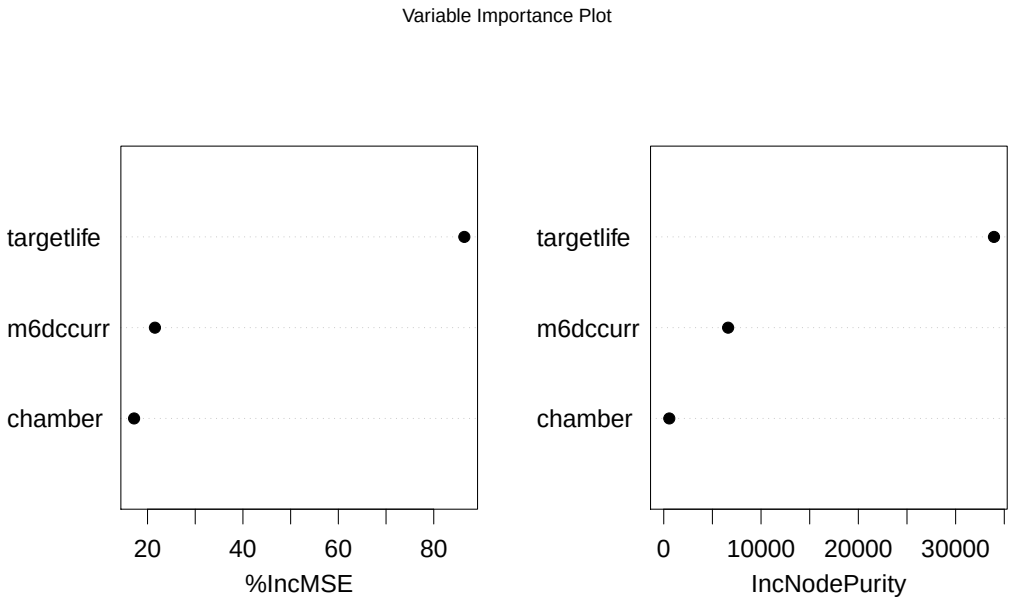

Figure 7.7: Variable Importance Plot of a random forest model with 3 predictors for the mean film thickness.

This is due to the fact that the underlying regression tree models separate the data values corresponding to their chamber difference by splitting the data accordingly. Thus, in contrast to a linear model, there is no need of a further categorical differentiation in our random forest model. This clear advantage of the tree-based method yields further model simplification. The output of a random forest model with 2 predictors is shown below.

```
               Type of random forest: regression
                     Number of trees: 200
No. of variables tried at each split: 2

          Mean of squared residuals: 57.92231
                    % Var explained: 71.8
```

Essentially, the model quality values do not change at all. We still have a RMSE of about 7.5 nanometre and a explained variance of about 72%. Furthermore, the model uses only 200 trees. The plot of the observed response against the fitted values is shown in figure 7.8.

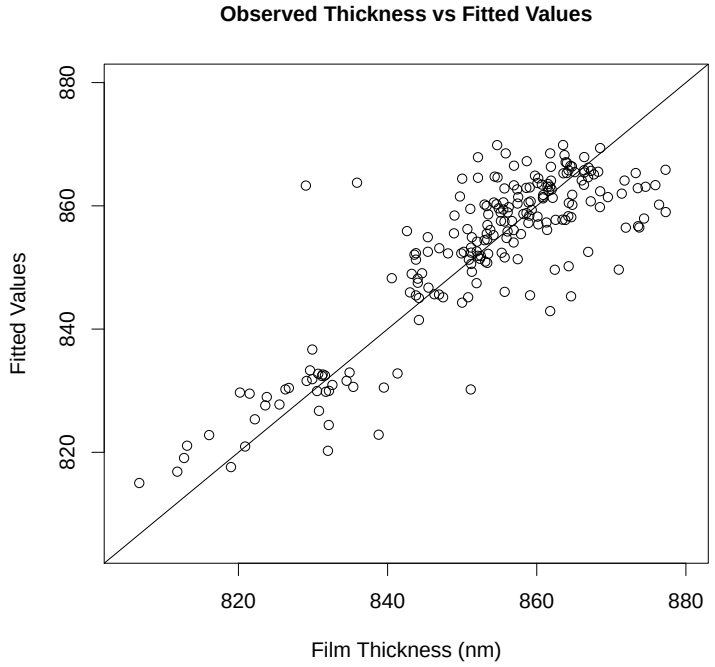

Figure 7.8: Observed mean film thickness against fitted values of a random forest model.

Basically, the model performance has improved for smaller thickness values and remained roughly the same for values above average.

Figure 7.9 shows the observed response values along with fitted values. The specification limits for all quantities are plotted as dashed blue lines.

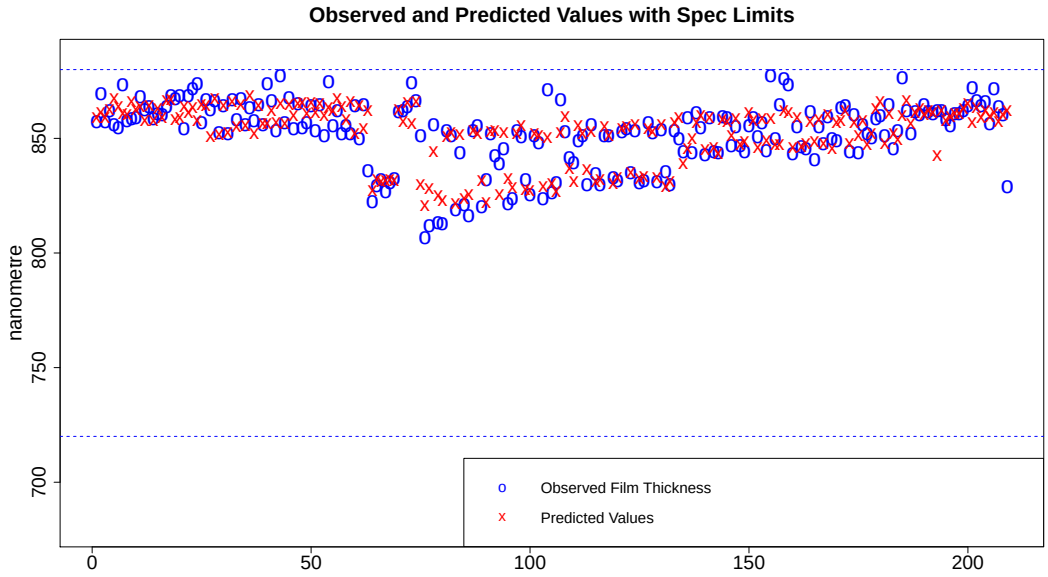

Figure 7.9: Observed film thickness values along with predicted values of a linear model. The dashed blue lines are specification limits.

As with the linear model, the random forest model predictions do not exceed the specification limits when the observed response values also do not, i.e. the model does not produce false alarms. Thus, it also gives the possibility to check if the tool is under control or not.

7.5 Summary

In summary, we have found a model for the thickness of film on a wafer deposited during the PVD process. We found a difference between two process chambers and the 2 most important continuous predictors. In comparison to a linear model, a random forest model performs better using the same set of predictors. The tree-based approach makes a categorical differentiation between the two process chambers redundant. The best model explains about 72% of the variance of the film thickness and has a root mean square error of 7.5 nanometre. With the models it is possible to check if the tool is under control or not.

Appendix A

Arcing on the wafer surface

The data for modelling the arcing problem is collected directly from austriamicrosystem's CVD tool from the tool event "wafer started" until the tool event "wafer finished" with a frequency of one second. The data collection consists of 25 tool variables which are traced during wafer processing. The raw data is then aggregated as mean, max, min, slope, area and/or standard deviation are calculated for relevant process steps or intervals of the tool recipe. Out of the 25 tool variables a set of 40 indicators (calculated summary data) are generated for every wafer. Thus one row in the data set represents aggregated values of 40 indicators for one processed wafer. The names of these indicators contain the aggregation method used (e.g. `Mean`), an optional term which indicates an optional cut of the time window over which the summary data is calculated (e.g. `Tb1e1` for 1 second cut at the start and at the end of the process), the process step or process step window in which the value is calculated and the name of the tool variable.

A.1 Table of variables

Table A.1 and table A.2 give an overview of the names of the indicators along with a description. Furthermore, all indicators have abbrevations by which they are referred to. These abbrevations serve as predictor names in the created models.

Indicator Name	Description	Abbrevation
Duration_Dep	Duration of the deposition process in seconds.	DurDep
Mean(S16_18(ChamberPressure))	Average chamber pressure in process steps 16 to 18.	s16-18CHPress
Mean(S16_18(RF_side_forward))	Average radio frequency power (rf) from the side in process steps 16 to 18.	s16-18RFside
Mean(S16_18(RF_side_reflected))	Average reflected rf power from the side in process steps 16 to 18.	s16-18RFside-ref
Mean(S16_18(RF_top_forward))	Average rf power from the top in process steps 16 to 18.	s16-18RFtop
Mean(S16_18(RF_top_reflected))	Average reflected rf power from the top in process steps 16 to 18.	s16-18RFtop-ref
Mean(Tb1e1(S15(ChamberPressure)))	Average chamber pressure in process step 15.	s15CHPress
Mean(Tb1e1(S15(RF_bias_forward)))	Average rf bias power in process step 15.	s15RFbias
Mean(Tb1e1(S15(RF_bias_reflected)))	Average rf bias reflected power in process step 15	s15RFbias-ref
Mean(Tb1e1(S15(RF_side_forward)))	Average rf power from the side in process step 15.	s15RFside
Mean(Tb1e1(S15(RF_side_reflected)))	Average rf reflected power from the side in process step 15.	s15RFside-rf
Mean(Tb1e1(S15(RF_top_forward)))	Average rf power from the side in process step 15.	s15RFtop
Mean(Tb1e1(S15(RF_top_reflected)))	Average rf reflected power from the top in process step 15.	s15RFtop-ref
Mean(Tb4e4(DEP(ChamberPressure)))	Average chamber pressure in the deposition step.	DepCHPress
Mean(Tb4e4(DEP(Dome_heater_GSH)))	Average dome heater temperature measured near the dome surface during deposition.	DHgsh
Mean(Tb4e4(DEP(Dome_heater_Side)))	Average dome heater temperature measured on the side during deposition.	DHside
Mean(Tb4e4(DEP(Dome_heater_TOP)))	Average dome heater temperature measured on the top during deposition.	DHtop
Mean(Tb4e4(DEP(GasFlow_Ar_Side)))	Average Argon flow through the side nozzle during deposition.	ARside
Mean(Tb4e4(DEP(GasFlow_Ar_Top)))	Average Argon flow through the top nozzle during deposition.	ARtop
Mean(Tb4e4(DEP(GasFlow_SiH4_Top)))	Average Silan flow through the top nozzle during deposition.	SIH4top
Mean(Tb4e4(DEP(Inner_helium_leakRate)))	Average helium leak rate measured on the inside of the wafer during deposition.	InnerLeakRate

Table A.1: Part 1 of the tabular overview of the indicators (predictors) used in the arcing problem.

Indicator Name	Description	Abbrevation
Mean(Tb4e4(DEP(Outer_helium_leakRate)))	Average helium leak rate measured on the outside of the wafer during deposition.	OuterLeakRate
Mean(Tb4e4(DEP(RF_bias_forward)))	Average rf bias power during deposition.	DEPRFbias
Mean(Tb4e4(DEP(RF_bias_reflected)))	Average rf bias reflected power during deposition.	DEPRFbias-ref
Mean(Tb4e4(DEP(RF_side_forward)))	Average rf power from the side during deposition.	DEPRFside
Mean(Tb4e4(DEP(RF_side_reflected)))	Average rf reflected power from the side during deposition.	DEPRFside-ref
Mean(Tb4e4(DEP(RF_top_forward)))	Average rf power from the top during deposition.	DEPRFtop
Mean(Tb4e4(DEP(RF_top_reflected)))	Average rf reflected power from the top during deposition.	DEPRFtop-ref
Mean(Tb4e4(DEP(WaferTemp)))	Average wafer temperature during deposition.	DEPWaferTemp
Mean(WaferTemp)	Average wafer temperature during the process.	WaferTemp
Range(Tb4e4(DEP(Dome_heater_GSH)))	Range of the dome heater temperature measured near the dome surface during deposition.	RngDHgsh
Range(Tb4e4(DEP(Dome_heater_Side)))	Range of the dome heater temperature measured on the side during deposition.	RngDHside
Range(Tb4e4(DEP(Dome_heater_TOP)))	Range of the dome heater temperature measured on the top during deposition.	RngDHtop
Range(WaferTemp)	Range of the wafer temperature.	RngWaferTemp
StdDev(Tb4e4(DEP(GasFlow_Ar_Side)))	Standard deviation of the Argon flow from the side during deposition.	sdARside
StdDev(Tb4e4(DEP(GasFlow_Ar_Top)))	Standard deviation of the Argon flow from the top during deposition	sdARtop
StdDev(Tb4e4(DEP(GasFlow_SiH4_Top)))	Standard deviation of the Silan flow from the top during deposition.	sdSIH4top
StdDev(Tb4e4(DEP(RF_bias_forward)))	Standard deviation of the rf bias power during deposition.	sdDEPRFbias
StdDev(Tb4e4(DEP(RF_side_forward)))	Standard deviation of the rf power from the side during deposition.	sdDEPRFside
StdDev(Tb4e4(DEP(RF_top_forward)))	Standard deviation of the rf power from the top during deposition.	sdDEPRFtop

Table A.2: Part 2 of the tabular overview of the indicators (predictors) used in the arcing problem.

A.2 R summary of predictors

```
  DurDep          s16.18CHPress  s16.18RFside  s16.18RFside.ref
Min.   : 55.06   Min.   :1999   Min.   :1978   Min.   :13.22
1st Qu.: 89.13   1st Qu.:2101   1st Qu.:2002   1st Qu.:17.37
Median : 89.98   Median :2399   Median :2013   Median :25.90
Mean   : 90.27   Mean   :2494   Mean   :2065   Mean   :24.63
3rd Qu.: 91.02   3rd Qu.:2888   3rd Qu.:2136   3rd Qu.:30.38
Max.   :117.15   Max.   :3536   Max.   :2374   Max.   :38.83

s16.18RFtop    s16.18RFtop.ref   s15CHPress      s15RFbias
Min.   : 988   Min.   : 3.000   Min.   :2136   Min.   :1228
1st Qu.:1001   1st Qu.: 4.000   1st Qu.:2411   1st Qu.:1501
Median :1006   Median : 6.500   Median :4104   Median :1506
Mean   :1017   Mean   : 7.009   Mean   :3439   Mean   :1514
3rd Qu.:1027   3rd Qu.: 7.958   3rd Qu.:4183   3rd Qu.:1510
Max.   :1095   Max.   :22.667   Max.   :4586   Max.   :2018

s15RFbias.ref     s15RFside      s15RFside.ref      s15RFtop
Min.   : 1.167   Min.   :2899   Min.   : 0.000   Min.   :1253
1st Qu.: 1.757   1st Qu.:3096   1st Qu.: 1.500   1st Qu.:1302
Median : 2.438   Median :3109   Median : 2.571   Median :1306
Mean   : 5.628   Mean   :3104   Mean   : 7.488   Mean   :1305
3rd Qu.:10.000   3rd Qu.:3116   3rd Qu.:14.554   3rd Qu.:1309
Max.   :26.286   Max.   :3141   Max.   :19.667   Max.   :1324

 s15RFtop.ref    DepCHPress       DHgsh           DHside
Min.   :5.000   Min.   :5983   Min.   :128.7   Min.   :148.1
1st Qu.:5.714   1st Qu.:6034   1st Qu.:134.7   1st Qu.:154.6
Median :6.000   Median :6054   Median :135.8   Median :156.5
Mean   :6.116   Mean   :6055   Mean   :135.5   Mean   :156.1
3rd Qu.:6.571   3rd Qu.:6082   3rd Qu.:137.2   3rd Qu.:158.0
Max.   :7.727   Max.   :6185   Max.   :138.2   Max.   :159.8

    DHtop           ARside          ARtop          SIH4top
Min.   :127.0   Min.   :111.0   Min.   :16256   Min.   :12388
1st Qu.:128.4   1st Qu.:111.0   1st Qu.:16257   1st Qu.:12408
Median :129.1   Median :111.0   Median :16258   Median :12408
Mean   :129.3   Mean   :111.0   Mean   :16258   Mean   :12407
3rd Qu.:130.1   3rd Qu.:111.0   3rd Qu.:16260   3rd Qu.:12408
Max.   :134.1   Max.   :111.1   Max.   :16263   Max.   :12409


InnerLeakRate   OuterLeakRate      DEPRFbias    DEPRFbias.ref
Min.   :11.55   Min.   : 48.53   Min.   :3501   Min.   :10.24
1st Qu.:14.21   1st Qu.: 72.60   1st Qu.:3506   1st Qu.:52.86
Median :15.94   Median : 80.97   Median :3507   Median :54.58
Mean   :15.69   Mean   : 81.44   Mean   :3507   Mean   :44.93
```

```
3rd Qu.:16.87   3rd Qu.: 91.72   3rd Qu.:3509   3rd Qu.:56.13
Max.   :19.73   Max.   :127.86   Max.   :3514   Max.   :59.33

  DEPRFside    DEPRFside.ref       DEPRFtop     DEPRFtop.ref
Min.   :3100   Min.   : 5.920   Min.   :1301   Min.   :4.393
1st Qu.:3106   1st Qu.: 6.423   1st Qu.:1304   1st Qu.:4.837
Median :3107   Median : 6.577   Median :1306   Median :4.934
Mean   :3107   Mean   : 6.833   Mean   :1305   Mean   :4.915
3rd Qu.:3109   3rd Qu.: 6.747   3rd Qu.:1307   3rd Qu.:5.021
Max.   :3117   Max.   :11.381   Max.   :1310   Max.   :5.407

 DEPWaferTemp     WaferTemp        RngDHgsh       RngDHside
Min.   :468.4   Min.   :468.5   Min.   :2.000   Min.   : 3.000
1st Qu.:496.5   1st Qu.:494.0   1st Qu.:3.000   1st Qu.: 8.000
Median :498.9   Median :496.3   Median :4.000   Median : 9.000
Mean   :495.7   Mean   :493.8   Mean   :4.057   Mean   : 8.755
3rd Qu.:505.0   3rd Qu.:502.3   3rd Qu.:4.000   3rd Qu.: 9.000
Max.   :541.7   Max.   :541.4   Max.   :7.000   Max.   :13.000

   RngDHtop      RngWaferTemp      sdARside          sdARtop
Min.   : 4.00   Min.   : 0.00   Min.   :0.00000   Min.   : 0.000
1st Qu.:17.00   1st Qu.:15.00   1st Qu.:0.00000   1st Qu.: 5.269
Median :18.00   Median :15.00   Median :0.00000   Median : 7.716
Mean   :18.27   Mean   :13.32   Mean   :0.03625   Mean   : 7.448
3rd Qu.:20.00   3rd Qu.:16.00   3rd Qu.:0.00000   3rd Qu.:10.504
Max.   :21.00   Max.   :49.00   Max.   :0.25155   Max.   :13.338

  sdSIH4top        sdDEPRFbias     sdDEPRFside      sdDEPRFtop
Min.   :  0.000   Min.   :13.20   Min.   :11.68   Min.   : 6.626
1st Qu.:  0.000   1st Qu.:17.54   1st Qu.:13.35   1st Qu.: 9.684
Median :  1.421   Median :18.71   Median :14.80   Median :10.493
Mean   :  5.371   Mean   :20.24   Mean   :16.78   Mean   :10.455
3rd Qu.:  2.393   3rd Qu.:20.59   3rd Qu.:16.11   3rd Qu.:11.243
Max.   :143.567   Max.   :35.70   Max.   :32.15   Max.   :13.773
```

A.3 Correlation matrix

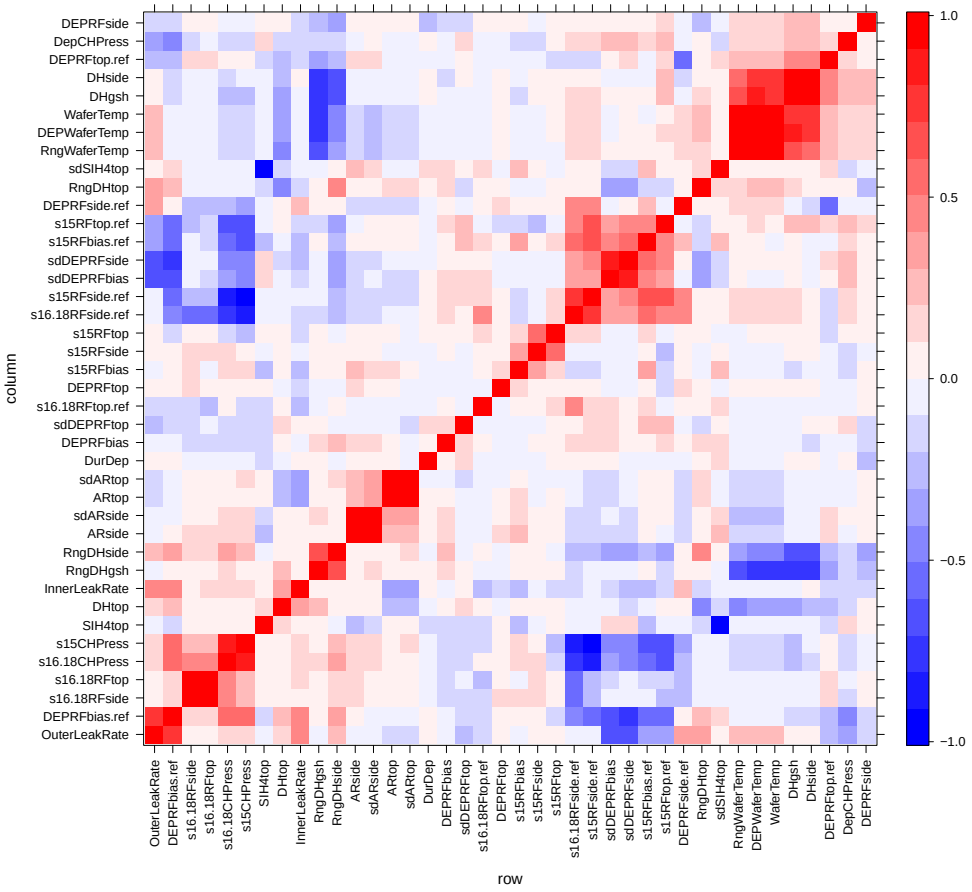

Figure A.1: Correlation matrix of all predictors of the arcing problem as `lattice` plot.

Appendix B

Modelling Film Thickness on a PVD Tool

For modelling the film thickness on a PVD tool, the values of the predictor variables come directly from the PVD tool.

B.1 Table of variables

Table B.1 gives an overview of the names of the indicators along with a description. Furthermore, all indicators have abbrevations by which they are referred to. These abbrevations serve as predictor names in the created models.

Indicator name	**Description**	**Abbrevation**
Maxtargetlife	Total lifetime of the PVD tool's aluminum target up to now.	targetlife
M.4Cold.cpctv_manometer.	Pressure in the process chamber in process step 4	m4manometer
M.4Cold.dc_curr.	Current of the plasma generator in step 4	m4dccurr
M.4Cold.dc_pwr_act.	Electrical power of the plasma generator in step 4	m4dcpwract
M.4Cold.iongauge_pres.	Pressure in certain chamber regions in step4	m4iongauge
M.4Cold.trgt_volt_sense.	Adjustment of the aluminum target in step 4	m4trgtsense
M.4Cold.wafer_temp.	Wafer temperature in step 4	m4wfrtemp
M.6Heat.cpctv_manometer.	Pressure in the process chamber in process step 6	m6manometer
M.6Heat.dc_curr.	Current of the plasma generator in step 6	m6dccurr
M.6Heat.dc_pwr_act.	Electrical power of the plasma generator in step 6	m6dcpwract
M.6Heat.wafer_temp.	Wafer temperature in step 6	m6wfrtemp
M.9Hot.cpctv_manometer.	Pressure in the process chamber in process step 9	m9manometer
M.9Hot.dc_curr.	Current of the plasma generator in step 9	m9dccurr
M.9Hot.dc_pwr_act.	Electrical power of the plasma generator in step 9	m9dcpwract
M.9Hot.iongauge_pres.	Pressure in certain chamber regions in step 9	m9iongauge
M.9Hot.wafer_temp.	Wafer temperature in step 9	m9wfrtemp
Time.count.dc_curr	Time elapsed	timecnt

Table B.1: Tabular overview of the 17 indicators (predictors) used for modelling film thickness.

B.2 R summary of predictors

```
  targetlife      m4manometer      m4dccurr       m4dcpwract
Min.   :   0.0   Min.   :2946   Min.   :19.95   Min.   :10975
1st Qu.:  89.0   1st Qu.:2985   1st Qu.:21.00   1st Qu.:11429
Median : 410.0   Median :2996   Median :22.06   Median :11556
Mean   : 471.6   Mean   :2995   Mean   :22.28   Mean   :11464
3rd Qu.: 804.0   3rd Qu.:3005   3rd Qu.:23.33   3rd Qu.:11572
Max.   :1110.0   Max.   :3051   Max.   :25.00   Max.   :11573

  m4iongauge      m4trgtsense      m4wfrtemp      m6manometer
Min.   :  1.00   Min.   :453.0   Min.   :286.0   Min.   :2058
1st Qu.: 13.35   1st Qu.:482.0   1st Qu.:293.8   1st Qu.:2104
Median : 32.71   Median :503.2   Median :294.8   Median :2115
Mean   : 53.18   Mean   :502.5   Mean   :295.3   Mean   :2113
3rd Qu.: 90.57   3rd Qu.:526.5   3rd Qu.:295.2   3rd Qu.:2125
Max.   :172.91   Max.   :561.1   Max.   :323.9   Max.   :2152

   m6dccurr       m6dcpwract      m6wfrtemp      m9manometer
Min.   :12.73   Min.   :24.31   Min.   :286.4   Min.   :2137
1st Qu.:14.77   1st Qu.:24.93   1st Qu.:293.8   1st Qu.:2164
Median :16.84   Median :26.00   Median :294.0   Median :2171
Mean   :18.48   Mean   :28.56   Mean   :294.9   Mean   :2173
3rd Qu.:22.23   3rd Qu.:32.13   3rd Qu.:295.0   3rd Qu.:2181
Max.   :25.29   Max.   :32.77   Max.   :322.5   Max.   :2209

   m9dccurr       m9dcpwract      m9iongauge       m9wfrtemp
Min.   :18.86   Min.   :11070   Min.   : 1.000   Min.   :286.6
1st Qu.:20.00   1st Qu.:11502   1st Qu.: 9.417   1st Qu.:293.0
Median :21.00   Median :11555   Median :21.143   Median :294.1
Mean   :21.21   Mean   :11501   Mean   :33.096   Mean   :294.5
3rd Qu.:22.00   3rd Qu.:11572   3rd Qu.:59.727   3rd Qu.:295.2
Max.   :24.00   Max.   :11573   Max.   :95.320   Max.   :320.2

   timestmp
Min.   : 1.599
1st Qu.:20.222
Median :21.269
Mean   :21.409
3rd Qu.:22.985
Max.   :29.695
```

B.3 Correlation matrix

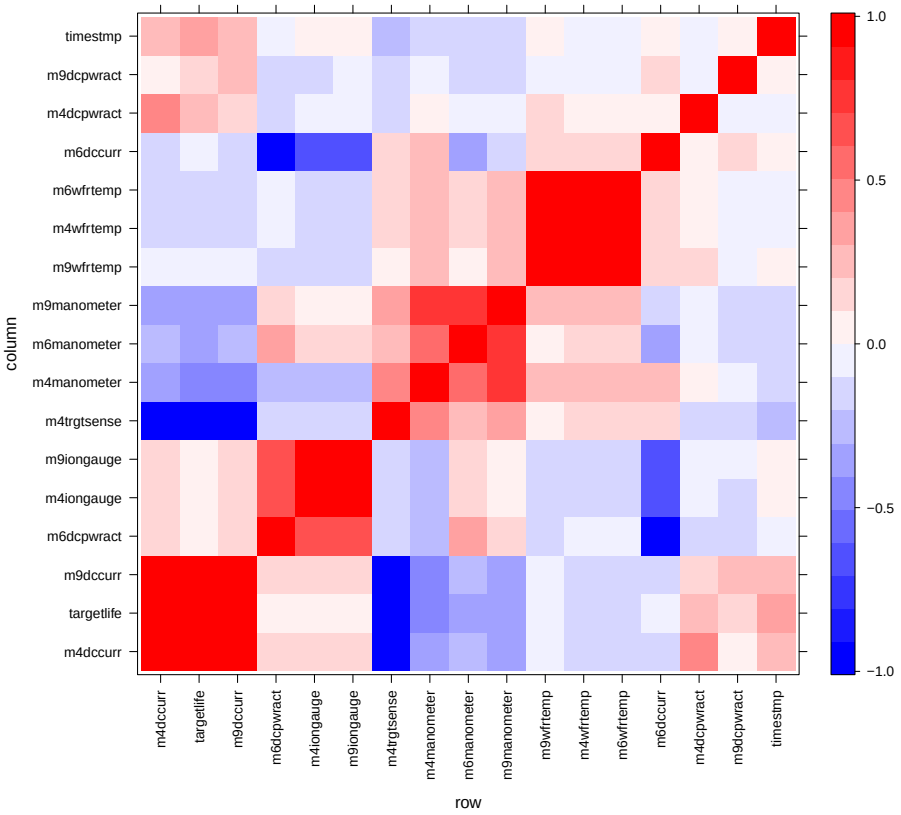

Figure B.1: Correlation matrix of all predictors of film thickness modelling as `lattice` plot.

Appendix C

The Caret R package

The `caret R` package introduced by Max Kuhn (see [16] Kuhn, 2008) offers a simple framework for several different classification and regression methods. The focus of the package lies on model training and tuning using numerous statistical techniques. It also contains methods for pre-processing training data, assessing variable importance and model visualizations.

The package serves as a wrapper for the different original packages that contain methods for modelling. We use the `caret` functionality as a wrapper for the `gbm` package to carry out stochastic gradient boosting.

C.1 The train command

The main command of the package is `train`. It allows to access the functions of the `gbm` package or any other package available and creates a corresponding model object.

The following creates a stochastic gradient boosting classification model for our `frogs` data example from the `DAAG` library.

```
> xtrain<-data.frame(altitude, distance, NoOfPools, NoOfSites,
  avrain, meanmin, meanmax)
> pres.abs<-as.factor(pres.abs)
> library(caret)
> mod<-train(pres.abs~., data=xtrain, method="gbm")
```

This performs stochastic gradient boosting as implemented in the `gbm` package with the binary response variable `pres.abs` and a predictor data frame `xtrain` with 7 predictors. The resulting `gbm` object can be accessed via `mod$finalModel`.

For tuning the model and finding the best model parameter constellation, the `caret` package allows the user to create a grid of parameter values and calculates the results over all the values of the grid. With

```
> gbmGrid<-expand.grid(.interaction.depth=(1:4),
  .n.trees=c(100, 500), shrinkage=0.075)
```

a grid of different parameter values is stored in `gbmGrid`. In this example, the interaction depth of the boosting algorithm runs from 1 to 4, the number of iterations is either 100 or 500 and the shrinkage rate is chosen to be 0.075. Thus, a grid with 8 parameter combinations results. The `train` command then finds the parameter combination that yields the best model quality value, e.g. classification accuracy or Cohen's Kappa for classification or R^2 for regression.

These values can be calculated using resampling methods such as bootstrapping or cross validation. For example, the control command

```
> modControl<-trainControl(method="cv", number=10)
```

can be used to evaluate a model using 10-fold cross validation.

For our `frogs` example the command

```
> mod<-train(pres.abs~., data=xtrain, method="gbm",
  tuneGrid=gbmGrid, trControl=modControl, metric="Accuracy")
```

yields the following output.

```
> mod

Call:
train.formula(form = pres.abs ~ ., data = xtrain, method = "gbm",
    tuneGrid = gbmGrid, trControl = modControl, metric = "Accuracy")

212 samples
7 predictors

summary of cross-validation (10 fold) sample sizes:
    191, 192, 190, 190, 190, 191, ...

cv resampled training results across tuning parameters:

  interaction.depth  n.trees  shrinkage  Accuracy  Kappa  Accuracy SD  Kappa SD  Selected
  1                  100      0.075      0.782     0.508  0.0891       0.229     *
  1                  500      0.075      0.735     0.42   0.0635       0.151
  2                  100      0.075      0.74      0.436  0.0488       0.093
  2                  500      0.075      0.721     0.395  0.0681       0.142
  3                  100      0.075      0.74      0.433  0.0702       0.153
  3                  500      0.075      0.702     0.354  0.0664       0.121
  4                  100      0.075      0.778     0.513  0.0512       0.116
  4                  500      0.075      0.712     0.379  0.0531       0.0981

Accuracy was used to select the optimal model using the largest value.

The final values used in the model were interaction.depth = 1, n.trees = 100 and shrinkage = 0.075.
```

The best model is chosen by means of classification accuracy (`metric`).

C.2 Other useful commands

For classification problems, the function `confusionMatrix` can be used to summarize the results:

```
 > confusionMatrix(predict(mod, xtrain, type="raw"), pres.abs)

Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 119  21
         1  14  58

               Accuracy : 0.8349
                 95% CI : (0.7779, 0.8822)
  No Information Rate : 0.6274
  P-Value [Acc > NIR] : 2.775e-11

                  Kappa : 0.6404

            Sensitivity : 0.8947
            Specificity : 0.7342
         Pos Pred Value : 0.8500
         Neg Pred Value : 0.8056
             Prevalence : 0.6274
         Detection Rate : 0.5613
 Detection Prevalence : 0.6604

     'Positive' Class : 0
```

The output shows the actual confusion table (transposed) along with an accuracy measure (`Accuracy`, number of correctly assigned divided by all observations), Cohen's kappa (`Kappa`), sensitivity and specificity values and other statistical measures.

The generic function `varImp` can be used to analyze the effect of the predictors on the model. It gives a variable importance ranking of all predictors used scaled between 0 and 100 where 100 is the importance of the most important predictor. The overview is based on the variable importance calculation method of the underlying model type. Furthermore, the command `varimpplot` yields a graphical output of the variable ranking.

For a detailed description of various other useful functions in the `caret` package see [16] Kuhn (2008).

Bibliography

[1] Altman D.G., Bland J.M. Diagnostic tests 1: sensitivity and specificity. *British Medical Journal*, 308:1552–1552, 1994.

[2] Berk R.A. *Statistical Learning from a Regression Perspective*. Springer, New York, 2008.

[3] Breiman L. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[4] Breiman L. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[5] Breiman L., Cutler A. Random forests. Berkley, http://www.stat.berkeley.edu/ breiman/RandomForests/, 2005.

[6] Breiman L., Friedman J.H., Olshen R.A., Stone C.J. *Classification and Regression Trees*. Wadsworth, California, 1984.

[7] Cohen J. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20:37–46, 1960.

[8] Elshabini-Riad A., Barlow F.D. *Thin Film Technology Handbook*. McGraw-Hill, 1998.

[9] Freund Y., Schapire R. A decision-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory*, pages 23–37. Springer Berlin / Heidelberg, 1995.

[10] Friedman J.H. Greedy function approximation: a gradient boosting machine. *The Annals of Statistics*, 29:1189–1232, 2001.

[11] Friedman J.H. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38:367–378, 2002.

[12] Genuer R., Poggi J.-M., Tuleau C. Random Forests: some methodological insights. INRIA, http://hal.inria.fr/inria-00340725/en/, 2008.

[13] Hastie T., Tibshirani R., Friedman J. *The Elements of Statistical Learning*. Springer, New York, 2001.

[14] Hothorn T., Hornik K., Zeileis A. Unbiased recursive partitioning: a conditional inference framework. *Journal of Computational and Graphical Statistics*, 15(3):651–674, 2006.

[15] Kriegler B. *Cost-Sensitive Stochastic Gradient Boosting Within a Quantitative Regression Framework*. Dissertation, Department of Statistics, UCLA, 2007.

[16] Kuhn M. Building predictive models in R using the caret package. *Journal of Statistical Software*, 28(5):1–26, 2008.

[17] Landis J.R., Koch G.G. The measurement of observer agreement for categorical data. *Biometrics*, 33:159–174, 1977.

[18] Liaw A., Wiener M. Classification and Regression by randomForest. Technical report, R News, 2002.

[19] Loh W.-Y. Regression trees with unbiased variable selection and interaction detection. *Statistica Sinica*, 12:361–386, 2002.

[20] Loh W.-Y., Shih Y.-S. Split selection methods for classification trees. *Statistica Sinica*, 7:815–840, 1997.

[21] Quinlan, J.R. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[22] Ridgeway G. *Generalized boosted models: a guide to the gbm package*, 2007.

[23] Sandri M., Zuccolotto P. Variable Selection Using Random Forests. In *Data Analysis, Classification and the Forward Search*, pages 263–270. Springer Berlin Heidelberg, 2006.

[24] Schapire R.E. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[25] Schapire R.E. A brief introduction to boosting. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1401–1406, 1999.

[26] Shih Y.-S. Families of splitting criteria for classification trees. *Statistics and Computing*, 9:309–315, 1999.

[27] Siroky D.S. Navigating random forests and related advances in algorithmic modeling. *Statistics Surveys*, 3:147–163, 2009.

[28] Strobl C., Boulesteix A.-L., Augustin T. Unbiased split selection for classification trees based on the Gini Index. Technical report, 2006.

[29] Strobl C., Boulesteix A.-L., Zeileis A., Hothorn T. Bias in random forest variable importance measures: illustrations, sources and a solution. *BMC Bioinformatics*, 8(25), 2007.

[30] Su X.G. Maximum Likelihood Regression Trees. In *2002 Proceedings of the American Statistical Association*, pages 3379–3383, 2002.

[31] Therneau T.M., Atkinson E.J. An Introduction to Recursive Partitioning Using the RPART Routines. Technical Report 61, Department of Health Science Research, Mayo Clinic, Rochester, 1997.

[32] Wolf S. *Microchip Manufacturing*. Lattice Press, 2003.

[33] Zeileis A., Hothorn T., Hornik K. Model-based recursive partitioning. *Journal of Computational and Graphical Statistics*, 17(2):492–514, 2008.