Sapienza
Università di Roma

# CART and Random Forest for claim watching and individual claims reserving

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Corso di Laurea Magistrale in Data Science

Candidate
Nello Castaldo
ID number 1794724

Thesis Advisors
Prof. Franco Moriconi
Prof. Ioannis Chatzigiannakis

Academic Year 2021/2022

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: castaldo.1794724@studenti.uniroma1.it

# Contents

# Chapter 1

# Introduction

Claim watching is the insurer's activity of monitoring the cost development of a single claim. This procedure includes different activities that involve both categorical variables and non-categorical variables. In particular, the non-categorical variables are more concerned with the payments, and the activity of predicting these variables is known as *Individual claim reserving.* The categorical variables are related to the claim status. For example they are used to model if a claim is closed, if it will involve a lawyer, what kind of payment it needs, etc. Obviously we can observe the single claim history until the present day, but an insurance company would like to know what will be the development path of that claim in the next years. That's why different methods of prediction have been proposed. Some of them involve machine learning. In particular this thesis is based on the methods proposed originally by Mario Wuthrich [9] and then developed in the article "Claim watching and individual claims reserving using classification and regression trees" [3]. The aim of this thesis is to start from the basis described in this paper and to optimize the CART techniques and then compare them with a bootstrapped method such as Random Forest. In practise we apply these algorithms to the Italian Motor insurance line, including CARD and NoCARD procedures.
In this thesis, the main objective is to highlight the comparison between CART and Random Forest, in terms of overall performance and, most of all, in term of stability in both aggregate and, in particular, individual claims analysis. The structure of the chapters has been chosen to allow the reader to understand the theory behind

the practical implementation of the model and the reasons of the choices made during the development of the software. The first chapter regards the explanation of the model proposed in the paper [3]. The second chapter introduces the theory of CART and Random Forest, with a particular attention to the R implementation of the algorithms, to show pros and cons of both the models, and their usability in the motor insurance field. The third chapter explains practically the structure of the code and the improvements made during the period of work. In particular the differences and the problems related to CART and Random Forest will be shown, plus a brief overview of two likelihoods methods, that are still in the initial phase of development. Finally, in the last chapter it is possible to see the results of the methods that have been previously explained and some considerations about them. It is important to stress that in all the thesis, for privacy reasons, data origin will be never mentioned, such as the real values of the variables. They have been masked to be unrecognizable, but their meaning is still consistent and meaningful, as it was originally.

## 1.1 Model Description

The aim of this chapter is to explain the theoretical idea behind the methods proposed, without going into the details of the class of algorithms used. In fact the model proposed can be implemented with different algorithms. Here the focus is to highlight the concepts that will help the reader to understand the practical part discussed in chapter 3.
In a non-life environment, we are interested in the individual claim development in the years (or in the unit of time used by the company). In particular this means that we want to know different information about the single claim. The typical claim watching problem can be expressed in the following form:

$$\mathbb{E}[\mathrm{Y}^C_{t+\tau}|F_t] = \mu(x^C_t),\, \tau > 0$$

where:

- $Y^C_{t+\tau}$ is the response variable
- $x^C_t$ is the vector of the *covariates*
- $F_t$ is the information available at time $t$
- $\mu$ is the predictor function, estimated from the data

These information are typically related to each other and they can be represented as quantitative or qualitative variables. For example questions about the claim status or the type of damage related to the claim need a qualitative response. Questions about the payments need a quantitative response.
In the software that we developed we address these problems in particular:

- Claim status (open or closed)
- Type of payment (CARD or NoCARD)
- Type of damage (Objects, people, both)
- CARD payment
- NoCARD payment

- CARD reserve
- NoCARD reserve

As we can notice, the first three elements of the list require a categorical response, while the others require a quantitative response. In machine learning these are two different class of problems. The first is known as *Classification*, the second is called *Regression*. In general, in the first case we have some observations and we want to assign a label to each of them by choosing one from a well specified set of classes. In the second case, we have again observations, but now we want to assign an estimated value that is the result of some kind of aggregation function. Obviously these problems cannot be solved by the same model. In fact there are models for *classification* and models for *regression*. We will see in the next chapter how the same idea of model needs to be modified in order to address these two different types of problem in the case of *Trees*. That's why the model proposed by Prof. De Felice and Prof. Moriconi is composed by two parts: *Frequency* and *Severity*. The *Frequency* part is composed by the *classification* model. It is used for the prediction of the event occurrences. The *Severity* part is used to solve the *regression* problems related to the paid amounts or the case reserves. It is also called *conditional severity*, because its predictions are conditioned on the responses of the *Frequency* model. This means, for example, that we estimate the amount of payment of type CARD, only if this type of payment has been predicted by the *frequency* model. The predictions that we obtain are called *one-period prediction*, because given the information available at time $t$, the model is able to predict only the responses at time $t+1$. However we will see that it can be used also for *multi-period predictions*. Before going into further details, we need to have a basic knowledge of the *covariates* used and the general notation.

First of all we have to specify that the *covariates* can be divided into two classes:

- Static
- Dynamic

The static variables are fixed for all the life of the claims. They are usually used to identify the claims and some of its fixed features. The dynamic variables can change during the life of the claims, depending on the events that can happen. They are very useful to keep track of the history of a single claim. In fact they are time dependent, so we have to store the value of the same variable in different moments because it can vary, and it contains important information that have to be used for the predictions. A possible drawback of this variation is that the feature vector increases as time passes. That means that the model has to be able to handle different input dimensions, and we need a large amount of space to store all these data. In our particular study case we have these fixed variables for each claim:

- *Accident year*

  The year in which the claim has happened. It is usually indicated with the letter $i$, where $i = 1, ..., I$.

- *Reporting delay*

  Claims can be reported with delay with respect to the accident year. This delay is indicated with the letter $j$ with $j = 0, 1..., J$. The reporting date of a claim with accident year $i$ and reporting delay $j$ is $i + j$.

- *Claims identification (cc)*

  It is a code used to identify each specific claim.

The dynamic variables are instead:

- *CaseRCA/CaseRNC*

  These are the amounts that the company reserves for the claims with CARD and NoCARD payments. They are quantitative variables.

- *L*

  This is a categorical variable that can assume only two values: 0 and 1. 1 means that the claims is involved in a lawsuit, 0 the opposite.

- *Z*

  Categorical variable that indicates if a claim is opened (0), or closed (1). A closed claim can be opened again.

- *PagcumCA/PagcumNC*

  Sum of the CARD and NoCARD payments in the years for a single claim.

- *RCA/RNC*

  Categorical variable that indicates if claims require a CARD or NoCARD case reserve. In this case the value is 1, otherwise is 0. A claim is allowed to have both types of payment.

- *TipoDnMax*

  Categorical variable used to identify the type of damage of a claim. It can be 0 if there is no damage, 1 if it involves only material things, 2 if it involves people, 3 if it involves both.

- *TipoDnRis*

  Categorical variable that specifies the type of damage that needs a reserve.

- *YCA/YNC*

  Categorical variable that indicates if a payment has been done in the observation year. It can be 0 if there is no payment, 1 otherwise.

So,in general, the matrix of *covariates* has a dynamic number of components, with a specific structure that can be represented using column vectors that are related to the static variables and to the dynamic ones for each observation year. We can use the following notation to represent this idea:

$$x^{(\nu)}_{i,j|k} = (A^{(\nu)}_{i,j}, B^{(\nu)}_{i,j|0}, ...., B^{(\nu)}_{i,j|k})$$

where $A^{(\nu)}_{i,j}$ is the vector of static variables, $B^{(\nu)}_{i,j|h}$ ,with $h = 0, .., k$, is the vector of dynamic variables observed in year $i + j + h$. The dimension of the $B$ vectors increases when $h$ increases. In fact the variables that has been predicted for the current year become *covariates* for the subsequent year. This is called *dynamic modeling.* This is the key idea that allows this model to be used for *multi-period predictions.*

Before going into the details of the train and test sets, we have to introduce a fundamental variable used to define the models. It is the *time-lag l.* A generic random variable $X^{(\nu)}_{i,j|k}$, observed at time $t = i + j + k$, has $l = j + k = t - i$ This variable allows us to redefine the prediction problem as:

$$\mathbb{E}[\mathrm{Y}^{(\nu)}_{i,j|k+1}|F_{i+j+k}] = \mu_l(x^{(\nu)}_{i,j|k})$$

The prediction function depends only on the *time-lag l.* This means that it can be applied to all the features that have the same *time lag.*
The following organization of data is a direct consequence of the previous statement. In fact, both for *Frequency* and *Severity* we can define the set of *lag observations* as:

$$D_l = D^C_l \cup D^P_l$$

with:

- $D^C_l = \{(x^{(\nu)}_{i,j|l-j}, Y^{(\nu)}_{i,j|l-j+1}); 1 \leq i \leq I - l - 1, j_1 \leq j \leq l, 1 \leq \nu \leq N_{i,j}\}$ : *calibration set*
- $D^P_l = \{(x^{(\nu)}_{I-l,j|l-j}, \cdot); j_{I-l} \leq j \leq l, 1 \leq \nu \leq N_{I-l,j}\}$ : *prediction set*

We can see from this definition, that the number of claims that we take into account for each calibration is directly dependent on the *lag l.* The organization of claims by *lag* shows a triangular structure, that is typical in claim reserving problems.
To fully understand the model and its predictions, the last thing we have to explain is the form of the *response variables*, in particular for the *frequency* model. In fact $Y^{(\nu)}_{i,j|k}$ is a multi-event response. This means that multiple variables have to be filled each time we get a prediction. So, the model should be able to associate multiple output to the same input. Hence, the response variable should take the following form:

$$F^{(\nu)}_{i,j|k} = ({}_hF^{(\nu)}_{i,j|k'}\ , h = 1, .., d) \text{ with } {}_hF^{(\nu)}_{i,j|k'} \in \{0, 1\}, h = 1, ..., d$$

This is a vector of categorical variables, that should be filled by the algorithm. But this is not supported by the R package *rpart*, so it is necessary to define an equivalent one-dimensional response. This response is formulated as:

$$W^{(\nu)}_{i,j|k+1} = \sum 2^{h-1}{}_hF^{(\nu)}_{i,j|k+1} \in \{0, ..., 2^d - 1\}$$

and we can define the conditional distribution of $W^{(\nu)}_{i,j|k+1}$ as:

$$W^{(\nu)}_{i,j|k+1}|F_{i+j+k} \sim Categorical(p^{(w)}_{j+k}(x^{(\nu)}_{i,j|k}))$$

where $p^{(w)}_{j+k}$ is a probability function. In our specific case we have 16 possible responses. Each of them corresponds to a specific combination of the output. It is possible to decompose the one-dimensional response in order to obtain all the single responses we are interested in. This is what we do in the software.
For the *multi-period predictions* we follow the simulation approach. Thanks to the objects calibrated on the *calibration sets* we are able to develop the path of each claim, applying subsequently models corresponding to different *lags.* As previously mentioned, the key to simulate the claim path is the *dynamic modelling.* This means that the *one-year* responses become the covariates for the subsequent years. The number of years that can be predicted by the simulation is limited by the maximum number of *lag* that we can observe in our initial data, that is $I - 2$ We will see, in chapter 3, how each variable is estimated with simulation in practise, and how many of them are used in the motor insurance case.
Finally we have to say that the claims we talked about until now are called *RBNS*, that stands for *Reported But Not Settled.* These are the claims that have been reported and that have a development over the years. These are only one portion of the final estimates. In fact there is also another category of claims that is called *IBNYR.* These are the *Incurred But Not Yet Reported.* These are the claims that happened during a year but that have not been reported yet, because of the reporting delay. These claims have to be taken into account in order to provide correct estimates for aggregate claim reserves. In the software we provide methods to calculate these claims, but they go beyond the scope of this thesis.

# Chapter 2

# Regression trees and Random Forest

In this chapter we will see some basic theory elements behind CART and Random Forest, to better understand how they work, their power and their weaknesses. In both cases we are in the field of *supervised learning*. That means that we need samples with associated correct labels or quantity of interest to train them. The distinction depends on the type of task we want to perform. In fact these methods are suitable for *classification* and *regression* tasks. In our specific case we use CART and Random Forest both for *classification* (*frequency* section) and for *regression* (*severity* section). We will see the properties of CART and the reason why they are a good algorithm for *claim watching* and the cases in which Random Forest can outperform CART, improving the performance of the general model.
As we saw in the previous chapter, we need to find a suitable function, that is able to fit the training data and to make prediction on new observed data, with a small error. In our specific case we have a lot of functions that have to be found, that are related to the *frequency* and *severity* parts of each *lag*. In machine learning literature, basic algorithms are commonly classified into two macro classes: *parametric* and *non-parametric*. The *parametric* algorithms return a prediction function by fitting a set of parameters that minimize some error function on the input dataset. Possible example could be *GLMs*. The *non-parametric* algorithms have not a fixed structural form for the prediction function, and they fit it to the dataset without any set

of parameters. Parameters are usually associated with inputs covariates, so the dimension of the parameters array is dependent on the input features. The lack of these parameters makes the model more elastic. This is a very helpful property in the case of the model we defined in chapter 1, because the number of input features increases as *lags* increase. CART and Random Forest belong to this second class of methods.

## 2.1 Classification And Regression Trees (CART)

The basic idea behind CART is to divide the feature space into regions, that contain elements of the dataset that are similar among each other and that can be efficiently represented by some sort of function, that could be, for example, the sample mean. So when we use tree based models, we end up having the feature space divided into a set of rectangles. More formally we can say that $X$ (feature space), is divided into a disjoint subset $(X_t)_{t \in T}$, where $T$ is a finite set of indexes. The tree that is able to perform this division is a *binary tree.* For each parent node we have two children nodes. The nodes that have not children are called *leaves.* The number of *leaves* corresponds to the number of region $(X_t)_{t \in T}$, where $T$ now becomes the index of the corresponding *leaf.* So when we want to assign a sample to a specific region, we simply have to look at the *leaf* the sample belongs to. So, each sample follows a specific path. This path depends on the nodes of the tree. In fact each node corresponds to a specific condition. These conditions are boolean questions that regard a single feature of the sample features vector. The feature can be both categorical or non-categorical. So we need an algorithm that is able to split the feature space in a sufficient number of regions and that can also choose what are the most significant features to split on. How does it know the optimal depth of the tree and the optimal choice of the conditions?
It needs a *goodness of split* criterion. What we said until now is valid both to *regression trees* and *decision trees.* The difference in the grow algorithm is the criterion we choose. Both for *regression* and *classification* we have different criteria, that in R are implemented in the *rpart* function. They can be chosen thanks to the parameter *method.* In the *classification* case the *rpart* routines use the *impurity*

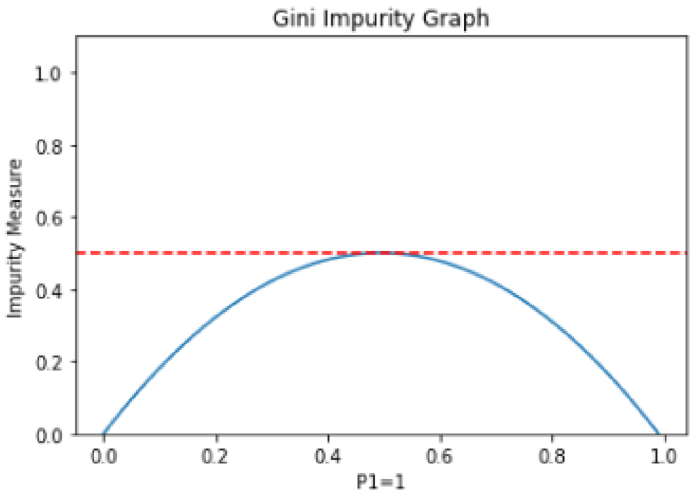


**Figure 2.1.** Gini impurity function

measures. Formally we can represent them as a general function:

$$I(A) = \sum_{i=1}^{C} f(p_{iA})$$

where $C$ is the number of classes, $p_{iA}$ is the proportion of elements of class $i$ in the node $A$. In our code, the function used to calculate the the *impurity* is the *Gini index*:

$$f(p) = p(1-p)$$

It is a concave function, and $f(1) = f(0) = 0$. We can observe it in the figure 2.1.

Intuitively a leaf is called *pure* when all its elements belong to the same class. In this case the *impurity* value is 0. The worst possible case is when the classes have a 50-50 splitting. There the function value is 0.5. Gini index is differentiable, that makes it more feasible for numerical optimization. Moreover it is more sensitive to changes in the node probabilities. In fact it is able to choose correctly the best splitting between two, even if they produce the same misclassification rate. For example we can consider a two-class problem with 400 samples for each class. Suppose we have to evaluate two splits, one with (300,100) on the first node and (100,300), and one with (200,400) and (200,0). Even if the misclassification rate is 0.25 for both, the second split produces a pure node, and it is probably better. Gini index is able to assign a better value to the second split. How this index is actually used? Basically we start from a root node and we try to find the best split that maximize the impurity reduction. In order to do this, the algorithm iterates over all the features of the feature vector to choose the best to split on, and the value of splitting in case it is a non-categorical variable. So it produces two new half-planes defined as

$$H_1(j,s) = \{X|X_j \leq s\} \text{ and } H_2(j,s) = \{X|X_j > s\}$$

Each of this half-planes becomes a new root node and the same procedure is applied. This is a recursive and greedy algorithm. The tree dimension is fundamental in order to make good predictions. In fact a small tree could not be able to capture possible differences among data, and, on the contrary, a very big tree could result in overfitting data. Both cases produce bad performances with new observed data. So we have to choose a good stopping criterion that avoids these problems. In the R implementation of the algorithm, we have the possibility to control the stopping criteria by choosing the value of some parameters that the function *rpart* takes as input. In particular, the list of control parameters is passed as argument of the *control* parameter of the function. Below we can see some examples:

- *minsplit*

  The minimum number of samples in a node that are required by the routine to try to perform a split on the node. The default value is 20.

- *minbucket*

  The minimum number of observation in a *leaf*. The default value is *minsplit*/3

- *maxdepth*

  The maximum number of internal nodes in the tree.

- *cp*

  The threshold complexity parameter.

*cp* is a regularization parameter that penalizes the tree according to the number of splits and the risk it has. The complete formula is

$$R_{cp}(T) = R(T) + cp * |T| * R(T_1)$$

where $R_{cp}$ is the total risk, $R(T)$ is the risk associated to the current tree, $T_1$ is the tree without splits and $|T|$ is the number of splits in the tree. At each new split, the R function associate a value for $R_{cp}$. We can follow this numerical random example to visualize it.

| | CP | nsplit | rel error | xerror | xstd |
|---|---|---|---|---|---|
| 1 | 0.066092 | 0 | 1.00000 | 0.50000 | 0.046311 |
| 2 | 0.040230 | 2 | 0.86782 | 0.73563 | 0.065081 |
| 3 | 0.034483 | 4 | 0.78736 | 0.91379 | 0.075656 |
| 4 | 0.022989 | 5 | 0.75287 | 1.01149 | 0.080396 |
| 5 | 0.019157 | 7 | 0.70690 | 1.17241 | 0.086526 |
| 6 | 0.011494 | 10 | 0.64943 | 1.21264 | 0.087764 |
| 7 | 0.010000 | 12 | 0.62644 | 1.31609 | 0.090890 |

As we can see, as the number of splits increase, the CP value decreases. When the CP value is equal to the *cp* threshold parameter, the algorithm stops. This means that a new split could not improve the error of at least *cp*. So if we assign 1 to the *cp* parameter, the algorithm will produce a tree with no splits. On the contrary, small values of *cp* correspond to large trees. In the previous example the value of *cp* is $cp = 0.01$. When we call the *rpart* function for a *classification* task, the parameter *method* allows us to specify all these options. In fact when we set it to the "*class*" value, it chooses by default the Gini index as splitting criterion. The resulting tree has a set of *leaves* that contains a subset of the original samples. The algorithm associate to each *leaf* a probability distribution over all the classes and the most probable class as predicted label. At prediction time we can choose if we want the algorithm to return the probability distribution or simply the predicted label. We can do this thanks to the *predict* function implemented in R, that allows to choose the type of output thanks to the parameter *type*. In particular with the value "*prob*", it returns a vector with the probability distribution associated to each new sample. This is fundamental because allows us to simulate the development of a claim in the software, and consequentially, to take into account some sort of variability.

In the *regression* case, the algorithm follows the same steps described above. However there are some differences that are necessary to adapt it to the *regression* task. These differences are implemented in the *rpart* function when we specify "*anova*" as value for the parameter *method*. "*anova*" implements a different splitting criterion, that maximize the improvement among root and children in term of residual sum of squares. More formally the quantity to be maximized is:

$$SS_T - (SS_L + SS_R) \text{ where } SS_T = \sum (y_i - \overline{\mathrm{y}})^2$$

The summary statistic that is now used to describe the node is the sample mean and the error is represented by the variance of $y$. The prediction error for new observations is instead $(y_{new} - \overline{\mathrm{y}})$.
In both *regression* and *classification*, there is still another procedure that is used to reduce the complexity of the model and to avoid overfitting. This procedure is called *pruning*. Intuitively, when a tree is pruned, only a subtree of the original tree is kept. With this method, we delete all the splits without a sufficient improvement in the performance that justifies them. In R, this procedure is implemented in the function *prune*, that takes as input two parameters: the tree to be pruned and the cost complexity parameter (*cp*). This parameter is not the same we passed as input to *rpart*. We can call it $\alpha$, and it represent the cost of adding another node to the tree. $\alpha$ is used as regularization parameter when we calculate the risk of a tree. The objective is to find $\alpha$ such as the associated subtree has the minimum risk. The problem is that we don't know $\alpha$ a priori, so we need some methods to find it. In the software we implement three different methods:

- *1-SD*
  In this case we pick a threshold by summing the minimum *xerror* (the cross validation error on the training set), and the *xstd* associated to that error (its standard deviation). Then we choose $\alpha$ by looking at the *CP* value of the first split with the error below the threshold.

- *Cross Validation*
  With this method we simply take the *CP* of the split with the minimum cross validation error.

- *Elbow method*

  When we look at the sequence of the errors, we can often observe that it drops at a certain point and then it stays flat. With this method we choose the *CP* that makes the error steeply drop.

All the information we need to choose the best $\alpha$ are contained in the *rpart* object, because they are automatically calculated.
At this point we have all the notions that are necessary to understand CART and the code explanation of the chapter 3. The last things we have to talk about are pros and cons of CART. In fact we said that they are a very powerful model, that are able to adapt to inputs of different dimensions, thanks to their non-parametric nature. Results they provide are interpretable and easy to visualize, both in case of *regression* and *classification*. They are able to handle both categorical and non-categorical covariates, even without normalization, and in a context with a lot of features, they are able to identify the most relevant in order to create partitions. So why do we want to upgrade our software with Random Forest? The answer is in one of the most critical aspects of CART: instability. In fact CART are largely affected by the data we use in the training set. This means that even a small change in the data could result in a big change of the tree structure. In claim watching and in individual claims reserving this could be a serious problem, because we register claims with very different developments and we don't want that the minority of particular claims could affect the predictions of all the other claims.

## 2.2 Random Forest

Random forest algorithm was introduced by Breiman [1]. It is an *ensemble* method. It is based on the combination of multiple predictors (trees), to improve the general performance of the model. The idea is to use multiple trees and then use the majority of votes to get the final prediction in the *classification* case, or to average all the results in *regression* case. This is not the only improvement in Random Forest. In fact, if we build many trees on the same dataset, we would obtain the same results for all of them. So, to differentiate trees and to reduce the correlation among them, two randomization methods have been introduced. The first method regards the dataset choice. From the original dataset, for each tree in the forest, we draw *bootstrap* samples. On these samples we build the tree and we take its vote. *Bootstrap* sampling consists in drawing observations with replacement from the original dataset. This allows us to differentiate the trees. As we said, trees are sensible to small change in the training data, and this help us to take into account the effect of the data variability on our predictors. This is also what *bagging* does. The problem here is that we still have a great correlation among the trees, and it has been proved that it influences negatively the performance of the model. That's why Random forests make a another step in terms of randomness. In fact, not all the possible features are considered when a tree is built. So, at training time, each tree can use only a subset of the original covariates to produce the result. This leads to an improvement both in term of robustness to outliers and in term of computational time. The computational time improvement is very important because it makes the algorithm feasible also for large datasets, as the one we use. At this point, we have all the information required to formalize the algorithm.

1. Choose $B$ as the number of trees in the forest
2. For $b = 1$ to $B$:
   (a) Get bootstrap sample of size N
   (b) Grow a tree $T_b$ to the bootstrapped data. At each terminal node use a subset of the original features to choose the best split
3. Output $\{T_b\}_1^B$

With the ensemble of trees we make predictions as:

- $\hat{f}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$ for *Regression.*
- $\hat{C}^B(x) = \textit{majority vote}\{\hat{C}_b(x)\}_1^B$ for *Classification.* $\hat{C}_b(x)$ is the class predicted by the $b_{th}$ tree.

To understand why this algorithm reduces variability we have to underline that trees in Random Forest are *i.d.* (*identically distributed*). In this case, the variance of the average is

$$\sigma_{ave}^2 = \rho\sigma^2 + \tfrac{1-\rho}{B}\sigma^2$$

The second term goes to zero as $B$ increases. The first term depends on $\rho$, the pairwise correlation among trees. Reducing $\rho$ results in reducing the variance of the model. To reduce the correlation among trees is fundamental to choose properly the number of features to use at each split when the trees are trained.
There are some guidelines about the parameters to choose to optimize the performances of Random Forest. For *classification*, it is suggested to choose the covariates for the splits from the square root of the total number of features, while for *regression*, it is better to use only $\frac{1}{3}$ of the total. These are good starting points, even if in general it is better to tune to optimal number of features according to the problem we are solving. In R, Random Forests are implemented in many packages, but the one we use is the *RandomForest* package, from the CRAN repository. It contains methods that allow us to tune the parameters we need and to keep track of the evaluation metrics. In particular we train the forest with the method *randomForest*, that is based on the original code proposed by Breiman and Cutler. Among all the parameters that this method has, we focused our attention on:

- *ntree*

  The number of trees to be trained. It corresponds to the $B$ parameter. The default value is 500.

- *mtry*

  Percentage of variables that can be chosen as optimal split at each node. The default values is $\sqrt{p}$ for *classification* and $\frac{p}{3}$ for *regression.* $p$ is the total number of features.

This method returns an object that contains many useful information about the predictors and the results. In particular we have:

- *confusion*

  It contains the confusion matrix in case of *classification* task.

- *mse*

  Vector of mean square errors for *regression.*

- *predicted*

  The predicted values for the input data.

These information are based on the *out-of-bag*(OOB) samples. Due to the bootstrap procedure, each observation is used only in a subset of the trees to calibrate them. So the idea is to use as predictor, for each sample, the average of the trees that have not that sample in the original bootstrapped dataset. This procedure is similar to the cross validation method, and it is useful because it is a good indicator of convergence for the Random Forest error at training time.
We said previously that increasing the value of $B$ is useful to reduce the variance of the total model. We have to ensure that this operation doesn't result in overfitting the data. In fact this could be one risk of a large model. But Random Forests are robust in this sense. In fact it can be proved that

$$\hat{f}_{rf}(x) = E_{\Theta|Z}T(x;\Theta(Z)) = \lim_{B\to\infty} \hat{f}^B_{rf}(x)$$

where T is the prediction function, conditioned on $\Theta$, that is a vector that contains the information about all the trees in the forest and that is dependent on the training data Z. So when B goes to infinity, the estimate approximates the value of the expectation.
Applying this limit to the previous formula of average variance we obtain :

$$Var\hat{f}_{rf}(x) = \rho(x)\sigma^2(x)$$

that is the the variance evaluated at a single point $x$. The factors that compose this equation are:

- $\sigma^2(x)$

  The sampling variance of a single tree of the forest.

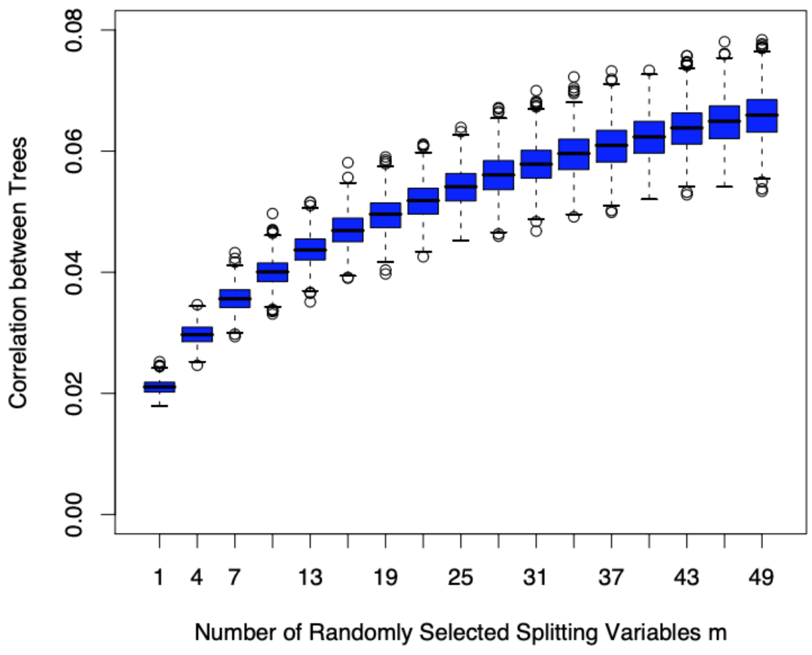


**Figure 2.2.** Boxplots of variance

- $\rho(x) = corr[T(x;\theta_1(Z)), T(x;\theta_2(Z))]$

  The sampling correlation between any pairs of trees. It represents the *theoretical* correlation between randomly drawn trees evaluated at $x$.

So, to reduce the variance of the ensemble, the first thing to do is to reduce the correlation $\rho$. This can be achieved by reducing the number of variables to use at each split. We can use $m$ to represent this value. Intuitively, if we force the trees to choose only from a small subset of variables at each split, it would be less probable to obtain similar trees in the forest. To show graphically this statement, we can use the results of a simulation with data coming from Gaussian random variable and random noise. After training the Random Forest on this model many times and with different values of $m$ we obtain the boxplots in figure 2.2. The most important aspect to highlight is that the effect of $m$ is not relevant on the variance of the single trees in the forest. This is the reason why the variance of the ensemble is significantly lower than the variance of CART.

One last thing to say regards the depth of the trees in the forest. In fact the

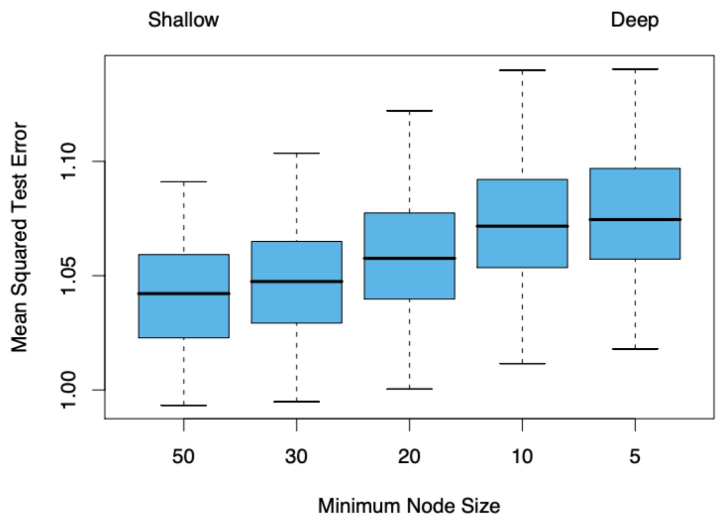


**Figure 2.3.** Boxplots of mean square error related to the minimum node size parameter

parameters that regularize the depth of the tree was fundamental in order to avoid overfitting and to improve performances in case of CART. This is not the case with Random Forest. As we can see in figure 2.3, the error of the Random Forest is poorly affected by the minimum node size. In this case we are using as evaluation metric the *mean square error* for *regression*, but it is also valid in *classification* case.

# Chapter 3

# Code overview

At this point, we have all the important information that are necessary to go through the software structure and to understand the choices we made during the development of the techniques that we used to improve the performances and the quality of the results, with particular attention to the individual claims reserve analysis.
The previous chapter conclusion talks about the improvement in term of variance of the Random Forest model, with respect to the CART techniques. This is the main aspect that led us to try to implement the same model proposed in chapter 1 with Random Forest. The aim of this choice is to find a method that is innovative in the individual analysis field. In fact there are different methods that are largely used for the aggregate analysis, that provide good results. CART estimates are quite good too, but in the individual analysis, they show the drawbacks we talked about in the previous chapter. This is the reason why they are not completely reliable for this kind of analysis. When we look at the individual claims reserve estimates, we analyze many claims that present a lot of differences among them. It usually happens that the most difficult claims are the ones we are more interested in, but they are also less probable to observe. So, we want to be accurate on them, but, at the same time, we don't want that a small part of data affects dramatically the predictions of the most common claims. In this sense CART are less reliable. Hence, with Random Forest, we want to provide another tool for this specific situation.
This chapter aim is to explain the general organization of the software and the way

it works from a general point of view. Then it goes into the details of each method in the correspondent paragraph.

## 3.1 Code structure

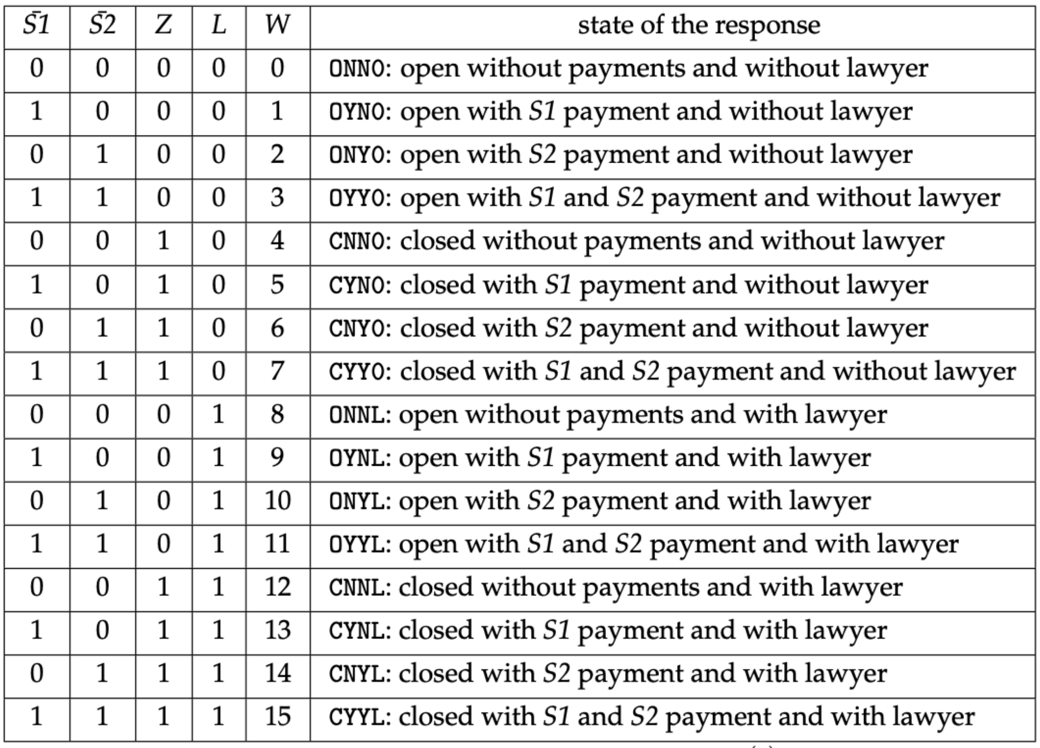
The original code structure is composed by four main sections:

- *Input*
- *Calibration*
- *Simulation*
- *Report*

In the first section, *Input*, there is the code related to the settings management. In this part the user can choose from many different options in order to run the program in the way he prefers. These options regard both data operations and model specification. The options regarding data are common for all the methods. They are used to specify static information about the input and to explicit if it needs to be modified before passing it to the model. At this point the user has to specify the minimum and the maximum accident years, such that the program can retrieve the *lag* number. He can choose if there are specific data he wants to exclude, or if he wants to use the entire dataset for the subsequent phases. He can also specify if data regarding payments need to be inflated or not, and what inflation rate has to be applied. Finally he can explicitly define what are the variables he want to use as covariates in the model. He can choose both static and dynamic variables. They are divided according to the part of the model they are used for. In fact variables for *frequency* and *severity* both for claim watching and case reserve are stored in different lists. At this point, all the options needed by the program are set and the user can run the calibration.
In the *calibration* section, the models are actually trained. This process is obviously dependent on the type of model the user chooses, but there are common actions that are model independent. As explained in the first chapter, the training phase is organized by *lag*. In practise this means that we repeat the same operations $l$

times, and for each iteration we store the trained models. The differentiation among the iterations is due to the data used. In fact, we said that data are different for each *lag* and that the quantity of covariates increase as $l$ increases. This is possible thanks to queries that are defined to dynamically change at each iteration. They use $l$ to calculate the correct column indices to get in the dataset. In the original database, data used for training contain all the information about the development of the claims. But, at training time, we obviously have to distinguish the feature columns and the response columns. We have to remember that, in dynamic modeling, response columns become feature columns for the next years. This is the reason why we have to calculate correctly the indices of the columns to use at each iteration.
There is a fixed number of queries at each iteration. Each query corresponds to a specific part of the general model (*frequency*, *severity*, case reserve). Before using the samples retrieved by the queries, it is necessary to explicitly define the response variables. For the *severity* parts it is sufficient to choose the correct columns from the dataset. For the *frequency* case, we have to define the response variable as explained in the first chapter, by manipulating the correct columns to create a single numeric label. It is the 16-states response variable explained in the first chapter. We can observe an example, for one *lag*, of all the possible states in image 3.1.

The case reserve analysis needs a special mention. In fact it has its own *simil-frequency* and *simil-severity* sections. They are similar to the *frequency* and *severity* sections we discussed before, but for case reserve. They are based on the variables *RCA* and *RNC* and on the case reserve estimates for the CARD and NoCARD cases. This section is divided into four sub-levels, that come from the combination of CARD and NoCARD with the states *open* and *closed.* This distinction is useful to make the models more precise and to give them similar data at training time.
The last common part in the *calibration* is the estimate of the *damage* associated to each claim. Machine learning models are not involved in this estimate. In this case we use a transition probability matrix to define the probabilities of a claim to change state during its life. This variable estimate is important because it usually happens that the most complex claims are associated to a higher damage level.
*Calibration* process can require a lot of time to complete, depending on the quantity



| $\bar{S1}$ | $\bar{S2}$ | $Z$ | $L$ | $W$ | state of the response |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | `ONNO`: open without payments and without lawyer |
| 1 | 0 | 0 | 0 | 1 | `OYNO`: open with *S1* payment and without lawyer |
| 0 | 1 | 0 | 0 | 2 | `ONYO`: open with *S2* payment and without lawyer |
| 1 | 1 | 0 | 0 | 3 | `OYYO`: open with *S1* and *S2* payment and without lawyer |
| 0 | 0 | 1 | 0 | 4 | `CNNO`: closed without payments and without lawyer |
| 1 | 0 | 1 | 0 | 5 | `CYNO`: closed with *S1* payment and without lawyer |
| 0 | 1 | 1 | 0 | 6 | `CNYO`: closed with *S2* payment and without lawyer |
| 1 | 1 | 1 | 0 | 7 | `CYYO`: closed with *S1* and *S2* payment and without lawyer |
| 0 | 0 | 0 | 1 | 8 | `ONNL`: open without payments and with lawyer |
| 1 | 0 | 0 | 1 | 9 | `OYNL`: open with *S1* payment and with lawyer |
| 0 | 1 | 0 | 1 | 10 | `ONYL`: open with *S2* payment and with lawyer |
| 1 | 1 | 0 | 1 | 11 | `OYYL`: open with *S1* and *S2* payment and with lawyer |
| 0 | 0 | 1 | 1 | 12 | `CNNL`: closed without payments and with lawyer |
| 1 | 0 | 1 | 1 | 13 | `CYNL`: closed with *S1* payment and with lawyer |
| 0 | 1 | 1 | 1 | 14 | `CNYL`: closed with *S2* payment and with lawyer |
| 1 | 1 | 1 | 1 | 15 | `CYYL`: closed with *S1* and *S2* payment and with lawyer |

**Figure 3.1.** Example of response states for one *lag*. $\overline{\mathrm{S1}}$ and $\overline{\mathrm{S2}}$ refer to CARD and NoCARD payments.

| | | | | feature-response pairs reorganized by lag ($\ell = j + k$) | | | |
|---|---|---|---|---|---|---|---|
| cc | ay: $i$ | rd: $j$ | $\nu$ | $\ell = 0$ | $\ell = 1$ | $\ell = 2$ | $\ell = 3$ |
| 1 | 1 | 0 | 1 | $\left(\boldsymbol{x}^{(1)}_{1,0\|0}, \boldsymbol{Y}^{(1)}_{1,0\|1}\right)$ | $\left(\boldsymbol{x}^{(1)}_{1,0\|1}, \boldsymbol{Y}^{(1)}_{1,0\|2}\right)$ | $\left(\boldsymbol{x}^{(1)}_{1,0\|2}, \boldsymbol{Y}^{(1)}_{1,0\|3}\right)$ | $\left(\boldsymbol{x}^{(1)}_{1,0\|3}, \cdot\right)$ |
| 2 | 1 | 1 | 1 | no | $\left(\boldsymbol{x}^{(1)}_{1,1\|0}, \boldsymbol{Y}^{(1)}_{1,1\|1}\right)$ | $\left(\boldsymbol{x}^{(1)}_{1,1\|1}, \boldsymbol{Y}^{(1)}_{1,1\|2}\right)$ | $\left(\boldsymbol{x}^{(1)}_{1,1\|2}, \cdot\right)$ |
| 3 | 1 | 2 | 1 | no | no | $\left(\boldsymbol{x}^{(1)}_{1,2\|0}, \boldsymbol{Y}^{(1)}_{1,2\|1}\right)$ | $\left(\boldsymbol{x}^{(1)}_{1,2\|1}, \cdot\right)$ |
| 4 | 1 | 3 | 1 | no | no | no | $\left(\boldsymbol{x}^{(1)}_{1,3\|0}, \cdot\right)$ |
| 5 | 2 | 0 | 1 | $\left(\boldsymbol{x}^{(1)}_{2,0\|0}, \boldsymbol{Y}^{(1)}_{2,0\|1}\right)$ | $\left(\boldsymbol{x}^{(1)}_{2,0\|1}, \boldsymbol{Y}^{(1)}_{2,0\|2}\right)$ | $\left(\boldsymbol{x}^{(1)}_{2,0\|2}, \cdot\right)$ | . |
| 6 | 2 | 1 | 1 | no | $\left(\boldsymbol{x}^{(1)}_{2,1\|0}, \boldsymbol{Y}^{(1)}_{2,1\|1}\right)$ | $\left(\boldsymbol{x}^{(1)}_{2,1\|1}, \cdot\right)$ | . |
| 7 | 2 | 2 | 1 | no | no | $\left(\boldsymbol{x}^{(1)}_{2,2\|0}, \cdot\right)$ | . |
| 8 | 3 | 0 | 1 | $\left(\boldsymbol{x}^{(1)}_{3,0\|0}, \boldsymbol{Y}^{(1)}_{3,0\|1}\right)$ | $\left(\boldsymbol{x}^{(1)}_{3,0\|1}, \cdot\right)$ | $(\cdot, \cdot)$ | . |
| 9 | 3 | 1 | 1 | no | $\left(\boldsymbol{x}^{(1)}_{3,1\|0}, \cdot\right)$ | $(\cdot, \cdot)$ | . |
| 10 | 4 | 0 | 1 | $\left(\boldsymbol{x}^{(1)}_{4,0\|0}, \cdot\right)$ | $(\cdot, \cdot)$ | $(\cdot, \cdot)$ | . |

**Figure 3.2.** Training data organized by *lag*.

of data used and to the model choices. So, the user can save the result of the calibrations and use them in different *simulation* scenario.

*Simulation* is the section of the program that is able to use the calibrations obtained in the previous step, to simulate the development of the claims and to fill the empty cells of the matrix of data. We can use as example the figure 3.2. We can see data organized by *lag*, observed at time $I = 4$. The highlighted cells represent the labelled samples that we use at training time for each *lag*. The cells with only the $x$ vector of features are related to the *one-year* prediction problem. The other ones are related to the *multi-period* prediction problem. The *simulation* section is able to fill both this kind of cells, by applying subsequently to each single claim, the *lag* models it needs to predict its path. The cells without the feature vector will be filled using the response vector of the previous model as input of the next model. This process is limited by the number of *lags*, hence by the number of models we can train.

As in the previous section, some parts of the code are specific for the chosen algorithm. Anyway, we can look at the fixed operation we do. First of all, it is necessary to have a calibration loaded in the environment. Thanks to this calibration, it is possible to retrieve all the settings needed by the simulation to start. Here, the user can set other options that are independent from the settings defined in the first section. One of these is related to the kind of analysis he wants to perform : *individual*, *aggregate* or both. The *aggregate* analysis is able to produce an overall estimate of the payments related to all the claims for each year. The *individual* analysis focus the attention on the development of the single claim, for all the claims in the dataset or only for a subset of them, specified by the user. Another option is related to the estimate of the IBNYR claims. Although they are outside the scope of this thesis, they have to be taken into consideration in the final cost estimate, in particular for the *aggregate* analysis. Finally the user can also choose the number of iterations of the simulation.

Now it is possible to use a unique query to retrieve all the data and to keep only the claims at he maximum observation year $I$. This dataset is used to get subsets of claims and it is completely rebuilt at each iteration. Each iteration of the simulation contains another loop. This is a loop over the *lags* needed to develop the claims paths. At the beginning of each iteration of this inner cycle, we have to select the claims that we have to pass as input to the trained models of the current *lag*. For example, if we have $l = 0$, we have to choose only the subset of claims that are happened in the maximum accident year, because they are the only ones that need the models of *lag* 0. At this point the input data are passed to the model of the *frequency* section that predicts the *16-states* response. The resulting value is then decomposed and it is used to fill the correspondent columns in the dataset. In particular the predicted variables are: *YCA*, *YNC*, *Z* and *L*. When all these categorical variables have been predicted, we can simulate the *severity* responses. The interested variables for this section are : *pagCA* and *pagNC*. The model used for the *severity* is conditioned on the results obtained in the *frequency*. In fact we have a *severity* for the claims with CARD payments and one for the claims with NoCARD payments. The choice of the model depends on the values related to the CARD and NoCARD payments that have been predicted in the *frequency* section.

In a similar way we can explain the case reserve section. It has indeed its own *simil-frequency* and *simil-severity* sections, and their behavior is analogous to the classic *frequency* and *severity* parts. The difference is in the variables we use now.

The focus is now on *RCA*, *RNC*, *caseRCA*, *caseRNC*. Another difference regards the type of claims we are interested in. In fact, the subset of data we choose is based also on the variable $Z$, that has to be equal to 0. So, we model the case reserve only for open claims. The value of $Z$ has been predicted in the *frequency* section, hence, in a sense, the case reserve modeling is conditioned on the main *frequency-severity* original model. From the *simil-frequency*, we obtain another single response value, that has to be decomposed. The results of this process are stored in *RCA* and *RNC*. Then, before going to the *simil-severity* part, we need to make a distinction among the open claims. We want to distinguish the claims with CARD and NoCARD payments, and in addition we want also to differentiate claims that were previously closed from the ones that were open. In this way we use four different models for the *simil-severity*, that is exactly what happened at calibration time.
For all the *severity* models, the conditioning process doesn't happen directly by looking at the predicted result of the correspondent *frequency* model. Instead we use the probability distributions over the classes, returned by the R function *predict*. We use these distributions as parameters of the *sample* function, that use them as weights vectors for the extraction of the response associated to each claim. In this way we include another source of randomization in the simulation process.
The last operation in each iteration regards the collection of the results. The method of collection depends on the type of results we want to observe (*individual*, *aggregate*). When all the iterations are completed we use the stored value to compute the summary statistics.
Finally in the last section, *Report*, we simply write an excel file to visualize these statistics in a readable format.

## 3.2 CART approach

The original code implemented only the CART approach to the problem with a limited number of options the user could choose. The possible settings in the *input* section regarded the cost complexity parameter, the type of pruning, the type of *severity* approach and the building criterion for the regression trees. In particular, the user could choose between two types of *severity*: the first one is the

classic *severity*, with CARD and NoCARD trees for each *lag*; the second one is a more complex approach to the problem. In fact it consists in training regression trees for each *leaf* of the *frequency* decision tree at each *lag*. Anyway, the complete explanation of this approach goes outside the scope of the thesis, because we did not change anything about it. The more interesting part is the one related to the building criteria of the *severity* trees. We started from the original likelihood method, based on the Normal distribution and we added also the LogNormal distribution. But we will see these methods better in the next sections of this chapter.
We started working on the code with a small subset of the original data, because the time required by the software to run with a greater quantity of samples was too large and it did not allow us to make changes in a reasonable time. So, we excluded from each *lag* some accident years. Another problem was related to the type of report we have for the results. In fact, there we find the resulting statistics of the whole simulation process, that is affected by the aggregation of the errors of all the models in all the *lags*. This means that we have to face multiple optimization problems, because each *lag* produces its own prediction functions, calibrated on its own dataset. This create problems to find the errors sources. For example, errors on the estimate of the case reserves in the last *lag*, may not be due to bad calibration in the *regression* tree that produces them, but they can be originated by an error produced by the *decision* tree calibrated in the first *lag*, and this is something that actually happened. For this reason we had to run the code piecewise multiple times, and we had to evaluate manually each part of each single *lag*. This is also the reason why we did not perform an automatic process of parameters tuning. It would have required a lot of time because it should have been performed on each model of each *lag*, and it would have been specific for the data that we were using for the development. So other users should have repeated the same process, but, again, it is infeasible in terms of time. Hence, we started looking at the models by *lag* at calibration time, evaluating the *frequency* part with the confusion matrices, and the *severity* part with some distance functions(*mean square error*, *residuals*). One of the first thing we noticed was the number of the incorrectly predicted samples. Although it was small in general, it was too large if we looked to the samples by

class. In fact, some classes were completely misclassified. These classes showed the lowest number of training samples in the dataset. So, the algorithm maximized its result by predicting correctly the samples related to the most frequent classes. In particular the response 4 (claims closed without payments and without lawyer) is the most present and its not comparable to the number of samples of all the other classes. This depends on the fact that most of the reported claims are simple, therefore, they are closed by the company quickly. For our algorithm this represents a problem, because the dataset results unbalanced, and the most difficult claims are misclassified because of the simplest ones. To solve this problem we introduced a *two-steps* model for the *frequency*. In the first step we define a binary classification problem, to distinguish between claims of class 4 and claims of the other classes. Then, in the second step, we train the classical multi-class algorithm on all the claims that are not in class 4. This upgrade resulted in a great improvement of the confusion matrices and, consequently, in an improvement of the final estimates.
Problems with *severity* regard mainly the lack of smoothness in the predictions. Sample mean is not always a good predictor for the individual claims reserve, and its performances are worse when we prune the *regression* trees. This often result in a tree that has zero or one splits. This means that the amount of predicted payments its equal for most of the claims. We are currently working on this problem for CART, with the Normal and LogNormal distributions for the *severity* trees. This is still another reason that makes CART less feasible for individual analysis, even if they still perform good on the aggregate. But this is due to a compensation effect among claims.

## 3.3 Random Forest approach

Most of the considerations of the previous paragraph are valid for Random Forest too. At first, we created new parts of code that were very similar to the existing ones, but with the Random Forest algorithm. As for CART, we noticed that Random Forest suffered the unbalanced classes problem too. So we used the two steps model also in this case. Random Forest introduce also new parameters that have to be chosen. Initially, we tried to run the code with the default parameters. First thing

we had to limit was the number of trees in the forest. The default parameter value was 500, and it worked with the initial subset of data, but it resulted in a memory error when we tried to increase gradually the number of samples. So, this parameter has to be decided taking into account the amount of data used, and so we added to the *Input* section, the option to customize the configuration of the algorithm. We decided also to include the *mtry* parameter to allow the user to change its value. In fact, default values are generally suggested, but they are not optimal for every situation. This is the case when the number of covariates is small. With the default parameters it is possible to discard the most significant features at each split, and this could result in a poor performance of the model. We have a similar situation in the first *lags*, when the number of covariates is still low. In our case we have to pay attention to the choice of this parameter, because for the first *lags*, increasing *mtry* could improve the performance, but on the other hand, the same value for the latest *lags* increases the correlation value among the trees, and, as we saw in chapter 3, this reduce dramatically the benefits of using an ensemble method. This parameter affects also the computational time required by the *Calibration* section. In general, with default parameters, the calibration time of Random Forest is not significantly greater than the time required by CART. This is due to the fact that, even if we are training many trees, the number of splits that each tree has to try is dependent on the number of variables we want to use. So, increasing them means increasing the number of possible splits that the algorithm have to perform to find the best one.
On the other hand, with Random Forest, we have not to worry about the *pruning* process anymore, because the depth of the trees has not an important effect on the quality of the predictions.
Finally we have to say that, for *severity*, we cannot consider the second approach explained in the CART section. This approach needs a well defined concept of *leaf*, that is not present in Random Forest.

## 3.4 Normal and LogNormal distributions

*Regression* trees use, as prediction function, the means of the samples that are grouped in each *leaf*. This means that at prediction time, the samples that fall

into a *leaf* will receive the same predicted value. Even if this could still be good in an aggregate analysis scenario, it lacks of precision when we want to perform individual claim evaluation. Random Forest avoids this problem because average over all the predictions made by each tree. To solve this problem with *regression* trees, we implemented a different method of building the tree. This method is based on the *likelihood* of distributions that can be chosen by the user. Until now we have implemented the Normal and the LogNormal distribution. The main difference is in the kind of estimate that we perform during the calibration. In fact now the tree that uses these methods, is able to estimate the parameters of the chose distribution, such as *mean* and *variance*. This is fundamental at simulation time, because now the value are predicted by applying the estimated parameters to randomly draw directly from the distribution. This allows us to take into account more variability among claims of the same *leaf*.
To add this methods we had to implement new custom splitting criteria and all the related functions required by the R documentation [5]. In particular in the *Input* section we defined:

- *Initialization* function
- *Evaluation* function
- *Splitting* function

The *initialization* function defines the structure of the parameters needed and the type of output response that will be returned. It can be also used to customize the built-in functions such as *rpart.plot* and *summary.rpart*, to define new types of output representations. The *evaluation* function returns a *deviance* value for each node. Finally the *splitting* function performs the actual splitting process. It is called for each covariate and it produces two output values: *goodness* and *direction*. *Goodness* represent a measure of the importance of the split, *direction* is used to decide if samples that are lower than the splitting point go to the left child or to the right child of the node.
Both for Normal and LogNormal distribution, the goodness of split is given by ratio between the sum of the children negative loglikelihood and the parent negative

loglikelihood, subtracted from 1.

The estimated parameters for the Normal distribution are $\mu$ and $\sigma$. This allows the algorithm to fit a Normal distribution for each node until it reaches the *leaves*. Anyway, a normal distribution is not always able to fit the samples distributions, because they can be very different among them and they can contain some important values also in the tails. For this reason we implemented the LogNormal distribution, that has a higher number of parameters. In fact, with $\mu$ and $\sigma$, that now are respectively the *location* and *scale* parameters, we estimate also the *skewness* and *threshold* $\beta$. These parameters allow to change the position and the direction of the distribution. In particular, the *skewness* regards the direction of the sample distribution, hence the position of the longest tail. $\beta$ instead is used to change the starting point of the fitted distribution. It is estimated from data and in our code we do it by solving an optimization problem. To improve these methods we did different tests, working both on the algorithm and on the input data.The most relevant drawback is the computational time required at *Calibration*, that makes difficult to test the algorithms in a reasonable time. For this reason we are still working on them, to make them more accurate and more feasible for large quantity of input data.

# Chapter 4

# Results

In this chapter we present the results obtained with both CART and Random Forest, without showing the actual values of the reserves, but focusing the attention on the comparison between the algorithms. To perform the evaluation, we based on a subset of claims that have 2008 as minimum accident year and 2013 as maximum accident year. With this dataset, we calibrated the models with the CART approach using the 1-SD pruning, $cp = 10^{-3}$ and 50 elements in each *leaf* at mimimum. The Random Forest was calibrated with $ntree = 50$ and the default values for *mtry*. Both the algorithms use the *two-steps* model for *frequency*, the *lag* approach for the *severity* and they performed 1000 iterations at *Simulation*. No inflation rates have been applied. We focused our attention on the individual claims reserve analysis, in case of CARD and NoCARD payments. For each claim we compared the real reserve value with the predicted one, both for CART and Random Forest. Then, we took the absolute value of the residuals of the CART predictions and we compared them to the ones of the Random Forest. In image 4.1 we can observe, in the first pie chart, the number of times the error of CART is higher than the error of Random Forest on the same claim, in case of CARD reserve. It is evident the improvement in term of quality of prediction. It is the same even for NoCARD reserve, as we can see in figure 4.4.
Another crucial point we wanted to test is the stability of the Random Forest. To do this, we created three different version of the original dataset: one with all the claims, as the one we used until now, one with the 95% of the claims and one with

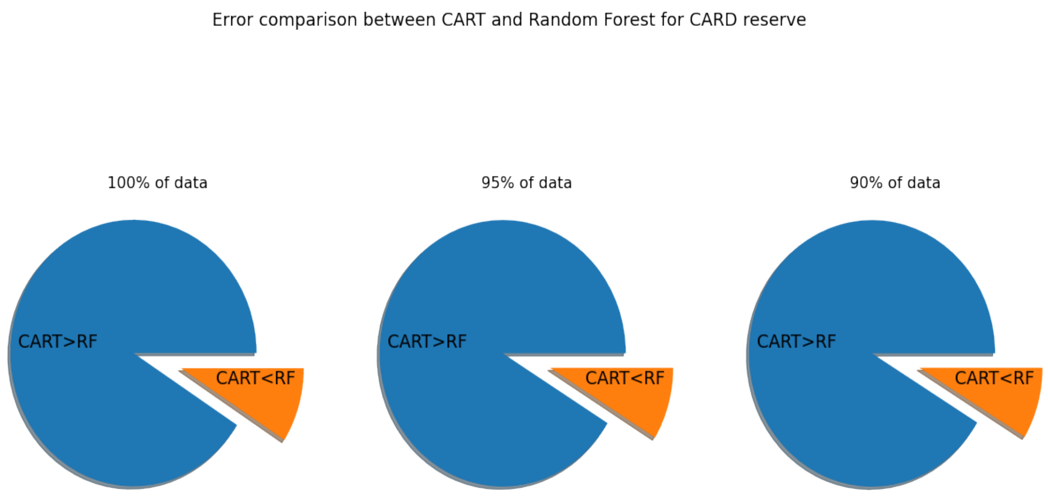


**Figure 4.1.** Comparison between the error on CART and Random Forest with all the three trained models for the CARD reserve.

the 90%. The discarded claims were chosen randomly. Then we calibrated again both the models on the three datasets. Again we looked at the prediction of the claims reserve. So, for each claim, we have three different predictions, one for each of the three calibrated models. To check the stability we used the standard deviation among these three predictions. Finally we obtained one *sd* for each claim, both for CART and Random Forest. At this point we evaluated, claim by claim, the difference between the standard deviation obtained with the two algorithms. In figure 4.2 we can observe the number of times the CART standard deviation is higher than the Random Forest standard deviation for CARD reserve, and in figure 4.3 we can observe the same for NoCARD reserve. This happened in the majority of the cases, and this is exactly the result we wanted to obtain. We evaluated also the error on the results of the other two models, that we trained on the subsets of the original dataset. We repeated the method that we used before. In the figure 4.1 we can observe that in all the cases, Random Forest predictions for the CARD reserve are more precise than the CART ones, even with a smaller number of training samples. For NoCARD reserve, this result is even more evident (figure 4.4).

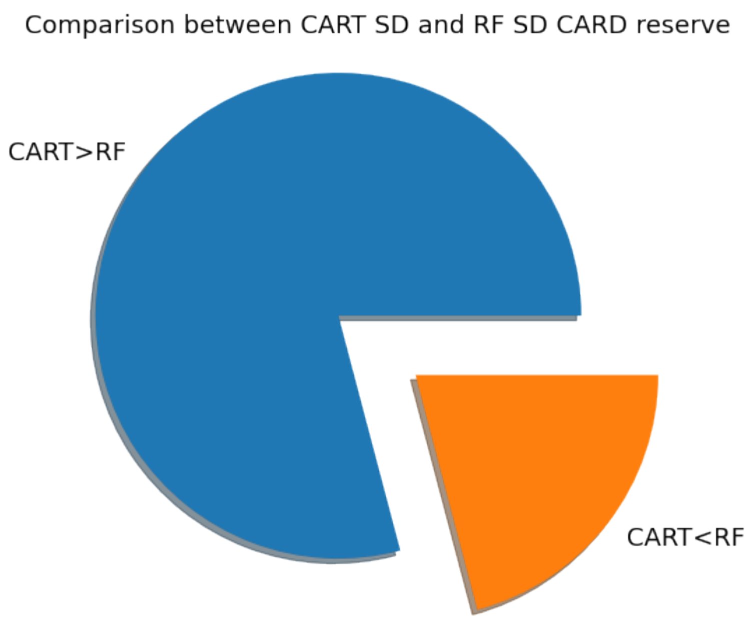


**Figure 4.2.** Comparison between the *sd* on CART and Random Forest for CARD reserve

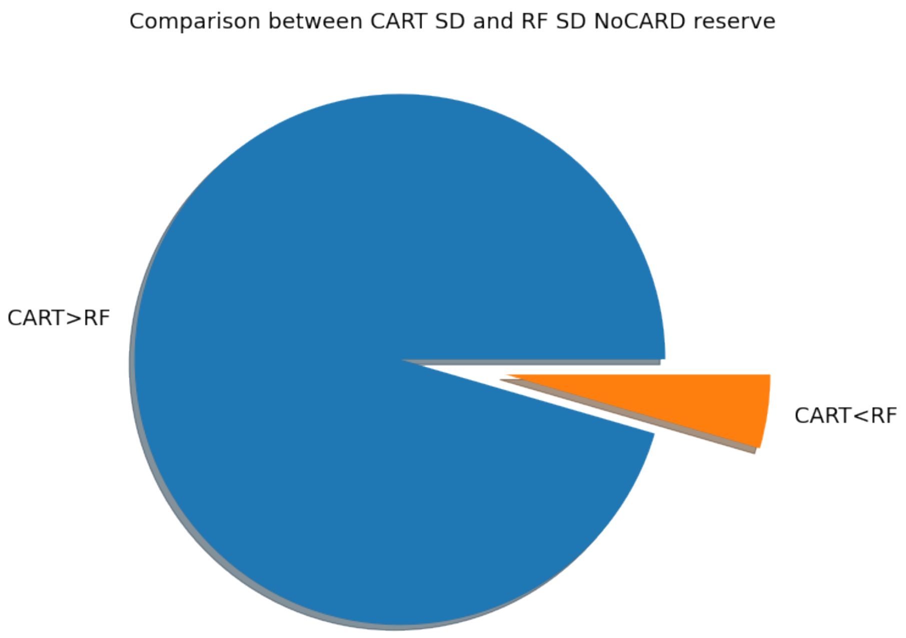


**Figure 4.3.** Comparison between the *sd* on CART and Random Forest for NoCARD reserve

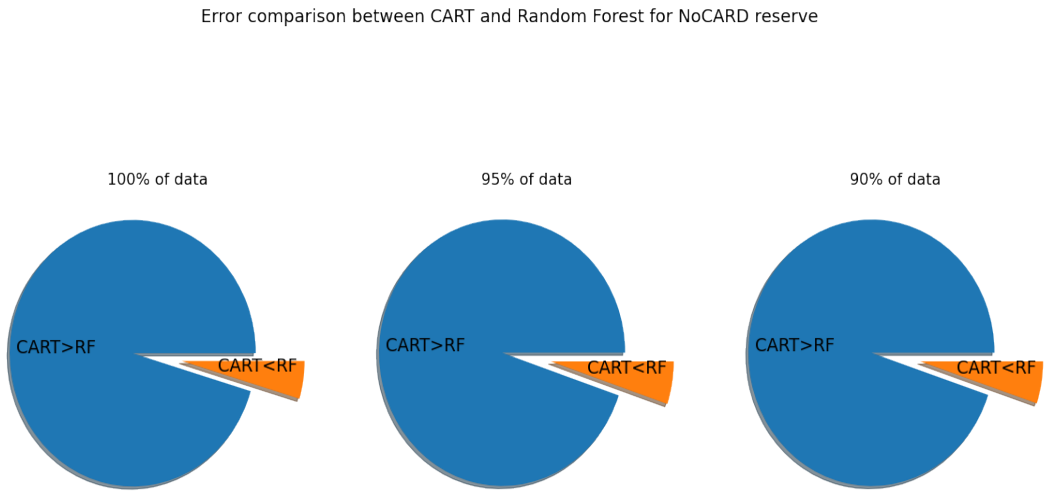


**Figure 4.4.** Comparison between the error on CART and Random Forest with all the three trained models for the NoCARD reserve.

# Chapter 5

# Conclusion

As shown in the results, Random Forest algorithm provides good improvements in the quality of the predictions and in the stability with respect to the data used at training time. Moreover the computational time required by the algorithm is still comparable to the training time required by CART. It keeps the same useful properties of CART in term of elasticity and capability to individuate the most important features, but at the same time it reduces drastically the risk of overfitting, that in CART was mainly caused by the depth of the tree and by the parameters that control it. This allows the user to have more freedom on the choice of the parameters. In Random Forest, the most important thing to pay attention to is the number of features used at each split. We noticed that this could create problems in the aggregate analysis, because final predictions are affected by the errors at first *lags*, where the number of covariates is still low, and, consequently, the probability to not use an important feature for the split is higher. At the same time, increasing the value of *mtry* for the first *lags* could result in increasing the correlation among the trees in the last *lags*. For this reason we are planning to implement some ideas that can solve this problem. For example we will add to the original dataset new covariates, that are the result of linear combinations among the most important features. Moreover we will try to introduce an higher customization level for the user. We will try to let the user choose different parameters for each *lag*. These are some of the upgrades we want to apply that go in a well defined direction: improve software elasticity and usability. In fact, we have to remember that our final objective is not to maximize the performance of the model for the dataset we used for this thesis, but to produce a software that is able to adapt efficiently to all the data the insurance companies are willing to use it for. In this sense, Random Forest is a very good and innovative way to follow for the individual claims analysis.

# Bibliography

[1] Breiman, L. (2001) Random Forests. *Machine Learning*, 45, 5-32.

[2] Breiman L., Friedman J.H., Olshen R.A., Stone C.J. (1998). *Classification and Regression Trees.* ChapmanHall/CRC.

[3] De Felice, M., Moriconi, F. (2019). Claim watching and individual claims reserving using classification and regression trees. *Risks.*

[4] Hastie T., Tibshirani R., Friedman J. (2017). *The Elements of Statistical Learning. Data Mining, Inference, and Prediction.* 2nd edition. Springer.

[5] Therneau, T., M. (2015). User written splitting functions for RPART. *Mayo Clinic. Disponível.*

[6] Therneau, T.,M., Atkinson, B., and Ripley, B. (2014), *rpart: Recursive Partitioning and Regression Trees.* R package version 4.1-8.

[7] Therneau,T.M., Atkinson. E.,J. (2022). An introduction to recursive partitioning using the RPART routines. Vol. 61. *Mayo Foundation: Technical report.*

[8] Wüthrich M.V. (2017). Data Analytics for Non-Life Insurance Pricing. *Draft Lecture Notes.* ETH Zurich.

[9] Wüthrich, M.V.(2016). Machine Learning in Individual Claims Reserving. *Swiss Finance Institute*, Research Paper n.16-67.