Case Reserving in Non-Life Practice using Individual Data and Machine Learning

Marco Aleandri
Ph.D. Student in Actuarial Science
Dept. of Statistical Sciences
Univ. "La Sapienza", Rome
Email: marco.aleandri@uniroma1.it

Abstract

This paper focuses on the potential of machine learning tools in micro-level reserving by using individual claim data, which is more and more available nowadays. This is especially relevant for non-life insurance, but it could also be useful for some specific life business branches.
After a brief introduction to the problem of reserve estimation in non-life, we will describe the algorithms behind some of the fundamental machine learning tools such as regression methods, naive Bayes, k-nearest neighbors, CARTs, and neural networks. All of them will be used to estimate closing delay and payment amount for individual claims of a specific automobile bodily injury claim dataset. Theoretically, these estimations represent the foundation for a triangle-free, machine-learning-based approach to non-life reserving.

Keywords: individual claims, case reserving, machine learning, generalized regression, naive Bayes, k-nearest neighbors, classification and regression trees, neural network

1. Introduction

Actuarial practice in non-life reserving is traditionally based on aggregate claims data structured in triangles. In fact, this has been proved (for instance, in [2] and [11]) to be an effective approach as long as we face high-probability low-impact claims such as those of motor insurance. Run-off triangles - like that in Figure 1 - are fundamentally based on the assumption that the reserve on future claim payments depend on the reporting year (or accident year) and closing delay only.

Actually, it is far to be true, unless the claims tend to be extremely homogeneous. And even in that case, triangle-based estimations will neglect relevant information about individual claims. This was necessary when actuaries had to use data in times of strong computational limits. Nowadays, this is no longer a major constraint. This is the reason why more and more studies promote micro-level reserving based on individual claims data, for instance [1], [7], [9], [12], [13], and [15]. To some extent, all of them assumes a rather fixed structural form for the timing or the amount of the payments. Unfortunately, such approaches are as rigid as any other parametric method, and cannot take into account all the available details for claims.

In the recent years, some researchers tried to introduce machine learning in actuarial practice by tackling classical problems with new techniques. In fact, such data-driven tools can ultimately overcome the issues explained so far in this section about more traditional approaches. Restricting ourselves to non-life practice, it is worth citing

[18] which applies classification trees to estimate number of future payments varying by accident year and reporting delay,

[19] which utilizes neural network to handle heterogeneity in data and improve Chain-Ladder reserving,

[20] which presents machine learning tools in non-life pricing.

This paper is especially inspired by [18], whose author uses decision trees to predict the number of payments by accident year and closing delay. On our side, we will try to extend those ideas by directly predicting claim amounts using a wider range of machine learning tools, in order to choose the most accurate one. However, a major difference with respect to [18] and, more generally, actuarial practice in non-life reserving, regards the concept of reserve we are going to refer to. Even if the results will be reported in the traditional

reporting year	*closing delay* 0	1	...	$n-1$	n	*tail*
$T-n$	$P_{T-n,0}$	$P_{T-n,1}$	...	$P_{T-n,n-1}$	$P_{T-n,n}$	$\widetilde{P}_{T-n,c}$
$T-(n-1)$	$P_{T-(n-1),0}$	$P_{T-(n-1),1}$	...	$P_{T-(n-1),n-1}$	$\widetilde{P}_{T-(n-1),n}$	$\widetilde{P}_{T-(n-1),c}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
$T-1$	$P_{T-1,0}$	$P_{T-1,1}$	...	$\widetilde{P}_{T-1,n-1}$	$\widetilde{P}_{T-1,n}$	$\widetilde{P}_{T-1,c}$
T	$P_{T,0}$	$\widetilde{P}_{T,1}$	...	$\widetilde{P}_{T,n-1}$	$\widetilde{P}_{T,n}$	$\widetilde{P}_{T,c}$

Fig. 1: Run-off triangle of payed amounts, and related reserve estimations (in red)

reporting-year-per-closing-delay format, our final goal is NOT the estimation of the ultimate loss reserve as at year end. By contrast, we are interested in estimating the so-called *case reserve*, that is, the final cost of each single claim on its own.

After a brief introduction to the reserving problem in non-life insurance (see Section 2) and the probabilistic model at the base of our analysis (see Section 3), we will recall some features of the machine learning tools we intend to use (see Section 4). All of them will be applied to publicly available automobile bodily injury claim data, in order to estimate individual case reserves. The analysis is presented in Section 5. In line with the underlying model, the results will be broken down into three steps:

1. closing delay estimation of individual claims through naive Bayes, k-nearest neighbors, and classification tree (see Subsection 5.2)

2. payment amount estimation of individual claims through generalized regression, regression tree, and neural network (see Subsection 5.3)

3. case reserve estimation of individual claims through some combinations between tools in 1. and tools in 2. (see Subsection 5.4).

While no machine learning tool will be able to explain a relevant amount of variance at the closing delay's step, payment amount estimations will be quite accurate though. More specifically, as pointed out in Section 6, the estimations returned by decision trees and neural networks will be significantly more accurate than those from generalized regression. Of course, this is not

to prove that machine learning can always outperform more traditional approaches. Nonetheless, it should provide actuaries with further techniques to better estimate reserves when it comes with skewed and heavy-tailed claim distributions.

2. Understanding claim timeline

Generally, a claim is triggered by an accident causing a damage covered by the insurance contract. In an ideal world, the related benefit is paid as soon as the accident occurs, but often this is not the case in non-life insurance. In fact, a number of years may pass between the effective occurrence and the final claim payment (or payments). This time gap represents the reason why insurance companies must allocate reserve sufficient to cover any future payments for outstanding loss liabilities.

Assume the premium is paid in t_0 for an insurance protection that is immediately effective for a period T. During that period, an accident occurs at time $t_a < T$, the so-called *accident date*. Ideally, the accident is immediately reported to the company, but for a number of reasons it may happen differently, that is, the accident is reported at any time $t_r \geq t_a$, the so-called *reporting date*. The difference $\Gamma := t_r - t_a$ is the *reporting delay*. If it is small, say days, it does not really represent a problem to the company. However, if the reporting delay extends for years, it generates an unknown outstanding loss liability for the company, which is backed by the so-called *Incurred-But-Not-Yet-Reported reserve* - or IBNYR reserve. Actually, the related claims are not in the company's systems yet, so data-driven tools are hardly adaptable to this problem. For this reason, we will not estimate the IBNYR reserve in this paper - it will be shortly discussed in Section 6 as a topic for further development.

As soon as the accident is reported, the company is able to collect information about it, which represents the starting point for our analysis. Typically, the claim cannot be settled immediately for a number of reasons, including further investigation, new information, court decisions, and so on. As a consequence, the claim is actually closed only at a future date $t_c \geq t_r$, the so-called *closing date*. The difference $\Delta := t_c - t_r$ is the *closing delay*, which may have very different features depending on the specific non-life business involved as well as the claim severity. For standard claims, it might be very small, like in health insurance contracts for the employees of a firm: the company receives standard claim documentation from the policyholder, ap-

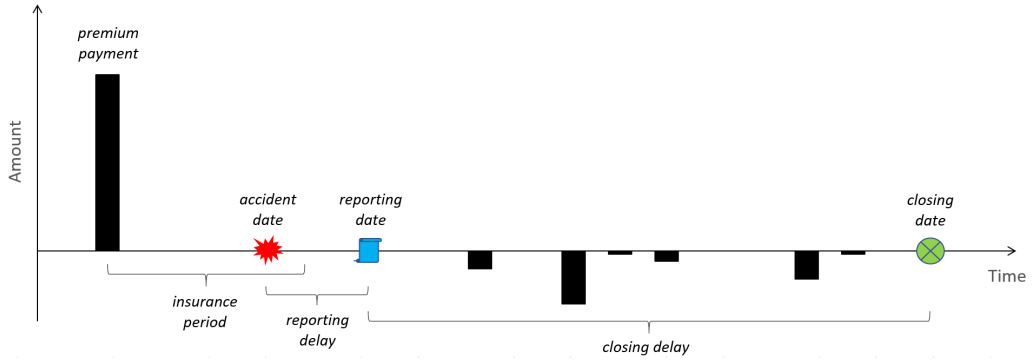

Fig. 2: Graphical representation of the claim timeline

prove it quickly, and refund him/her with one single payment. On the other hand, more severe claims often lead to more tortuous - and longer - closing delay: sometimes no payment is due, sometimes a final payment is due, and sometimes company's investigation justifies claim benefits all the way along and a series of cash-flows is correspondingly paid. At reporting date, the company must allocate reserve to cover the payments expected during the closing delay. Such a reserve is the so-called *Reported-But-Not-Yet-Settled reserve* - or RBNYS reserve. Given that it is allocated at reporting date, when some information about the claim is already known, we can use it to build our individual reserve estimation.
The process described in this section is represented in a sort of timeline in Figure 2.

3. Assumptions and model

Since our valuation date is the reporting date, we can assume that all the information about the claim is known at that date, and thus it can be used to predict future payments. This is a first simplification, which may be unacceptable in some specific cases. In fact, one of the causes of the closing delay is the further investigation by the company, which could discover new information at a later date. For sake of simplicity, we will ignore this possibility.
According to the explanation in Section 2, the company pays an amount to the policyholder as soon as it is justified by the aforementioned investigation, thus generating a series of cash-flows. The emerging of such cash-flows is

modeled in [18]. However, it would be a further complication for us, and the available data does not include those details. Therefore, we will assume one single, aggregate payment for each claim due at closing date.

Once these two assumptions are accepted, the model is a rather simple one. Assume that the closing delay Δ is a discrete variable measured in years, say $0, \ldots, m$. That means: no payment is delayed more than m years after the reporting date. Moreover, consider a number of predictors $x_1, \ldots, x_n$, that is, information about the policyholder, the claim, or any other relevant details. They are all available at reporting date, so they can be used to predict how likely the payment related to the claim i occurs after k years, that is, $\Delta_i = k$. Using a proper classification tool, formally represented by a function $p_k(\cdot)$ of the predictors, we will estimate $P(\Delta_i = k)$ for each admissible k:

$$\widehat{P}(\Delta_i = k) := p_k(x_{i1}, \ldots, x_{in}), \qquad \forall k \in [0, m]. \tag{1}$$

Now, we can still use the same predictors $x_{i1}, \ldots, x_{in}$ to estimate the payment amount due for the claim i; additionally, we will also use the information about Δ_i. Using a proper regression tool, formally represented by a function $f(x_{i1}, \ldots, x_{in}|\Delta_i)$ of the closing delay and the predictors, we will estimate the payment amount:

$$\widehat{C}_i(\Delta_i) := f(x_{i1}, \ldots, x_{in}|\Delta_i). \tag{2}$$

providing us with an estimation conditioned to Δ_i, which is unknown at reporting date. To overcome this limit, we first calculate the following estimations:

$$\widehat{C}_i(k) := f(x_{i1}, \ldots, x_{in}|k), \qquad \forall k \in [0, m] \tag{3}$$

and then estimate the payment amount for the claim i as follows:

$$\begin{aligned} \widehat{C}_i &:= \sum_{k=0}^{m} \widehat{P}(\Delta_i = k)\widehat{C}_i(k) = \sum_{k=0}^{m} p_k(x_{i1}, \ldots, x_{in}) f(x_{i1}, \ldots, x_{in}|k) = \\ &= \sum_{k=0}^{m} p_k(\mathbf{x}_i) f(\mathbf{x}_i|k) \end{aligned} \tag{4}$$

which is an unconditioned estimation.

In the next sections, we will present some of the machine learning techniques that can be used to estimate p_k and f. We will separate them because of their different nature: while p_k estimates the probabilities related to the

categorical variable Δ_i, f estimates the claim amount $C_i(k)$. The former refers to a classification problem, whereas the latter refers to a regression problem.

4. Fundamental machine learning tools

The closing delay estimation is a classification problem, that is, the goal is the prediction of how likely a claim will be closed - and the related amount payed - after k years from the reporting date. By contrast, the claim amount estimation is a regression problem since the target variable is numerical. Similarly, the closing delay estimation would have been treated as a regression problem too, if we had assumed k as a continuous time variable. However, this would complicate the model, being inconsistent with the traditional assumption of a discrete k with some upper limit m, just like in any triangle-based reserve calculation exercise.
Although we will distinguish between tools used for closing delay estimation and tools used for claims amount estimation as described at the end of Section 1, this distinction is not strict. In fact, most of the fundamental machine learning tools we will recall in the following subsections - generalized regression, naive Bayes, k-nearest neighbors, decision trees, and neural networks - are flexible enough to be used for both classification and regression problems.

4.1. Generalized regression

Regression models are by far the most used fitting tools in industry. They rely on a unique combination of theoretical solidity and practical interpretability, which makes them one of the favorite tools among actuaries. For the same reasons, regression models do not represent a powerful machine learning tool. Unfortunately, the assumption set they require causes a sort of rigidity that limits the ability of learning from data. Nonetheless, since we are going to use - among others - a generalized regression model to predict claim amounts (see Subsection 5.3), it is worth providing a brief description. Very generally, we can assume that the target variable y is a function of some predictors, that is, the explanatory variables of the i^{th} record:

$$y_i := \phi(x_{i1}, \dots, x_{in}), \qquad \forall i. \tag{5}$$

If we can rely on an algorithm that estimates ϕ by building a new regular function f, we will get an estimation of the target variable:

$$\widehat{y_i} = \phi(x_{i1}, \dots, x_{in}) + u_i = f(x_{i1}, \dots, x_{in}), \qquad \forall i \tag{6}$$

where u_i denotes the estimation error (notice the similarities between (3) and (6)). Quite intuitively, the function f will be somehow estimated in order to minimize the errors u_i, or, more precisely, a relevant combination of them representing the overall error of the model. However, the shape of f should be defined as an assumption. In *multiple linear regression*, for instance, its shape is linear:

$$\widehat{y}_i = \beta_0 + \sum_{j=1}^{n} x_{ij}\beta_j, \qquad \forall i \tag{7}$$

where β_j denotes the estimation parameter related to the j^{th} predictor. Remember that the linearity refers to the coefficients $\beta_0, \ldots, \beta_n$, not to the predictors $x_{i1}, \ldots, x_{in}$. In other terms, we can use the predictors as they appear in the dataset as well as any transformation or interaction of them, linear or nonlinear: in any case, the model will still be linear. Defining

$$\mathbf{y} := \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}, \mathbf{X} := \begin{pmatrix} 1 & x_{11} & \ldots & x_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & \ldots & x_{Nn} \end{pmatrix}, \beta := \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_n \end{pmatrix}, \mathbf{u} := \begin{pmatrix} u_1 \\ \vdots \\ u_N \end{pmatrix} \tag{8}$$

the model (7) may be also expressed in a matricial form:

$$\mathbf{y} = \mathbf{X}\beta + \mathbf{u} \tag{9}$$

and

$$\widehat{\mathbf{y}} = \mathbf{X}\widehat{\beta}. \tag{10}$$

The shape of f is not the only assumption of the model. Rather, the followings should hold too:

1. $r(\mathbf{X}) = n$
2. $E[\mathbf{u}] = \mathbf{0}$
3. $Var[\mathbf{u}] = \sigma^2\mathbf{I}_N$.

In other words, the rank of $\mathbf{X}$ is the maximum possible rank so that no redundant information is there (assumption 1.), the random variables of the estimation error are null on average (assumption 2.), and they are also independent and homoscedastic with variance σ^2 (assumption 3.). In particular, the assumption 2. directly implies the followings:

2a. $E[\mathbf{y}] = E[\mathbf{y}|\mathbf{X}] = \mathbf{X}\beta = \widehat{\mathbf{y}}$

2b. $Var(\mathbf{y}) = Var(\mathbf{y}|\mathbf{X}) = \sigma^2\mathbf{I}_N$.

Once the assumptions have been verified, the vector of parameters β can be estimated through the *least squares method*. It is based on the minimization of the sum of squared error u_i^2:

$$\begin{aligned} Q(\beta) &:= \sum_{i=1}^{N} u_i^2 = \sum_{i=1}^{N}(y_i - \widehat{y}_i)^2 = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta) = \\ &= \mathbf{y}^T\mathbf{y} - \beta^T\mathbf{X}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}\beta + \beta^T\mathbf{X}^T\mathbf{X}\beta = \\ &= \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}\beta + \beta^T\mathbf{X}^T\mathbf{X}\beta \end{aligned} \tag{11}$$

which is a quadratic function, thus its minimum is necessarily its only stationary point, that is, the solution of the following system:

$$\frac{\partial Q(\beta)}{\partial \beta} = \frac{\partial \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}\beta + \beta^T\mathbf{X}^T\mathbf{X}\beta}{\partial \beta} = -2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\beta = 0 \tag{12}$$

that is

$$\widehat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{13}$$

Using (10), we also get

$$\widehat{\mathbf{y}} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{14}$$

and using (9)

$$\widehat{\mathbf{u}} = \mathbf{y} - \widehat{\mathbf{y}} = \mathbf{y} - \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} = [\mathbf{I}_N - \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T]\mathbf{y}. \tag{15}$$

Actually, we got a model now, that is, we are able to calculate $\widehat{\mathbf{y}}$ by using $\widehat{\beta}$ in (10), as long as all the aforementioned assumptions hold. However, the traditional multiple linear regression includes a further, crucial assumption:

$$\mathbf{u} \sim N(\mathbf{0}, \sigma^2\mathbf{I}_N) \tag{16}$$

which implies

$$\mathbf{y} \sim N(\mathbf{X}\beta, \sigma^2\mathbf{I}_N). \tag{17}$$

Thanks to the normality of the vector $\mathbf{u}$ of the residuals, a whole range of relevant statistics can be deduced from the model, including parameter distributions, confidence interval, and so on. Among others, for instance, it can be easily proved that $\widehat{\beta}$ is normally distributed with mean β, and $\widehat{\mathbf{y}}$ is normally distributed with mean $\mathbf{y}$.

Once the model has been defined, we want an effective way to evaluate its performance. The theoretical framework helps us to define a rather intuitive but rigorous tool for performance evaluation. First, the *total deviance* is defined as

$$D_T(\mathbf{y}) := \sum_{i=1}^{N}(y_i - E[\mathbf{y}])^2 \tag{18}$$

and it represents a measure of the information contained into the data. The (18) may be further manipulate as follows:

$$\begin{aligned} D_T(\mathbf{y}) &= \sum_{i=1}^{N}(y_i - \widehat{y}_i + \widehat{y}_i - E[\mathbf{y}])^2 = \sum_{i=1}^{N}(y_i - \widehat{y}_i)^2 + \sum_{i=1}^{N}(\widehat{y}_i - E[\mathbf{y}])^2 =: \\ &=: D_E(\mathbf{y}) + D_R(\mathbf{y}) \end{aligned} \tag{19}$$

where the *explained deviance* $D_E(\mathbf{y})$ measures the information explained by the model, while the *residual deviance* $D_R(\mathbf{y})$ measures the residual information that the model could not detect. Naturally, the greater $D_E(\mathbf{y})$, the smaller $D_R(\mathbf{y})$, the better the model. In fact, the *coefficient of determination* - known as *R-squared* and defined by the deviance measures - is the most common performance evaluation tool for multiple linear regression:

$$R^2(\mathbf{y}) := \frac{D_E(\mathbf{y})}{D_T(\mathbf{y})} = 1 - \frac{D_R(\mathbf{y})}{D_T(\mathbf{y})} \tag{20}$$

which is a value in $[0, 1]$. Once again, the greater $D_E(\mathbf{y})$, the greater $R^2(\mathbf{y})$, the better the model, but the R^2 has the additional advantage of the normalization. In practice, it means that the R^2 makes possible a fair performance comparison among different models.
Using (17), the model definition (9) may be written differently:

$$\mathbf{y} = E[\mathbf{y}] + \mathbf{u} \tag{21}$$

which is a direct consequence of the normality assumed. However, if we relax that assumption, it can be more generally assumed that there is a regular (i.e., invertible and derivable) *link function* g - different to the identity function - mapping $E[\mathbf{y}]$ to $\mathbf{X}\beta$:

$$g(E[\mathbf{y}]) = \mathbf{X}\beta \tag{22}$$

so that, using (21),

$$\mathbf{y} = g^{-1}(\mathbf{X}\beta) + \mathbf{u}. \tag{23}$$

The main reason why such a *generalized regression* is very useful in practice is that g maps from some subset $X \subseteq \mathbb{R}$ to $\mathbb{R}$, that is, g^{-1} transforms the linear predictor $\mathbf{X}\beta \in \mathbb{R}$ to the target variable prediction in X. In fact, generalized regression is able to handle target variables defined in a specific subset of $\mathbb{R}$. This is not possible in multiple linear regression, unless we operate a proper transformation of the target variable itself (this is sometimes enough, but it introduces *transformation bias* into the model). For instance, if the target variable represents a positive amount, the quickest ways to use regression are

- convert amounts to logaritmic amounts, predict the latter through multiple linear regression, and convert the predictions back by using the exponential function;
- choose a link function $g : (0, +\infty) \to \mathbb{R}$, and predict the amounts through the related generalized regression.

Actually, the choice of the link function is not direct, rather it is a consequence of the distribution we assume for the target variable. This is possible since generalized regression assumes that such a distribution belongs to the exponential family, whose density is:

$$f_e(y_i; \theta_i, \phi) := e^{\frac{y_i\theta_i - c(\theta_i)}{\phi} + h(y_i, \phi)} \tag{24}$$

where θ_i denotes the *canonical parameter* varying by observation, while ϕ denotes the *dispersion parameter* that is constant for all observations. In other words, the distribution constraint is not completely eliminated, rather it is "generalized" to a wider range of distributions - a distribution family.
When it comes with prediction of amounts, whose distributions are often nonnegative and heavy-tailed, a common choice is the Gamma distribution, defined by a *shape parameter* $\varphi > 0$ and a *scale parameter* $\vartheta_i > 0$:

$$f_\Gamma(y_i; \vartheta_i, \varphi) := \frac{y_i^{\varphi-1} e^{-\frac{y_i}{\vartheta_i}}}{\vartheta_i^{\varphi}\Gamma(\varphi)}. \tag{25}$$

which can be easily rearranged as follows:

$$f_\Gamma(y_i; \vartheta_i, \varphi) = e^{-\frac{y_i}{\vartheta_i} - \varphi \ln \vartheta_i + (\varphi-1)\ln y_i - \ln\Gamma(\varphi)}. \tag{26}$$

If we define $\theta_i := -\frac{1}{\varphi\vartheta_i}$ and $\phi := \frac{1}{\varphi}$, then

$$f_\Gamma(y_i; \vartheta_i, \varphi) \;=\; e^{\frac{y_i\theta_i}{\phi} - \frac{1}{\phi}\ln\left(-\frac{1}{\varphi\theta_i}\right) + \left(\frac{1}{\phi}-1\right)\ln y_i - \ln\Gamma\left(\frac{1}{\phi}\right)} =$$

$$= e^{\frac{y_i\theta_i}{\phi}-\frac{1}{\phi}\ln\left(-\frac{1}{\theta_i}\right)+\left(\frac{1}{\phi}-1\right)\ln y_i+\varphi\ln\varphi-\ln\Gamma\left(\frac{1}{\phi}\right)}=$$
$$= e^{\frac{y_i\theta_i-\ln\left(-\frac{1}{\theta_i}\right)}{\phi}+\left(\frac{1}{\phi}-1\right)\ln y_i-\frac{1}{\phi}\ln\phi-\ln\Gamma\left(\frac{1}{\phi}\right)} \quad (27)$$

which belongs to the exponential family in (24) with

$$c(\theta_i) := \ln\Big(-\frac{1}{\theta_i}\Big), \qquad h(y_i,\phi) := \Big(\frac{1}{\phi}-1\Big)\ln y_i - \frac{1}{\phi}\ln\phi - \ln\Gamma\Big(\frac{1}{\phi}\Big). \quad (28)$$

This is important because it implies that

$$E[y_i] = \frac{dc(\theta_i)}{d\theta_i} = \frac{d}{d\theta_i}\ln\Big(-\frac{1}{\theta_i}\Big) = -\frac{d}{d\theta_i}\ln\theta_i = -\frac{1}{\theta_i} = \varphi\vartheta_i \quad (29)$$

$$Var(y_i) = \phi\frac{d^2c(\theta_i)}{d\theta_i^2} = -\phi\frac{d}{d\theta_i}\frac{1}{\theta_i} = \frac{\phi}{\theta_i^2} = \varphi\vartheta_i^2. \quad (30)$$

All in all, how should we define the link function starting from these considerations? First, notice that the normal distribution belongs to the exponential family too, because

$$f_N(y_i;\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(y_i-\mu)^2}{2\sigma^2}} = \ldots = e^{\frac{y_i\mu-\frac{\mu^2}{2}}{\sigma^2}-\frac{y_i^2}{2\sigma^2}-\ln\sqrt{2\pi\sigma^2}} \quad (31)$$

thus we should define $\theta_i := \mu$ and $c(\theta_i) := c(\mu) = \frac{\mu^2}{2}$, so that

$$E[y_i] = \frac{dc(\mu)}{d\mu} = \mu = \theta_i. \quad (32)$$

Actually, if the target variable is normally distributed, we will be back to the multiple linear regression. In such a case, we may use (21) to write:

$$\mathbf{y} = E[\mathbf{y}] + \mathbf{u} = \theta + \mathbf{u} \quad (33)$$

so a generalized regression model might be simply redefined as

$$\mathbf{y} = g^{-1}(\theta) + \mathbf{u}. \quad (34)$$

In fact, the link function is now the function that maps $E[\mathbf{y}]$ to the vector θ of the canonical parameters:

$$g(E[\mathbf{y}]) = \theta \quad (35)$$

but if the target variable is distributed as a Gamma, we also know that $E[y_i] = \varphi\vartheta_i$ and $\theta_i = -\frac{1}{\varphi\vartheta_i}$, so

$$g(\varphi\vartheta_i) = -\frac{1}{\varphi\vartheta_i} \tag{36}$$

which means that g may be simply defined as $g(x) := -\frac{1}{x}$. This is a somewhat natural choice, coming directly from the theory, thus it is called *canonical link function*. For generalized regression models, it is always possible to define g in a canonical way, that is, imposing (35), and this choice implies a range of desirable features in our model. However, others link functions might be rather used, and sometimes there are good reasons to.

Unfortunately, parameters of generalized regression models cannot be explicitly calculated as in the multiple linear regression. In this case, we can only look for maximum likelihood estimates by using numerical methods.

At the beginning of this subsection, we implicitly assumed to know the range of explanatory variables $x_1, \ldots, x_n$. Of course, we know the explanatory variables in the dataset, but how should we select them as $x_1, \ldots, x_n$? Because of multicollinearity among potential explanatory variables, we cannot simply run the regression on all of them, and then select only the most significant ones based on their p-values. Rather, we should somehow select different sets of explanatory variables and run the related regression: the model with the highest R^2 will be selected. The different sets of explanatory variables depend on the algorithm used to select them. There are mainly three popular iterative search algorithms.

In *forward selection*, we start with no predictors, and then add them one by one. Each added predictor is that (among all predictors) that has the larges contribution to R^2 on top of the predictors that are already in it. The algorithm stops when the contribution of additional predictors is not statistically significant.

In *backward selection*, we start with all predictors, and then eliminate the least useful one at each step according to statistical significance. The algorithm stops when all the remaining predictors have significant contributions.

Finally, *stepwise selection* is like forward selection except that at each step we consider dropping predictors that are no longer statistically significant, as in backward selection. In the Subsection 5.3, our data will be regressed through stepwise selection.

To finally conclude the section, some practical remarks are to be highlighted.

Whatever the specific regression model, its assumptions make it theoretically strong, but VERY weak from a practical perspective. Actually, We may condensate them with the following, practical premises:

- no allowance for nonlinear relationships
- no allowance for dependencies among predictors
- no allowance for outliers.

They somewhat simplify the original assumptions, but give a fair idea of what those assumptions really mean, i.e. we can hardly expect good performance from regression methods when information is far to be regular. Data is generally affected by missing values, redundancies, correlations, heavy tails, asymmetries, nonlinearities, and any other kind of distortion. In fact, as we will discuss in Subsection 5.1, our data is no exception.

4.2. Naive Bayes

This tool is surely one of the easiest machine learning techniques. It is a transformation of the well-known *Bayesian classifier*. Given the values for the predictor vector $\mathbf{x}_i$ related to the claim i, the Bayes' theorem returns the (exact) Bayesian classifier:

$$\begin{aligned} p_k(\mathbf{x}_i) &= P(\Delta_i = k|\mathbf{x} = \mathbf{x}_i) = \\ &= \frac{P(\Delta_i = k)P(\mathbf{x} = \mathbf{x}_i|\Delta_i = k)}{\sum_{h=0}^{m} P(\Delta_i = h)P(\mathbf{x} = \mathbf{x}_i|\Delta_i = h)}, \quad \forall k = 1, \ldots, m. \end{aligned} \tag{37}$$

This approach is theoretically correct, but presents a fundamental limit. The predictor in (37) implicitly assumes that we can find a sufficient number of records in the sample sharing the same vector $\mathbf{x}_i$. Perhaps, this is reasonable when there are VERY few predictors in the dataset, otherwise it is completely impracticable.

A straight modification to (37) represents a very simple solution to such a problem. If we give up to the assumption that the best probability estimation is solely returned by those records matching the record to be classified, we will be able to use the whole dataset for the estimation. As a consequence of this assumption, the classifier in (37) changes as follows:

$$\begin{aligned} p_k(\mathbf{x}_i) &= P(\Delta_i = k|\mathbf{x} = \mathbf{x}_i) = \frac{P(\Delta_i = k)P(\mathbf{x} = \mathbf{x}_i|\Delta_i = k)}{\sum_{h=0}^{m} P(\Delta_i = h)P(\mathbf{x} = \mathbf{x}_i|\Delta_i = h)} = \\ &= \frac{P(\Delta_i = k)\prod_{j=1}^{n} P(x_j = x_{ij}|\Delta_i = k)}{\sum_{h=0}^{m} P(\Delta_i = h)\prod_{j=1}^{n} P(x_j = x_{ij}|\Delta_i = h)}, \quad \forall k = 1, \ldots, m \end{aligned} \tag{38}$$

which is, indeed, the so-called *naive Bayes classifier*.
This approach is extremely simple to understand and to use. Moreover, it presents no computational issues: it is just a formula to apply as it is, rather than a complex algorithm. Unfortunately, this simplicity hides a major drawback, that is, the assumption of stochastic independence among predictors. In effect, that is exactly the assumption which allows us to move from the Bayesian classifier in (37) to the naive Bayes classifier in (38). Comparing the two formulas, we can easily notice this point. However, this is seldom the case.
Moreover, some studies (see, for instance, [10]) point out the strengths of such a classifier when it comes with record ranking, but also its weaknesses when it comes with probability estimation. In practice, naive Bayes classifier's probability estimation can be very biased by the assumption of stochastic independence. If the same bias is shared by each record, the classification can be still good, but of course we cannot rely on the estimated probability. This is the reason why this classifier often outperforms more sophisticated classifiers as a classification tool, and it is still widely used in several fields (see, for instance, the spam filtering case in [14]).
A last drawback is quite relevant. Actually, what if some predictor category is not present in the training dataset (for instance, it could be very rare)? In this case, $P(x_j = x_{ij}|\Delta_i = k) = 0$ for some j and k, thus $p_k(\mathbf{x}_i) = 0$, which is clearly wrong. In fact, the naive Bayes classifier works well if each and every category is well represented. That has a two-fold meaning. First, the training dataset should be large enough to well represent each and every category. Second, more importantly, numerical predictors are not admissible at all by definition, that is, we can use it for closing delay estimation (which is a classification problem), but not for claim amount estimation (which is a regression model). For the latter, we need the methods described in the next subsections.

4.3. K-nearest neighbors

The idea behind the k-nearest neighbors algorithm is very intuitive, but it still guarantees a high level of adaptability to data. To score a new record, the method relies on finding the most "similar" records - the so-called "neighbors" - in the training dataset. In fact, this is a pure nonparametric method: no assumption needs to be established, no parameter needs to be estimated, no functional form needs to be assumed.
The sole issue regards the choice of a measure to calculate the "distance"

between two records, that is, their grade of similarity. The most popular measure is the *Euclidean distance*: given the vectors of predictors for two different records, say $\mathbf{x}_i = (x_{i1}, \ldots, x_{in})$ and $\mathbf{x}_j = (x_{j1}, \ldots, x_{jn})$, the Euclidean distance between them is defined as follows:

$$d(\mathbf{x}_i, \mathbf{x}_j) := \sqrt{\sum_{k=1}^{n} (x_{ik} - x_{jk})^2}, \qquad \forall i, j. \tag{39}$$

It is worth noting that this definition of distance would give much more weight to higher scales than lower scales, so all the predictors should be first standardized before computing the (39). Otherwise, the k-nearest neighbors algorithm could result in extremely biased predictions.

Once we have chosen a distance measure, we can calculate the distance between any pair of records, but we still need a rule to score new records. The simplest rule is: a new record is classified in the same category of its closest neighbor. Therefore, given the predictors of such a record, we will compute all the distances between it and the records in the training dataset. Among them, pick up that with the smallest distance to the new record: its category will be assigned to the new record itself. In effect, we have just applied a 1-nearest neighbors algorithm.

Nonetheless, the approach might be easily generalized to any number k of neighbors. Instead of picking up the closest record only, pick up the k closest records, and assign the majority class among them to the new record. In practice, the usage of more neighbors tends to reduce misclassification error. If we use one neighbor only, it could be the case that a record is the closest one to the new record only by chance: there could be noise there, rather than information. To some extent, the greater the k, the lower the error, the greater the predictive power. On the other hand, however, if k is too high, we will miss out on the method's ability to capture the local structure in the data, that is, we will ignore information.

Anyway, the choice of k is straightforward. First, choose an upper limit K for k. Then, score each record in the *validation* dataset using the closest record in the training dataset (1-nearest neighbors algorithm), and calculate the validation error ϵ_1. Likewise, score each record in the *validation* dataset using the two closest records in the training dataset (2-nearest neighbors algorithm), and calculate the validation error ϵ_2. Repeat the process until the K-nearest neighbors algorithm, that is, the last admissible algorithm according to our upper limit. At the end, we will get the validation errors

$\epsilon_1, \ldots, \epsilon_K$. Finally, pick up the k related to the smallest validation error, say k_{min}, and use the k_{min}-nearest neighbors algorithm for scoring.
Remember that ϵ_k denotes the validation error of k-nearest neighbors algorithm. Correspondingly, we could compute the training error as well, that is, the error committed when scoring each record in the *training* (rather than validation) dataset using the k closest records in the training dataset itself. The training error, however, cannot be used for scoring because the smallest training error always relates to the 1-nearest neighbors algorithm: whichever the training record to be scored, its closest training record is obviously the record itself!
Of course, the k-nearest neighbors classifier is as simple as the naive Bayes classifier. As already discussed at the beginning of this section, its main advantage is the lack of parametric assumptions. Unfortunately, there are some drawbacks. For instance, from a computational perspective, this algorithm can be very expensive. Sometimes, data scientists try to reduce predictors by using other, less expensive machine learning tools before applying the k-nearest neighbors algorithm to their datasets. Dimension reduction my be performed, among others, by classification trees, which is the topic of the next subsection.

4.4. Classification and regression trees (CARTs)

Decision trees were used as a machine learning tool in [3] for the very first time to segment a population by splitting up the dataset through binary rules. The algorithm is now referred to as *classification tree*. Since one of our goals is the categorical classification among the closing delay classes, this tool is a good candidate for us. By contrast, we should properly adapt it to regression problems, if we want to use it to predict the claim amount. In this case, we call it a *regression tree*. However, given that the basic algorithm does not change, we can always refer to *classification and regression trees*, or CARTs.
CARTs are based on recursive partitioning, which divides up the multidimensional space (that is, the dataset) of the explanatory variables into non-overlapping multidimensional rectangles. This division is accomplished recursively, i.e. operating on the results of the prior divisions. An example of CART is in Figure 3. First, one of the explanatory variables is selected, say $x_{k(0)}$ (the first *node* of the tree, so-called *root*), and a value of $x_{k(0)}$, say $s_{k(0)}$), is chosen to split the n-dimensional space into two parts: one part contains all the records with $x_{k(0)} \leq s_{k(0)}$, say $n(1,1)$ records, while the other with all

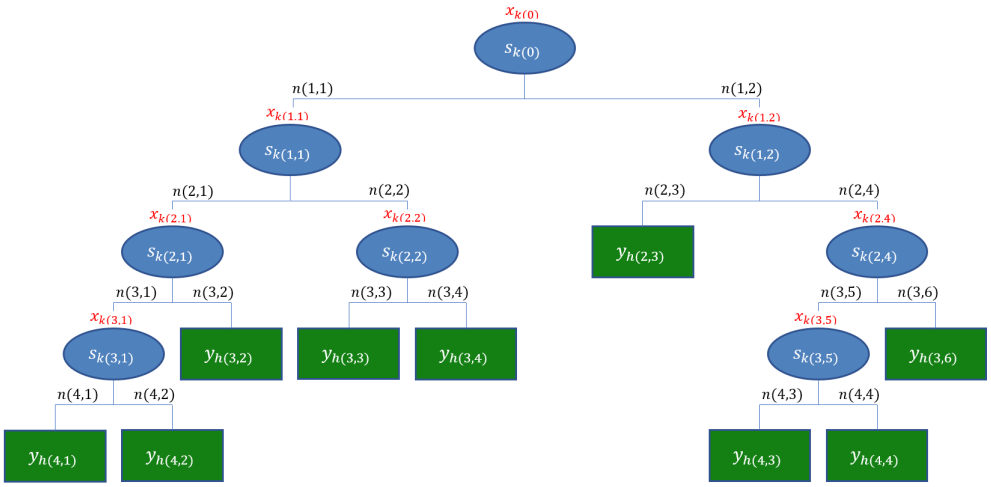

Fig. 3: Example of CART produced by recursive partitioning

the records with $x_{k(0)} > s_{k(0)}$, say $n(1,2)$ records. The two sub-datasets represent the first level of the tree. Let's consider one of them: it could be either *pure* - i.e. it contains only records sharing the same value of the independent variable - or *impure*. In the first case, no further split is possible, so the sub-dataset will represent a *leaf* of the tree. In the second case, other splits are possible, so the sub-dataset will represent another node of the tree. In Figure 3, leaves are denoted by green rectangles, while nodes are denoted by blue circles. Unless both of the sub-datasets generated by the root node are pure, one of them (at least) will be divided in a similar manner by choosing a variable again, and a split value for the variable. In Figure 3, where both of the sub-datasets of the first level are impure, they represent the two nodes of the first level, and the initial dataset is further partitioned into four regions:

- the first one contains the $n(2,1)$ records with $x_{k(1,1)} \leq s_{k(1,1)}$, and it will represent a node for the next level using $x_{k(2,1)}$ as a splitting variable
- the second one contains the $n(2,2)$ records with $x_{k(1,1)} > s_{k(1,1)}$, and it will represent a node for the next level using $x_{k(2,2)}$ as a splitting variable
- the third one contains the $n(2,3)$ records with $x_{k(1,2)} \leq s_{k(1,2)}$, and it will represent a leaf for the next level containing all the records with target variable equal to $y_{h(2,3)}$

- the fourth one contains the $n(2,4)$ records with $x_{k(1,2)} > s_{k(1,2)}$, and it will represent a node for the next level using $x_{k(2,4)}$ as a splitting variable.

Since some splits are still possible, the recursive partitioning goes on, getting smaller and smaller sub-datasets, either nodes or leaves. Sooner or later, we will have divided the whole dataset up into pure sub-datasets (of course, this is not always possible, as there may be records that belong to different classes but have exactly the same values for everyone of the predictor variables).
In the case of closing delay estimation, the dataset will be partitioned into sub-datasets which contain either claims closed in the reporting year (closing delay 0), or claims closed the year after (closing delay 1), or claims closed two years after (closing delay 2), ..., or claims closed m years after (closing delay m). In fact, the classification tree resulting from recursive partitioning is a *pure* tree: all the closing delay categories are perfectly separated.
The main problem of recursive partitioning is the choice of the splitting rule node by node, that is, the choice of $x_{k(\cdot,\cdot)}$ and $s_{k(\cdot,\cdot)}$ at each step of the algorithm. Assume to define an *impurity function* $i(A)$ as an impurity measure of some rectangle A, or its related node. A specific splitting rule on A results in two sub-rectangles A_L and A_R, which are generally impure, that is, $i(A_L)$ and $i(A_R)$ are both nonzero. Intuitively, we want to choose the splitting rule in order to minimize some combination of $i(A_L)$ and $i(A_R)$. The most natural choice is the function

$$I(A_L, A_R) := \frac{|A_L|}{|A|} i(A_L) + \frac{|A_R|}{|A|} i(A_R) \tag{40}$$

which is the average of the two impurity measures, weighted by the number of observations in each rectangle. By comparing the reduction in $I(A_L, A_R)$ across all possible splits in all possible predictors, the next split is chosen.
What about the impurity function i? In our application and in most of them, one uses the *Gini index* (as defined in [14]):

$$i(A) := 1 - \sum_{k=0}^{m} p_k^2(A) \tag{41}$$

where p_k is the proportion of records in rectangle A that are closed after k year from the reporting. However, other impurity measures are also widely

used, for example the *entropy index* (as defined in [14]):

$$E(A) := -\sum_{k=0}^{m} p_k(A) \log_2[p_k(A)]. \tag{42}$$

All in all, so far the algorithm is quite intuitive as well as its application in classifying new records. For instance a new observation, whose explanatory values are known, will be dropped down the tree until it reaches a leaf. So the new observation will be simply classified on the base of the specific leaf's classification.

As discussed at the beginning of the section, the algorithm can be easily adapted to predict numerical variables. We only need some changes. First, the response value assigned to a record in a leaf is determined by the average of the response variable among the records in that leaf (by contrast, in a classification tree, it is determined by one of the possible category of the response variable). Second, given that we cannot use discrete measure of impurity such as the Gini index and the entropy index, the typical impurity measure used in regression problems is the sum of squared deviations from the mean, that is, the sum of squared errors.

By definition, recursive partitioning produces a tree which classify the records without errors. Actually, we used a training dataset to train the classification tree, which perfectly predict closing delay on that dataset. But what if we use the same tree on the validation dataset? In general, the predicted values on the validation dataset will result in a positive *misclassification error*. In fact, the error cannot be zero on datasets other than the training dataset itself. However, our full classification tree has a major drawback represented by Figure 4. As it usually happens through the first splits on the validation dataset, the full tree can still guarantee comparable misclassification errors on the two datasets. However, as the number of splits increases, the full tree starts *overfitting* the validation data: since it fully reflects the training dataset without distinguishing between "signal" and "noise", the noisy component cause too high misclassification error in the validation dataset. Indeed, the typical consequence of overfitting is that, after some number of splits, the misclassification error on the validation dataset stops decreasing and starts increasing (in Figure 4, it occurs after ten splits). In the first ten splits both the training and validation misclassification errors decrease, but thereafter the full tree overfits the validation data.

Overfitting prevent us from using the full tree for predicting purposes, so we

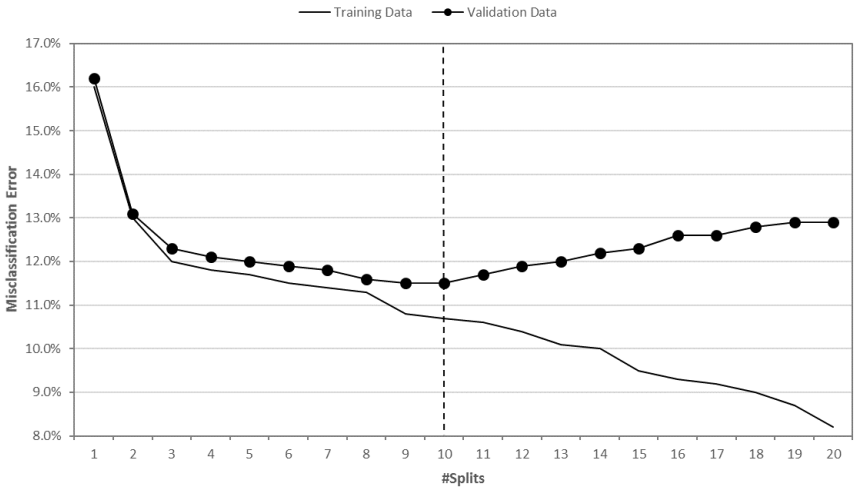

Fig. 4: Training Misclassification Error vs. Validation Misclassification Error

need to choose another tree, that is, some subtree of the full tree. There are several criteria to do that, but two major categories of methods can be distinguished:

- forward stopping-tree methods
- backward pruning-tree methods.

Some empirical forward methods can be easily implemented by setting conditions to the tree characteristics such as maximum number of splits, minimum number of records in a node, and minimum reduction in impurity. Unfortunately, these approaches are solely based on the tree complexity, rather than its predictive power.

Among the forward methods, the most natural choice is suggested by the Figure 4 itself, i.e. we can simply use the tree consisting of the the first n splits that do not induce overfitting ($n = 10$ in Figure 4). In other words, we let the full tree grow until the first step leading to a validation error higher than the previous step. If we have new observations to classify, they will be dropped down this subtree until they reach a leaf.

More complex methods have been developed as well, for instance, the so-called *chi-squared automatic in-training data* (CHAID). At each node, the algorithm select the predictor with the strongest association to the target

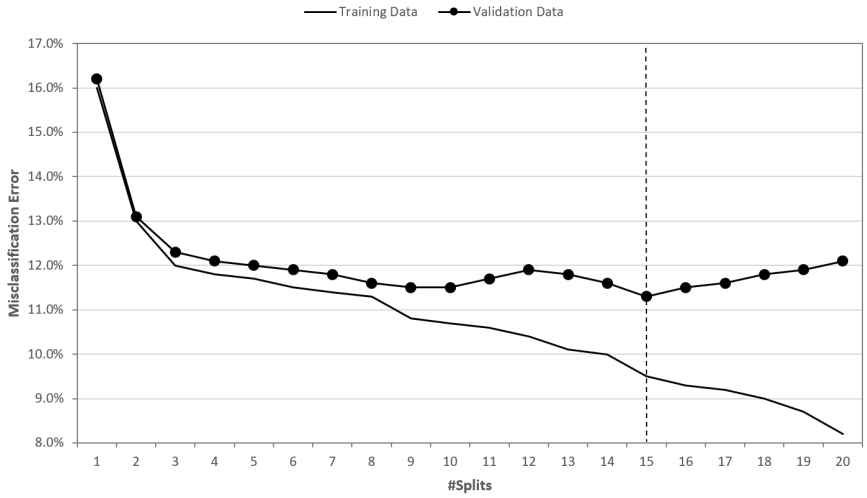

Fig. 5: Training Misclassification Error vs. Validation Misclassification Error

variable, measured by the p-value of the chi-squared test of independence. If such a p-value is low enough, the split of the node significantly improves its purity, so the algorithm will carry it out, and the growth of the tree goes on. Otherwise, the growth is stopped.

The alternative to stopping the growth of the tree is represented by pruning the tree, that is, "climb" the training full tree and "chop" the weakest branches until some conditions are met. Intuitively, pruning the tree is more computationally expensive, because we have to build the full tree anyway, and then work on that further. However, it has been proven to be more successful too, and it is not so surprising. In effect, the tree is pruned considering more information: not only what-if-we-keep-the-smaller-tree information, but even what-if-we-take-the-bigger-tree information.

A simple backward approach follows the idea of picking the tree with the smallest validation error, just like in one of the forward methods. However, the full tree is still built until the last leaf, and then we choose the subtree leading to the smallest validation error. Notice that we would still pick the subtree after 10 splits in Figure 4, but that is not always the case (see, for instance, Figure 5).

However, choosing the subtree according to the smallest validation error only means we completely ignore the complexity of the tree. For instance, take a look at Figure 5 once again. We are picking the 15-split subtree since it

leads to the smallest error, but we get a rather small error with the 10-split subtree as well. Actually, we accept five splits more - a relevant increase in complexity - for a little decrease in error. Somehow, it does not seem to be the best choice. To consider this issue in pruning the tree, we may use the so-called *cost complexity criterion*as described in [3]. For a tree with L leaves and training error ϵ, the cost complexity of a tree T is defined as follows:

$$\gamma(T) := \alpha L(T) + \epsilon(T) \tag{43}$$

where α denotes a (nonnegative) penalty factor for the tree size. Notice that, if $\alpha = 0$, there is no penalty for the tree size, and the best tree is simply the full tree itself. On the other hand, the greater the α, the greater the relevance of the tree size in pruning the tree. If α is great enough, the training error is no longer relevant for the algorithm, and the tree is pruned until the very first node, that is, the root. The algorithm therefore starts with the n-level full tree, and compares its γ with the γs of all the possible $n-1$-level subtrees. The α gets increased little by little, until the γ of the full tree exceeds that of one of those subtrees. Such a subtree is considered as the best of its level, and the same procedure is repeated starting from it. In fact, the algorithm finds the best subtree of each size based on the cost complexity criterion: the lower γ, the better the subtree. In practice, we get a sequence of "best subtrees" for their sizes, based on the training data only. Finally, the so-called *best pruned tree* used for scoring will be the subtree among them leading to the smallest validation error.

All in all, the full tree is useless for scoring purposes, rather it is just the formal result of recursive partitioning. What is really useful for prediction is the subtree extracted by using one of the several algorithms to stop growing or pruning the full tree.

So far we described the fundamentals of CARTs. One of the reasons for their popularity is that they are adaptable to a wide variety of applications, and have been successfully used in many situations. In particular, if there is a highly non-linear and complex relationship to describe, decision trees may outperform regression models. Furthermore, CARTs do not require massive data preparation, that is, they can handle non-standardized data, categorical data, missing data, outliers, and so on. By contrast, we should standardize the variables and take the natural logarithm of some numerical variables before running standard linear regression, logistic regression, or even the k-nearest neighbors method described in Subsection 4.3. Finally, trees provide easily understandable classification rules (at least if they are not too large),

even easier than in regression.
An important advantage of CARTs is that no further selection algorithm is necessary. As opposed to parametric regression methods, the process itself selects the most relevant explanatory variables. We simply let the machine learning tool run on the whole dataset, and the resulting tree will include only some of the explanatory variables, which are the most significant on the base of the impurity measure used to split the dataset.

4.5. Neural networks

Algorithms like k-nearest neighbors method or CARTs have the major advantage of non-dependence on underlying structures or parameters. This is what makes them perfect to handle - and, to some extent, discover - unknown relationships in datasets. On the other hand, this flexibility makes them extremely weak when data is not enough to train them properly. *Neural networks* are trained by data too, but assume an underlying function that is generally much more complex than the typical functions used in regression. To some extent, we might consider neural networks as a trade-off between pure nonparametric methods and traditional regressions.
A number of successful applications contributed to the great spread of the neural network concept, including some relevant financial topics (see [16]) such as bankruptcy prediction, asset allocation, fraud detection, and customer relationship management.
An example of neural network is in Figure 6. Actually, each neural network has its own structure based on *neurons*, but all of them share the same fundamental features:

- one *input layer* consisting of a number of neurons - one for each predictor
- one or more *hidden layers*, each of them consisting of its own neurons
- one *output layer* consisting of a number of neurons which returns the predictions.

In particular, notice that the output layer consists of one neuron only if the target variable is binary (i.e., the neural network predicts one probability only) or numerical (i.e., the neural network predicts one numerical value only).
Another important remark regards the number of hidden layers. One issue

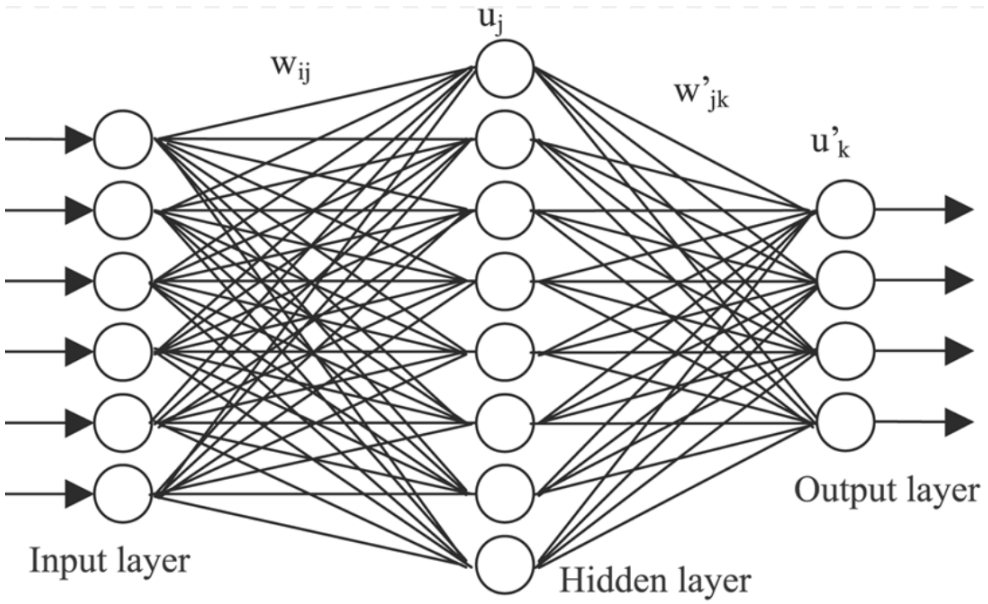

Fig. 6: Example of neural network

within this subject on which there is a consensus is the performance difference from adding additional hidden layers: the situations in which performance improves with a second (or third, etc.) hidden layer are very few. One hidden layer is sufficient for the large majority of problems, as stated in [14] as well:

> *The most popular choice for the number of hidden layers is one. A single hidden layer is usually sufficient to capture even very complex relationships between the predictors.*

Therefore, we will solely consider one-hidden-layer neural networks, just like the example in Figure 6.
Furthermore, each input neuron is connected to each hidden neuron, and each hidden neuron is connected to each output neuron.
Neural network training is based on the calibration of the following parameters (see Figure 6):

- the weight parameters w_{ij}, one for each connection from the input layer to the hidden layer
- the bias parameters u_j, one for each hidden neuron

- the weight parameters w'_{jk}, one for each connection from the hidden layer to the output layer
- the bias parameters u'_k, one for each output neuron.

The input layer, which knows the raw data of the predictors x_i, communicate it to the hidden layer. Such an information, however, is weighted by the connection itself, that is, by the related weight parameter. In other words, the j^{th} hidden neuron receives the information $w_{ij}x_i$. Additionally, given that it receives data from each and every input neuron (i.e., predictor) at the same time, it aggregates information through some *hidden activation function* $f(\mathbf{x}, \mathbf{w}_j, u_j)$ of the predictor values, the weights, and the bias. The most popular hidden activation function is the traditional weighted average:

$$H_j := f(\mathbf{x}, \mathbf{w}_j, u_j) = u_j + \sum_i w_{ij}x_i, \qquad \forall j. \tag{44}$$

Likewise, the output neurons receive weighted information from each hidden neuron. In particular, the k^{th} output neuron receives the value $w'_{jk}H_j$ from the j^{th} hidden neuron. In fact, the k^{th} output neuron receives information from each and every hidden neuron, so it will manipulate it too. More specifically, the output neurons use an *output activation function* $g(\mathbf{H}, \mathbf{w}'_k, u'_k)$ of their own parameters. Of course, g is generally different to f, for instance, g is often defined as a linear transformation of the logistic function:

$$\begin{aligned} O_k &:= g(\mathbf{H}, \mathbf{w}'_k, u'_k) = logit\left(u'_k + \sum_j w'_{jk}H_j\right) = \\ &= \frac{1}{1 + e^{-u'_k - \sum_j w'_{jk}H_j}}, \qquad \forall k. \end{aligned} \tag{45}$$

Another common output activation function in artificial neural network is the hyperbolic tangent function:

$$\begin{aligned} O_k &:= g(\mathbf{H}, \mathbf{w}'_k, u'_k) = tanh\left(u'_k + \sum_j w'_{jk}H_j\right) = \\ &= \frac{e^{u'_k + \sum_j w'_{jk}H_j} - e^{-u'_k - \sum_j w'_{jk}H_j}}{e^{u'_k + \sum_j w'_{jk}H_j} + e^{-u'_k - \sum_j w'_{jk}H_j}}, \qquad \forall k. \end{aligned} \tag{46}$$

Several other options are still possible, but O_k always represents the prediction provided by the k^{th} output neuron. If the target variable is categorical,

O_k equals the probability of the k^{th} category related to a specific record.
There is still an interesting observation to do. Assume that the target variable is binary, that is, the neural network gets only one output neuron that predicts the probability of "success", whereas the number of hidden neuron is the same as the number of predictors. Moreover, assume the following parameters within the hidden layer:

$$u_i = 0, \qquad w_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}, \qquad \forall i, j \tag{47}$$

which actually means

$$H_j \equiv H_i = x_i, \qquad \forall j. \tag{48}$$

Using (45) for the single output neuron, we get

$$\widehat{P}(1) = O_1 := g(\mathbf{x}, \mathbf{w}'_k, u'_k) = logit\left(u'_1 + \sum_i w'_{i1} x_i\right) = \frac{1}{1 + e^{-u'_1 - \sum_i w'_{i1} x_i}} \tag{49}$$

which is equivalent to the functional form of a logistic regression's prediction. However, we are going to see why it does not mean at all that this peculiar neural network equals the logistic regression in terms of predicted probabilities.
In fact, the main difference between neural networks and regressions lies in the way parameters are estimated. While regression methods rely on predetermined target functions to minimize or maximize, neural networks actually "learn" from data. Estimating biases and weights is a consequence of such a learning process, whichever its algorithm is. Anyway, the most popular one is the so-called *back propagation*.
We denoted the prediction from the k^{th} node by O_k. Moreover, let's define the prediction error related to the first record as

$$\epsilon_{1k} := O_{1k}(1 - O_{1k})(y_1 - O_{1k}) \tag{50}$$

where y_1 equals the actual value of the target variable. Given a global *learning rate* $\lambda \in (0, 1)$ and some initialization values for the parameters of the network, weights and biases are updated as follows:

$$w_{ij} \longrightarrow w_{ij} + \lambda \epsilon_{1k}, \quad \forall i, j \tag{51}$$

$$u_j \longrightarrow u_j + \lambda \epsilon_{1k}, \quad \forall j \tag{52}$$

$$w'_{jk} \longrightarrow w'_{jk} + \lambda \epsilon_{1k}, \quad \forall j \tag{53}$$

$$u'_k \longrightarrow u'_k + \lambda \epsilon_{1k}. \tag{54}$$

After that, the second record goes through the network to get its own estimations O_{2k}, then the error is computed:

$$\epsilon_{2k} := O_{2k}(1 - O_{2k})(y_2 - O_{2k}) \tag{55}$$

and the parameters are updated once again starting from those of the previous step:

$$\begin{aligned} w_{ij} &\longrightarrow w_{ij} + \lambda\epsilon_{2k}, \quad \forall i, j && (56)\\ u_j &\longrightarrow u_j + \lambda\epsilon_{2k}, \quad \forall j && (57)\\ w'_{jk} &\longrightarrow w'_{jk} + \lambda\epsilon_{2k}, \quad \forall j && (58)\\ u'_k &\longrightarrow u'_k + \lambda\epsilon_{2k}. && (59) \end{aligned}$$

The computation is repeated for all the records, all the way through the training dataset: after the last observation, the first *epoch* is completed. Generally, a number of epochs is predefined, that is, the records are estimated several times until some tolerance on the significance of parameter updating is broken, or some threshold on the training error is finally met.
What we have just described is called *case updating*, but it is not the sole option. For instance, in *batch updating*, the whole training dataset is run through the network before each updating takes place. As a consequence, the parameters are updated on the base of the overall training error, or its average:

$$\bar{\epsilon}_k := \frac{1}{N}\sum_i \epsilon_{ik} \tag{60}$$

and

$$\begin{aligned} w_{ij} &\longrightarrow w_{ij} + \lambda\bar{\epsilon}_k, \quad \forall i, j && (61)\\ u_j &\longrightarrow u_j + \lambda\bar{\epsilon}_k, \quad \forall j && (62)\\ w'_{jk} &\longrightarrow w'_{jk} + \lambda\bar{\epsilon}_k, \quad \forall j && (63)\\ u'_k &\longrightarrow u'_k + \lambda\bar{\epsilon}_k. && (64) \end{aligned}$$

In practice, case updating tends to be more accurate than batch updating, but it is also more computationally expensive, given that the parameters are updated N times per epoch rather than only once.
Neural networks can be very powerful, if their architecture is significant, that is, the number of hidden layers and hidden neurons is "right" - whatever it means. There are algorithms that automatically select this features, but

none of them seems clearly superior to a simple trial-and-error approach (see [14]). Nonetheless, network architecture depends on the predictors too. Unfortunately, neural network are rigid in this sense: they cannot really choose among predictors, as opposite to other methods such as stepwise regressions and CARTs. Neural networks always use all the predictors given as input, so they should be chosen very carefully by the data scientist, for instance, by using a proper selection method. Clustering, principal component analysis, and CARTs themselves are all suitable approaches.

The various forms of their architecture give neural network a unique flexibility in dealing with data. Potentially, they can recognize any type of pattern. However, the architecture itself is the origin of their major drawback too, that is, their black-box structure. While anyone can easily "read" a tree, or interpret the parameters of a regression, this is generally impossible when dealing with neural networks. Of course, knowing the transfer functions and any parameters, we may write the ultimate function returning predictions, but then we would probably find no meaning in that. Too many parameters and too complex functions are often involved in neural network, and it must be accepted as it is. The usual validation tools can be used to measure the predictive power of a neural network, but unfortunately we cannot rely on model interpretation.

5. An application to automobile bodily injury claim data

As we may have noticed in Section 4, machine learning tools are very flexible, and could potentially improve many of the traditional processes in actuarial practice. Non-life reserving is just one of them.

This section describes the path that has been followed to predict the closing delay and claim amount on a publicly available motor insurance dataset.

5.1. Data

Data comes from an R package containing a number of datasets for actuarial and actuarial-affine applications (see [5]). It was also used in [4] as a starting point for generalized regression analysis. As such, we can surely rely on that without further validation. In [5], the dataset is called *Automobile bodily injury claim dataset in Australia* (`ausautoBI8999`), with the following description:

This dataset contains information on 22036 settled personal injury insurance claims in Australia. These claims arose from accidents occurring from July 1989 through to January 1999. Claims settled with zero payment are not included.

Actually, it relates to a small part of the entire claim scope of a motor insurance company, but it is probably the most interesting part to us. In effect, bodily injury claims are the most expensive and long-lasting ones, of course the most important to predict. In comparison to traditional triangle-based reserving methods (more suitable for standard claim prediction), machine learning can express its greatest potential for this kind of claims.
The dataset includes the following fields:

- `AccDate` for the claim accident date
- `ReportDate` for the claim reporting date
- `FinDate` for the claim closing date date
- `AccMth` for the claim accident month
- `ReportMth` for the claim reporting month
- `FinMth` for the claim closing month
- `OpTime` for the operational time
- `InjType1` for the injury severity of the first injured person
- `InjType2` for the injury severity of the second injured person
- `InjType3` for the injury severity of the third injured person
- `InjType4` for the injury severity of the fourth injured person
- `InjType5` for the injury severity of the fifth injured person
- `InjNb` for the number of injured persons (max 5)
- `Legal` for the legal representation (yes/no)
- `AggClaim` for the aggregate claim amount after closing.

category	**score**
NA (no injury)	0
minor injury	25
small injury	25
medium injury	50
not recorded	50
high injury	75
severe injury	75
fatal injury	100

Fig. 7: Codification of injury severity

The injury severity variables are categorical, but we converted them in numerical between 0 and 100, in order to calculate an overall severity score for each claim. The conversion is based on the rules in Figure 7, and the overall score of a single claim is given by the sum of its own scores (notice that it will be always positive because each claim caused one injury at least). Therefore, we define the numerical variables `InjScore1`, `InjScore2`, `InjScore3`, `InjScore4`, and `InjScore5`, in correspondence to the original `InjType1`, `InjType2`, `InjType3`, `InjType4`, and `InjType5`. Additionally, we get the overall severity score variable `InjScoreTot`.
Moreover, from the variables `AccDate`, `ReportDate`, and `FinDate`, we extract the year - `AccYr`, `ReportYr`, and `FinYr` - and use it to add two more variables:

- `ReportTime` for the reporting delay defined as `ReportYr` − `AccYr`
- `FinTime` for the closing delay defined as `FinYr` − `ReportYr`.

One last additional variable that will be useful in multiple linear regression is `LnAggClaim`, that is, the natural logarithm of the claim amount `AggClaim`. Using `ReportYr` and `FinTime`, let's have a look at the run-off triangle of the entire dataset in Figure 8 for the claim numbers, in Figure 9 for the claim payments, and in Figure 10 for the claim average payments (a dozen of claims relating to the highest amounts have been excluded). Unfortunately, some reporting years and some closing delays include too few observations, as we can easily observe in Figure 8. This is the reason why we will select a proper training/validation dataset as well as a test dataset for the ultimate analysis.

reporting year	closing delay						
	0	1	2	3	4	5	6
1993	**1.191**	**1.970**	**1.152**	**558**	426	277	49
1994	**874**	**1.531**	**870**	**709**	443	65	0
1995	**718**	**1.381**	**1.188**	**881**	134	0	0
1996	**529**	**1.511**	**1.314**	**174**	0	0	0
1997	585	2.020	283	0	0	0	0
1998	807	352	0	0	0	0	0
1999	14	0	0	0	0	0	0

Fig. 8: Claim numbers

reporting year	closing delay						
	0	1	2	3	4	5	6
1993	**26.038.265**	**77.236.474**	**74.285.051**	**58.268.484**	48.169.135	38.347.205	8.444.935
1994	**16.746.725**	**42.454.730**	**42.485.823**	**55.798.479**	44.833.960	7.119.057	0
1995	**6.076.308**	**20.958.156**	**41.895.020**	**50.580.334**	8.286.925	0	0
1996	**4.090.537**	**21.665.535**	**46.736.025**	**8.158.885**	0	0	0
1997	3.787.223	33.675.352	8.543.935	0	0	0	0
1998	6.207.963	3.475.759	0	0	0	0	0
1999	36.599	0	0	0	0	0	0

Fig. 9: Claim amounts

More specifically, the reporting years 1993-1996 for the sole closing delays from 0 to 3 (in bold in Figure 8, 9 and 10) and will be considered for training and validation, while the reporting years 1997-1998 will be considered for test using all the available closing delays, that is, from 0-2 for 1997 and 0-1 for 1998 (in Roman in Figure 8, 9 and 10). All the other observations (in gray in Figure 8, 9 and 10) are not used in the analysis.

Figure 8 also demonstrate that there is no regular path by reporting year or closing delay. For instance, the zero-delay claims tend to decrease by reporting year, while the 3-year-delay claims have a rather different path. Comparing reporting years, we see less differences, but still some relevant discordances. For instance, compare the 1-year-delay claim average payment to the 2-year-delay claim average payment in Figure 10: in 1993 the latter is about 50% greater than the former, while in 1996 the latter is about 150%

reporting year	closing delay						
	0	1	2	3	4	5	6
1993	**21.863**	**39.206**	**64.484**	**104.424**	113.073	138.438	172.346
1994	**19.161**	**27.730**	**48.834**	**78.700**	101.205	109.524	0
1995	**8.463**	**15.176**	**35.265**	**57.412**	61.843	0	0
1996	**7.733**	**14.339**	**35.568**	**46.890**	0	0	0
1997	6.474	16.671	30.191	0	0	0	0
1998	7.693	9.874	0	0	0	0	0
1999	2.614	0	0	0	0	0	0

Fig. 10: Claim average amounts

greater than the former. Remember that triangle-based methods often assume no differences in path among reporting years, but actually we would lose some important information in this specific case.
In order to better include timing information, we consider both the reporting year and the closing delay as categorical variables. As a consequence, we convert each of them in three binary variables (the forth one would be redundant):

- `ReportYr` is converted to `ReportYr1` for 1994, `ReportYr2` for 1995, and `ReportYr3` for 1996
- `FinTime` is converted to `FinTime1` for one year, `FinTime2` for two year, and `FinTime3` for three year.

Similarly, we convert `InjNb` to `InjNb2` for two injuries, `InjNb3` for three injuries, `InjNb4` for four injuries, and `InjNb5` for five injuries.
Finally, we get the following fields:

- `InjScoreTot` for the overall severity score
- `InjNb2` for two injures (1/0)
- `InjNb3` for three injures (1/0)
- `InjNb4` for four injures (1/0)
- `InjNb5` for five injures (1/0)
- `LegalBin` for the legal representation (1/0)

- `ReportTime` for the reporting delay
- `ReportYr1` for the reporting year 1994 (1/0)
- `ReportYr2` for the reporting year 1995 (1/0)
- `ReportYr3` for the reporting year 1996 (1/0)
- `FinTime` for the closing delay
- `FinTime1` for the 1-year closing delay (1/0)
- `FinTime2` for the 2-year closing delay (1/0)
- `FinTime3` for the 3-year closing delay (1/0)
- `AggClaim` for the aggregate claim amount after closing
- `LnAggClaim` for the logarithm of `AggClaim`

beyond an ID variable to identify the different records.

5.2. Claim closing delay estimation

In our framework, estimating the closing delay means using the predictors `InjScoreTot`, `InjNb2`, `InjNb3`, `InjNb4`, `InjNb5`, `LegalBin`, `ReportTime`, `ReportYr1`, `ReportYr2`, and `ReportYr3` as inputs for some machine learning tool to return an estimation of the probability that the claim is definitely closed after 0, 1, 2 and 3 years after the reporting. In other words, the target variable is `FinTime`. We use three methods: naive Bayes, k-nearest neighbors, and classification tree.

According to the description in Subsection 4.3, the number k of neighbors should correspond to the lowest validation error. As demonstrated in Figure 14, it occurs when $k = 19$. It is worth noting the peculiar, opposite shape of the two lines. In Subsection 4.3, we noticed that the training dataset has actually nothing to gain from k-nearest neighbors prediction: in fact, the more the neighbors, the greater the error. This happens because the best prediction in the training dataset always occurs when $k = 1$ by definition. On the contrary, the validation error slightly decreases as k increases since the algorithm neglects more and more noisy information. So this is not surprising if the greatest validation error occurs when $k = 1$, just as the training error is minimum.

Likewise, we need to choose a proper classification tree on the base of the validation error as well. In Figure 17, we observe that the lowest validation error occurs after twelve splits (gray dashed line in Figure 17), but, as discussed in Subsection 4.4, this is not necessarily the best choice if we want to take into account the complexity of the tree too. This is the reason why we will use the best-pruned tree for scoring, that is, the tree consisting of three splits (black dashed line in Figure 17). Furthermore, Figure 18, 19, and 20 shows the full tree, the minimum error tree, and the best pruned tree respectively.

Remember: we are not really interested in classifying record among one of the four classes, rather we will directly used the estimated probabilities. However, trying to classifying them using the various methods is the easiest way to compare their performances. Therefore, such an assessment is reported in Figure 12-13, 15-16, and 21-22. There we can observe the related confusion matrices together with the training and validation errors. All the errors seems to swing around 60%. In other words: if we use the rule that assigns a record the greatest predicted probability among the four possible category, we will correctly predict around 40% closing delays.

Unfortunately, this represents a rather poor result, and it can be proved through a straightforward remark. The training data is characterized by the prior (empirical) probabilities per closing delay category as in Figure 11. The most obvious prediction algorithm would classify all the observations in the category that appears most often in the training dataset, that is, category 1 with 38,86%. In such a case, we would correctly predict 38,86% claims, that is, an error of 61,14%. But it is just slightly higher than the overall validation errors of the 19-nearest neighbors algorithm (see Figure 16) and the best pruned classification tree (see Figure 22). Curiously, it is even lower than the overall validation error of naive Bayes (see Figure 13). In other word, it does not seem we gain any insight by using machine learning on this dataset to predict closing delay. This is simply due to the low informative value of the data itself. However, we are going to see in Subsection 5.3 the great potential of machine learning in claim amount estimation as compared to traditional regression methods.

closing delay	prior probability
0	20,01%
1	38,86%
2	27,00%
3	14,14%

Fig. 11: Prior probabilities using training data

Actual	*Predicted Class*				*Cases*	*Errors*	*Errors*
Class	0	1	2	3	*Number*	*Number*	*Percentage*
0	740	1.069	151	27	1.987	1.247	62,76%
1	870	2.210	689	90	3.859	1.649	42,73%
2	434	1.380	764	103	2.681	1.917	71,50%
3	242	730	281	151	1.404	1.253	89,25%
Total					**9.931**	**6.066**	**61,08%**

Fig. 12: Naive Bayes summary results using training data

Actual	*Predicted Class*				*Cases*	*Errors*	*Errors*
Class	0	1	2	3	*Number*	*Number*	*Percentage*
0	477	708	122	18	1.325	848	64,00%
1	536	1.454	490	54	2.534	1.080	42,62%
2	284	988	483	88	1.843	1.360	73,79%
3	153	445	226	94	918	824	89,76%
Total					**6.620**	**4.112**	**62,11%**

Fig. 13: Naive Bayes summary results using validation data

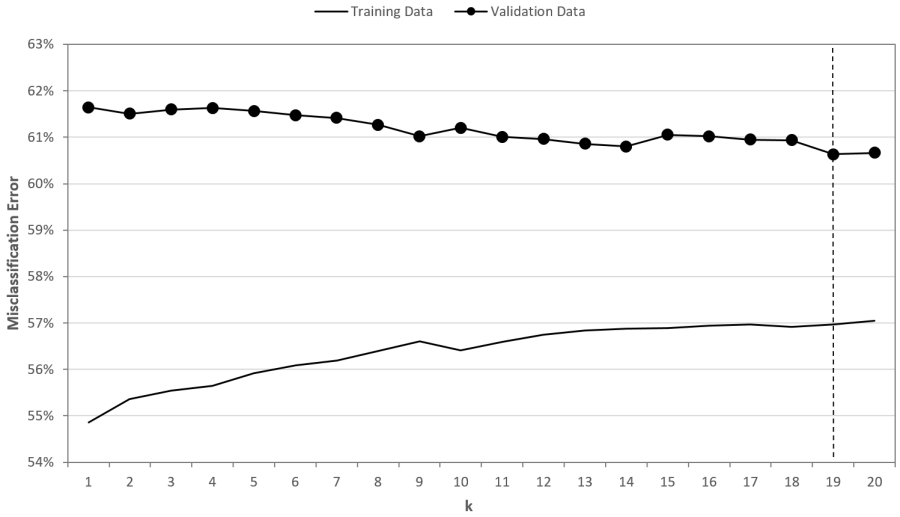

Fig. 14: K-nearest neighbors - training and validation error varying by k

Actual	*Predicted Class*				*Cases*	*Errors*	*Errors*
Class	0	1	2	3	*Number*	*Number*	*Percentage*
0	472	1.389	105	21	1.987	1.515	76,25%
1	367	3.140	286	66	3.859	719	18,63%
2	222	1.869	502	88	2.681	2.179	81,28%
3	115	938	191	160	1.404	1.244	88,60%
Total					**9.931**	**5.657**	**56,96%**

Fig. 15: K-nearest neighbors summary results using training data

Actual	*Predicted Class*				*Cases*	*Errors*	*Errors*
Class	0	1	2	3	*Number*	*Number*	*Percentage*
0	267	959	83	16	1.325	1.058	79,85%
1	269	1.990	224	51	2.534	544	21,47%
2	164	1.355	248	76	1.843	1.595	86,54%
3	75	614	128	101	918	817	89,00%
Total					**6.620**	**4.014**	**60,63%**

Fig. 16: K-nearest neighbors summary results using validation data

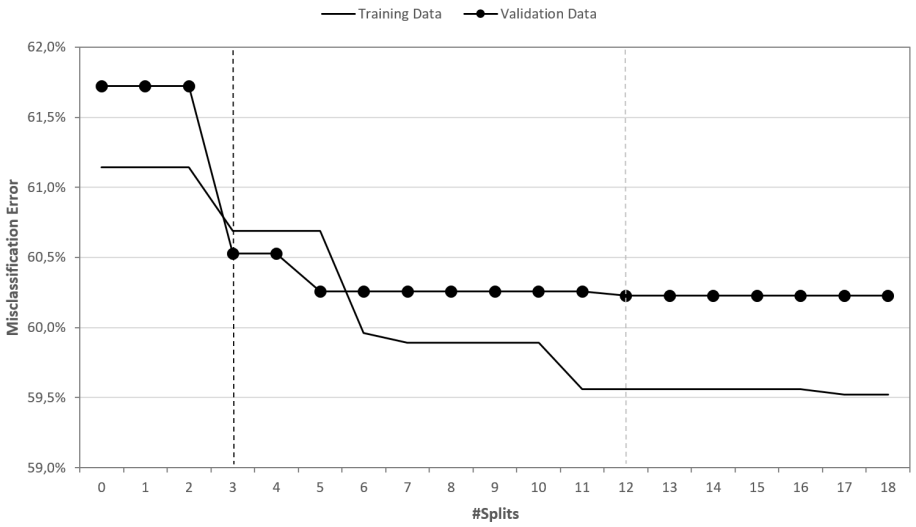

Fig. 17: Classification tree - training and validation error varying by number of splits

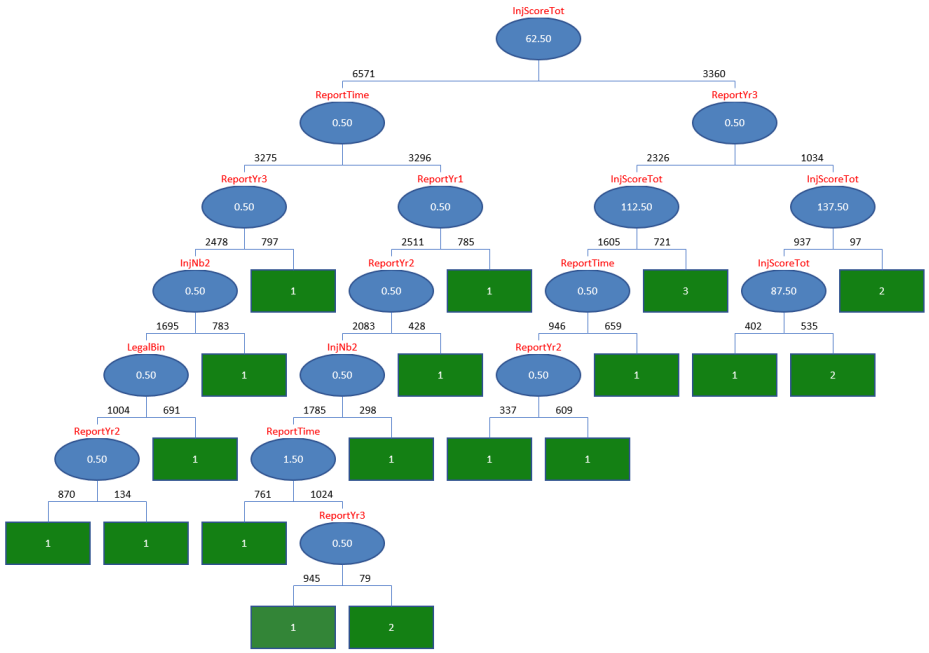

Fig. 18: Full classification tree

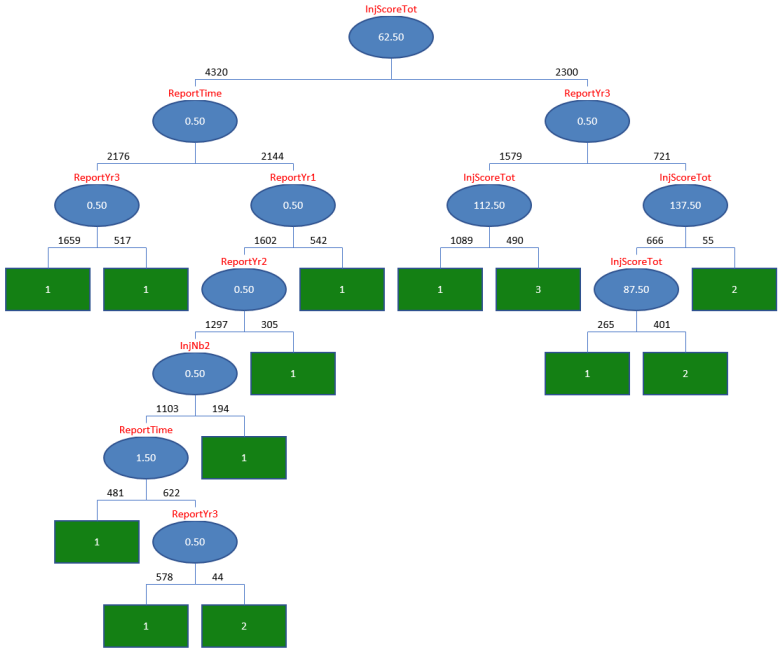

Fig. 19: Minimum error classification tree

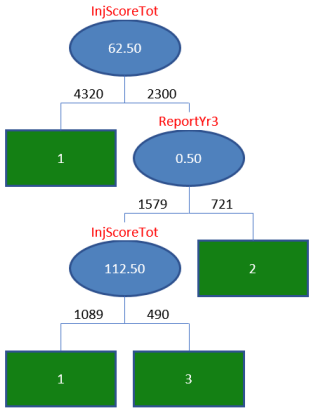

Fig. 20: Best pruned classification tree

Actual	*Predicted Class*				*Cases*	*Errors*	*Errors*
Class	0	1	2	3	*Number*	*Number*	*Percentage*
0	0	1.892	57	38	1.987	1.987	100,00%
1	0	3.412	258	189	3.859	447	11,58%
2	0	2.101	347	233	2.681	2.334	87,06%
3	0	1.094	49	261	1.404	1.143	81,41%
Total					**9.931**	**5.911**	**59,52%**

Fig. 21: Classification tree summary results using training data

Actual	*Predicted Class*				*Cases*	*Errors*	*Errors*
Class	0	1	2	3	*Number*	*Number*	*Percentage*
0	0	1.229	67	29	1.325	1.325	100,00%
1	0	2.121	294	119	2.534	413	16,30%
2	0	1.363	315	165	1.843	1.528	82,91%
3	0	696	45	177	918	741	80,72%
Total					**6.620**	**4.007**	**60,53%**

Fig. 22: Classification tree summary results using validation data

5.3. Claim payment amount estimation

In our framework, estimating the claim payment amount means using the predictors previously used to estimate the closing delay (see Section 5.2) together with the binary variables `FinTime1` for the 1-year closing delay, `FinTime2` for the 2-year closing delay, and `FinTime3` for the 3-year closing delay. They will represent the inputs for some machine learning tool to return an estimation of `AggClaim`. We use three methods: generalized regression with Gamma distribution, regression tree, and neural network.

When it comes with regression methods, we should pay attention to the features of our predictors. First, it's important to avoid asymmetries and heavy tails. This can be check by having a look at the skewness and kurtosis in Figure 23. Only the target variable `AggClaim` got very high skewness (5,42) and kurtosis (42,21), but this is simply due to the nature of the variable itself. A simple solution is represented by the natural logarithm of `AggClaim`, `LnAggClaim`, which would permit us to use multiple linear regression (Figure 23 also reports skewness and kurtosis of `LnAggClaim`: both of them are very low). However, it would also introduce a significant component of transformation bias (see Subsection 4.1), so we will not try this approach.

Another typical issue related to regression models is multicollinearity among numerical variables. In our case, however, if all the correlations among the binary variables of the same categorical variable are excluded, few remaining correlations are significant. Therefore, we will run the gamma regression including all the predictors, expecting that the highest correlations will be automatically solved by the stepwise selection algorithm. Figure 24 shows the related results (we also included multiple linear regression results, although they will be not used for claim amount prediction). By comparing Figure 23 and Figure 24, it is worth noting that the selected 9-coefficient model includes the predictors characterized by the highest correlations with `LnAggClaim`. Moreover, the exclusion of all the `InjNb` binary variables leads to the exclusion of the most relevant multicollinearities, that is, those among `InjNb` and `InjScoreTot` (see Figure 23). Actually, this is not surprising: to some extent, in effect, the greater the number of injuries, the greater the overall claim severity.

When it comes with regression trees, we need to do some further remarks with respect to classification tree. Actually, the former predict numerical variables, while the latter classify records among a range of categories. Given that each claim relates to a different payment, a mere full tree would get as many leaves as the number of records in the training dataset. Obviously,

	InjScoreTot	InjNb2	InjNb3	InjNb4	InjNb5	LegalBin	ReportTime	ReportYr1	ReportYr2	ReportYr3	FinTime1	FinTime2	FinTime3	LnAggClaim	AggClaim
mean	56,99	0,19	0,13	0,08	0,09	0,63	0,82	0,24	0,25	0,21	0,39	0,27	0,14	**9,59**	**35.857,34**
st. deviation	37,98	0,39	0,34	0,27	0,29	0,48	1,14	0,43	0,43	0,41	0,49	0,45	0,35	**1,42**	**67.073,67**
skewness	1,36	1,58	2,15	3,14	2,79	-0,52	1,39	1,21	1,14	1,40	0,47	1,02	2,07	**-0,53**	**5,42**
kurtosis	2,16	0,49	2,63	7,86	5,79	-1,73	1,26	-0,53	-0,69	-0,04	-1,78	-0,97	2,29	**1,48**	**42,21**

	InjScoreTot	InjNb2	InjNb3	InjNb4	InjNb5	LegalBin	ReportTime	ReportYr1	ReportYr2	ReportYr3	FinTime1	FinTime2	FinTime3	LnAggClaim
InjScoreTot	100,0%													
InjNb2	-5,4%	100,0%												
InjNb3	22,3%	-19,0%	100,0%											
InjNb4	36,9%	-14,1%	-11,5%	100,0%										
InjNb5	68,6%	-15,6%	-12,6%	-9,4%	100,0%									
LegalBin	9,1%	0,1%	2,8%	4,1%	6,6%	100,0%								
ReportTime	-13,0%	-7,8%	-9,5%	-7,1%	-8,8%	8,3%	100,0%							
ReportYr1	-8,7%	1,0%	-4,2%	-4,4%	-6,2%	-29,4%	2,2%	100,0%						
ReportYr2	17,9%	6,4%	10,0%	8,9%	10,0%	3,2%	-20,3%	-32,7%	100,0%					
ReportYr3	17,8%	4,9%	8,2%	8,2%	11,0%	19,0%	-20,5%	-29,3%	-30,2%	100,0%				
FinTime1	-6,9%	1,7%	-0,1%	-0,6%	-6,7%	-0,1%	-2,1%	-0,2%	-6,5%	4,5%	100,0%			
FinTime2	13,0%	1,6%	4,3%	5,4%	9,2%	5,6%	-6,9%	-6,9%	1,5%	11,6%	-48,7%	100,0%		
FinTime3	14,6%	-0,7%	2,2%	3,2%	11,7%	-1,2%	-3,1%	6,1%	11,9%	-13,6%	-32,0%	-24,8%	100,0%	
LnAggClaim	**26,8%**	**-1,4%**	**3,9%**	**7,2%**	**19,5%**	**20,6%**	**27,7%**	**0,9%**	**-5,1%**	**-10,4%**	**-11,9%**	**20,7%**	**28,1%**	**100,0%**

Fig. 23: Descriptive statistics

Stepwise Selection	#Coeffs	R^2	Adj. R^2	Intercept	InjScoreTot	InjNb2	InjNb3	InjNb4	InjNb5	LegalBin	ReportTime	ReportYr1	ReportYr2	ReportYr3	FinTime1	FinTime2	FinTime3
	1	0,0%	0,0%	x													
	2	7,9%	7,9%	x													x
	3	16,3%	16,3%	x							x						x
	4	25,8%	25,8%	x							x					x	x
	5	30,8%	30,8%	x							x				x	x	x
	6	34,7%	34,7%	x	x						x				x	x	x
	7	36,8%	36,8%	x	x					x	x				x	x	x
	8	38,0%	38,0%	x	x					x	x			x	x	x	x
	9	**39,4%**	**39,3%**	**x**	**x**					**x**	**x**		**x**	**x**	**x**	**x**	**x**

Multiple Linear Regression																	
Coefficient	7,7530	0,0090								0,5396	0,3511		-0,4377	-0,5941	0,7727	1,4149	1,8110
Standard Error	0,0334	0,0003								0,0239	0,0105		0,0294	0,0317	0,0310	0,0338	0,0399
t-Statistic	232,26	27,92								22,61	33,32		-14,90	-18,77	24,94	41,84	45,35
P-Value	0,0000	0,0000								0,0000	0,0000		0,0000	0,0000	0,0000	0,0000	0,0000
Conf. Interval Lower	7,6876	0,0083								0,4928	0,3305		-0,4952	-0,6562	0,7120	1,3486	1,7328
Conf. Interval Upper	7,8185	0,0096								0,5864	0,3718		-0,3801	-0,5321	0,8334	1,4812	1,8893

Gamma Regression																	
Coefficient	8,5839	0,0084								0,3353	0,3985		-0,4863	-0,5905	0,5660	1,2402	1,6610
Standard Error	0,0478	0,0005								0,0342	0,0151		0,0421	0,0453	0,0444	0,0484	0,0572
t-Statistic	179,54	18,31								9,81	26,40		-11,56	-13,02	12,76	25,61	29,04
P-Value	0,0000	0,0000								0,0000	0,0000		0,0000	0,0000	0,0000	0,0000	0,0000
Conf. Interval Lower	8,4849	0,0075								0,2677	0,3665		-0,5686	-0,6807	0,4782	1,1439	1,5476
Conf. Interval Upper	8,6840	0,0093								0,4027	0,4308		-0,4032	-0,4996	0,6532	1,3362	1,7752

Fig. 24: Regression summary after stepwise selection

this is not feasible, so we need an additional rule to stop the growth of the full tree, and the prediction related to a specific leaf will equal the average payment of the training records in that leaf. Generally, the rule is quite empirical, for instance, a maximum number of tree levels, a maximum number of nodes, and so on.

In our analysis, we choose a minimum number of records in any leaf, and then predictions and errors from that tree are evaluated. The results are summarized in Figure 25, not only training error, validation error, and overall error, but also a percentage weighted error and a percentage overall error. The percentage weighted error is based on the sixteen percentage errors per reporting year and closing delay weighted on the actual payment amount itself. Instead, the percentage overall error is simply the percentage difference between the total of the payments in the dataset and the total of the related estimates.

First, Figure 25 shows no overfitting, which is a quite important advantage for any predictive model. In other words, training error and validation error are very close regardless of the minimum records per leaf. Secondly, observe the fluctuation of the percentage errors in the last to columns in Figure 25. When very few records are required in each leaf, it seems that a lot of noise affects the tree: errors are quite material, especially the percentage weighted errors, which are greatest between 10 and 40 minimum records per leaf. After all, the regression tree is predicting much better between 50 and 100: it will be probably there where it performs at best. Nonetheless, further increase in minimum records per leaf implies a new increase in error: in fact, if too many records are required into each leaf, the regression tree will no longer be able to detect information in data.

All in all, the chosen tree is highlighted in bold in Figure 25, that is, the tree leading to the lowest percentage weighted record 3,67%. Further, although it leads to the lowest average overall error, it does not lead to the lowest percentage overall error (which is however very low).

Unfortunately, we cannot report the whole full tree and best pruned tree since they are too big. Anyway, it is worth noting that all the predictors have been used to build the tree, that is, each predictor is the splitting variable for one node at least. As briefly mentioned in Subsection 4.5, standard neural networks lack an embedded algorithm to select relevant predictors and exclude irrelevant predictors, but we may refer to the predictors implicitly selected by the regression tree itself. Since it has used all of them, we will run the neural network on the full dataset.

minimum records per leaf	*average training error*	*average validation error*	*average overall error*	*percentage weighted error*	*percentage overall error*
10	28.262	28.554	28.379	16,63%	0,38%
20	28.393	28.622	28.485	16,74%	0,38%
30	27.819	28.020	27.899	14,28%	0,39%
40	27.819	28.020	27.899	14,28%	0,39%
50	**26.695**	**27.528**	**27.028**	**3,67%**	**0,29%**
60	26.982	27.477	27.180	5,06%	0,22%
70	27.088	27.326	27.183	5,54%	0,10%
80	27.154	27.393	27.250	5,49%	0,09%
90	27.185	27.331	27.243	6,10%	0,01%
100	27.273	27.385	27.318	6,30%	0,15%
110	27.339	27.457	27.386	6,83%	0,16%
120	27.525	27.568	27.542	9,14%	0,16%
130	27.603	27.596	27.600	9,47%	0,10%
140	27.647	27.679	27.660	9,83%	0,25%
150	27.729	27.753	27.739	9,85%	0,41%
160	27.632	27.680	27.651	8,43%	0,37%
170	27.652	27.662	27.656	8,45%	0,40%
180	27.743	27.718	27.733	9,09%	0,39%
190	27.764	27.741	27.755	9,38%	0,43%
200	27.806	27.797	27.802	9,60%	0,51%
210	27.830	27.777	27.809	9,59%	0,48%
220	27.842	27.788	27.820	9,81%	0,50%
230	27.826	27.766	27.802	9,67%	0,48%
240	27.952	27.866	27.918	9,93%	0,54%
250	27.971	27.906	27.945	10,18%	0,55%
260	27.971	27.913	27.948	10,96%	0,56%
270	28.000	28.116	28.046	11,02%	0,67%
280	28.012	28.139	28.063	11,30%	0,72%
290	28.032	28.171	28.088	11,50%	0,69%
300	28.196	28.430	28.289	13,86%	0,63%

Fig. 25: Regression tree performance varying by minimum number of records per leaf

number of hidden neurons	*average training error*	*average validation error*	*average overall error*	*percentage weighted error*	*percentage overall error*
1	30.420	30.068	30.279	16,23%	12,50%
2	29.634	29.621	29.629	15,20%	9,17%
3	29.706	29.838	29.758	13,96%	12,09%
4	28.858	28.947	28.894	8,96%	8,57%
5	29.028	29.285	29.131	11,26%	9,27%
6	28.734	28.828	28.771	10,08%	8,52%
7	28.848	29.338	29.044	11,13%	9,80%
8	**28.068**	**28.607**	**28.284**	**8,30%**	**5,57%**
9	28.609	29.273	28.875	8,89%	8,28%
10	28.615	29.411	28.933	10,66%	6,89%
11	28.318	29.056	28.613	10,04%	8,81%
12	28.553	29.229	28.824	10,60%	8,02%
13	28.376	28.975	28.616	9,81%	8,33%
14	28.375	29.368	28.772	9,82%	7,10%
15	28.261	28.900	28.516	8,82%	6,44%
16	28.480	29.020	28.696	9,64%	7,19%
17	28.402	29.299	28.761	10,07%	7,77%
18	28.188	28.939	28.488	9,00%	6,90%
19	28.321	28.831	28.525	8,64%	6,55%
20	28.044	28.890	28.382	9,06%	5,76%

Fig. 26: Neural network performance varying by number of hidden neurons

So far, we know the structure of the input layer: thirteen neurons for thirteen predictors. As discussed in Subsection 4.5, we may also accept the assumption of one single hidden layer. However, how many hidden neurons shall we use? Intuitively, too few hidden neurons will not be able to detect information, while too many hidden neurons will imply overfitting. Just like for the regression tree, let's try various cases by gradually increasing the number of hidden layers, say from 1 to 20. The results are summarized in Figure 26 (the fields have the same meaning as in Figure 25). Once again, training error and validation error are very close: the neural network is not overfitting. The bad performance due to few hidden neurons - between 1 and 7 - is quite clear. By adding more hidden neurons, the error tends to decrease slowly, but it seems that it is not overfitting by 20 neurons yet: it will probably start overfitting a bit later. The chosen neural network is that (in bold in Figure 25) leading to the lowest percentage weighted error, which also coincides to

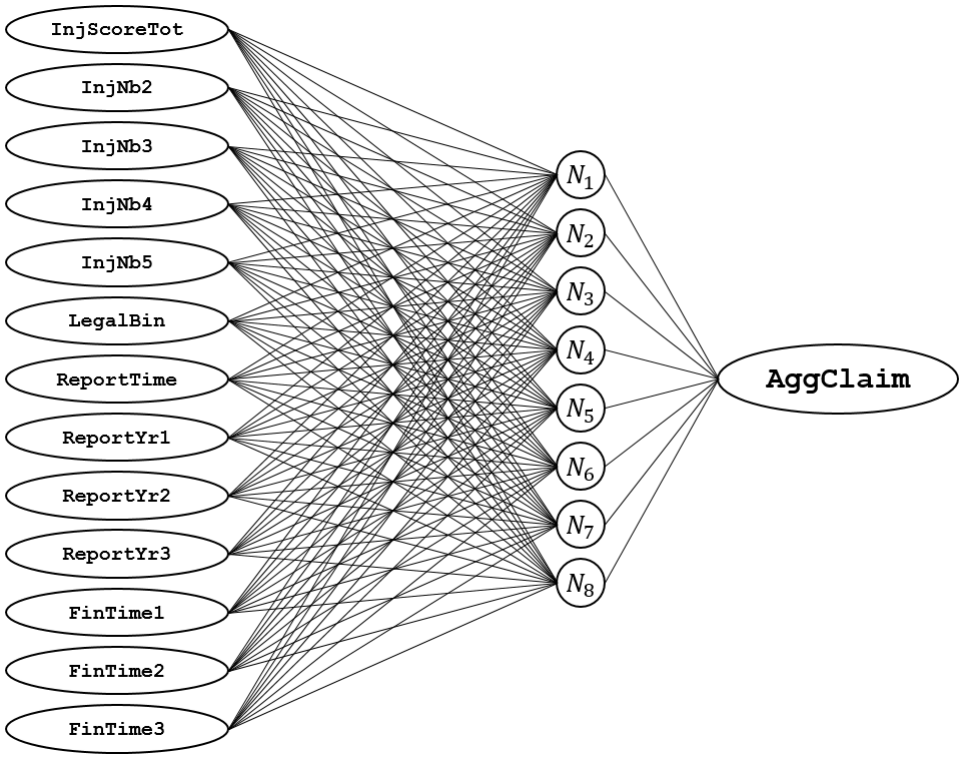

Fig. 27: 8-hidden-neuron neural network for claim amount prediction

the lowest average overall error and the lowest percentage overall error. A formal representation of this neural network is in Figure 27, while the related parameters are reported in Figure 28.

As a conclusion to this subsection, we will present some global results. Firstly, the convergence plots of the regression tree and the neural network in Figure 29 and 30 respectively. More importantly, however, we should compare actual data and estimations - see Figure 31-34. Remember that the chosen regression tree and neural network got a weighted error of 3,67% and 8,30% respectively, so their good punctual estimations for the claim payments are not surprising at all. Likewise, the gamma regression shows good performance too, although there seems to be some overestimation by closing delay 2 and 3.

	Bias	1,6433	0,9728	-2,6430	3,4999	-0,4701	-0,8636	0,6062	1,2669
	InjScoreTot	-3,6164	1,6972	-1,7925	-7,9758	-1,0843	3,7490	-2,8758	-1,2362
	InjNb2	2,5818	1,4252	2,8352	-1,1509	-0,5093	-0,7649	-0,7377	-0,3781
	InjNb3	-1,7530	-2,0918	1,9940	-2,4436	1,1031	0,3761	1,7829	0,2160
	InjNb4	3,7145	1,6844	-0,2911	1,8491	-0,1809	-1,6364	0,5876	0,3556
	InjNb5	1,5213	-0,3456	2,8264	-1,0558	0,0054	-1,2920	1,6087	-1,5291
	LegalBin	-1,4256	-1,0866	0,2715	-0,2891	-1,0853	-0,0509	0,5342	-2,8530
	ReportTime	-6,4297	-3,9165	-0,2582	0,6026	-0,6748	-2,5407	-1,1021	-2,6102
	ReportYr1	4,7466	3,8977	0,5149	-2,6189	-0,4819	0,2611	2,5912	1,5195
	ReportYr2	-2,8519	-2,7222	0,8436	-1,0376	-0,3715	0,0108	1,1027	-1,6984
	ReportYr3	-1,7670	-2,6942	-0,4083	-2,1818	-1,1539	-0,0667	1,8782	0,1445
	FinTime1	1,0403	1,3477	-0,3577	-0,7600	0,1805	0,1251	0,8446	0,2697
	FinTime2	1,5695	1,8607	-0,6853	-0,7159	0,5422	-0,6917	1,0201	0,9820
	FinTime3	-1,4015	-0,6039	0,5690	-0,4504	-0,3393	-2,9462	-2,0530	1,1273
	Bias	*Neuron 1*	*Neuron 2*	*Neuron 3*	*Neuron 4*	*Neuron 5*	*Neuron 6*	*Neuron 7*	*Neuron 8*
Response	0,3426	-2,8543	4,0754	-1,9584	-2,4236	1,7553	-3,2518	-2,0831	-1,4936

Fig. 28: 8-hidden-neuron neural network parameters

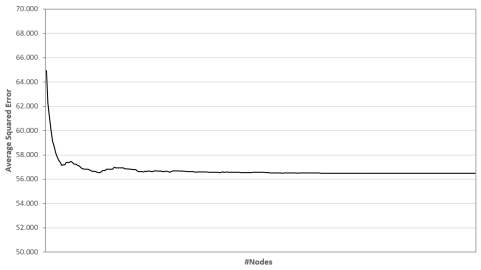

Fig. 29: Convergence of the error for the regression tree

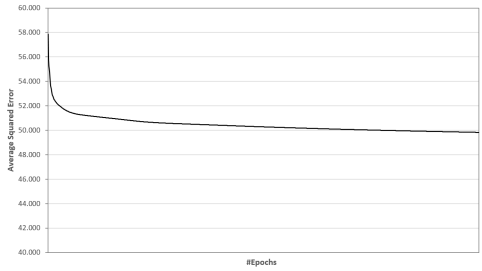

Fig. 30: Convergence of the error for the neural network

reporting year	closing delay 0	1	2	3
1993	26.038.265	77.236.474	74.285.051	58.268.484
1994	16.746.725	42.454.730	42.485.823	55.798.479
1995	6.076.308	20.958.156	41.895.020	50.580.334
1996	4.090.537	21.665.535	46.736.025	8.158.885

Fig. 31: Actual data

reporting year	closing delay 0	1	2	3
1993	23.942.277	72.096.426	85.372.988	68.135.627
1994	15.615.349	43.607.867	49.123.608	69.977.076
1995	6.294.610	21.590.823	41.199.048	51.380.998
1996	4.338.858	23.040.826	46.178.322	10.158.513

Fig. 32: Gamma regression

reporting year	closing delay 0	1	2	3
1993	27.777.729	74.715.426	74.860.543	53.671.653
1994	17.317.427	44.645.051	41.115.866	56.014.481
1995	7.819.221	22.443.511	40.888.916	51.092.375
1996	5.285.040	22.293.701	46.183.852	9.048.303

Fig. 33: Regression tree

reporting year	closing delay 0	1	2	3
1993	30.323.454	80.122.598	74.090.121	56.198.161
1994	21.876.621	49.791.426	44.957.040	51.693.100
1995	8.007.124	25.605.931	47.106.288	49.418.199
1996	5.358.636	27.349.566	46.177.586	8.455.383

Fig. 34: Neural network

5.4. Claim reserve estimation as an ensemble

In machine learning, an *ensemble* is a complex machine learning algorithm consisting of a number of simpler machine learning tools. The combination may be very easy to implement (for instance, relating to the three predictive models in Subsection 5.3, an ensemble for the prediction of the claim amount could be the average of the three different predictions), or very difficult. In any case, the goal is the improvement of the predictive performance.

Performance assessment for an ensemble typically requires more computation than performance assessment for its constituents, so ensembles may be thought of as a way to compensate for poor learning algorithms by performing a lot of extra computation. Therefore, fast algorithms such as CARTs are commonly used in some ensemble versions (for example *random forests*, *bagging trees*, and *boosting trees*).

From this perspective, the estimation expressed by the (4) may be seen exactly this way, that is, as an ensemble resulting from the combination of a classification tool and a regression tool. Of course, considering all the tools described in Section 4, any of the classification methods may be combined with any of the regression methods. The choice will depend on the performance reported in Figure 35-38 by reporting year, and the overall performance reported in Figure 39.

Globally, gamma regression significantly misestimates the claim payments, especially for the reporting year 1993 and 1994. More importantly, the major problem is that the performance is quite different across reporting years

Actual claim amount 235.828.275	*gamma regression*	*regression tree*	*neural network*
random	262.835.516	242.262.782	245.845.509
naive Bayes	239.075.514	217.568.447	231.965.733
k-nearest neighbors	249.685.356	229.945.901	238.026.479
classification tree	258.083.300	233.422.075	243.846.286

Δ%	*gamma regression*	*regression tree*	*neural network*
random	11,45%	2,73%	4,25%
naive Bayes	1,38%	-7,74%	-1,64%
k-nearest neighbors	5,88%	-2,49%	0,93%
classification tree	9,44%	-1,02%	3,40%

Fig. 35: Predictive performance for reporting year 1993

Actual claim amount 157.485.757	*gamma regression*	*regression tree*	*neural network*
random	174.026.704	155.206.357	167.166.597
naive Bayes	177.763.171	149.619.919	165.548.345
k-nearest neighbors	182.202.506	159.124.199	169.743.174
classification tree	182.470.513	156.827.235	170.113.669

Δ%	*gamma regression*	*regression tree*	*neural network*
random	10,50%	-1,45%	6,15%
naive Bayes	12,88%	-4,99%	5,12%
k-nearest neighbors	15,69%	1,04%	7,78%
classification tree	15,86%	-0,42%	8,02%

Fig. 36: Predictive performance for reporting year 1994

Actual claim amount 119.509.819	*gamma regression*	*regression tree*	*neural network*
random	103.147.343	107.212.659	114.257.199
naive Bayes	126.742.857	129.321.735	136.983.874
k-nearest neighbors	117.758.364	120.167.319	128.321.910
classification tree	116.673.918	119.593.669	126.714.629

Δ%	*gamma regression*	*regression tree*	*neural network*
random	-13,69%	-10,29%	-4,40%
naive Bayes	6,05%	8,21%	14,62%
k-nearest neighbors	-1,47%	0,55%	7,37%
classification tree	-2,37%	0,07%	6,03%

Fig. 37: Predictive performance for reporting year 1995

Actual claim amount 80.650.982	*gamma regression*	*regression tree*	*neural network*
random	83.862.742	82.642.642	86.413.193
naive Bayes	87.835.102	86.321.719	91.700.340
k-nearest neighbors	83.617.031	81.500.277	87.200.645
classification tree	83.901.388	82.788.649	87.006.662

Δ%	*gamma regression*	*regression tree*	*neural network*
random	3,98%	2,47%	7,14%
naive Bayes	8,91%	7,03%	13,70%
k-nearest neighbors	3,68%	1,05%	8,12%
classification tree	4,03%	2,65%	7,88%

Fig. 38: Predictive performance for reporting year 1996

Actual claim amount 593.474.833	*gamma regression*	*regression tree*	*neural network*
random	623.872.306	587.324.439	613.682.497
naive Bayes	631.416.644	582.831.820	626.198.292
k-nearest neighbors	633.263.257	590.737.696	623.292.207
classification tree	641.129.119	592.631.628	627.681.248

Δ%	*gamma regression*	*regression tree*	*neural network*
random	5,12%	-1,04%	3,40%
naive Bayes	6,39%	-1,79%	5,51%
k-nearest neighbors	6,70%	-0,46%	5,02%
classification tree	8,03%	-0,14%	5,76%

Fig. 39: Overall predictive performance

regardless of the classification tool used. On the other hand, the best performances seem to be related to two ensembles:

- k-nearest neighbors and regression tree (overall error -0,46%)
- classification tree and regression tree (overall error -0,14%)

and their error per reporting year is always lower than 3% in absolute value. Basically, if one of this two combinations is used to predict the payment as soon as the claim is reported (assuming that all the relevant information is immediately available), we will reserve an extremely accurate amount for any reporting year. That's sounds quite good, but it is not necessarily the best solution. Actually, a proper reserve should always account for some level of conservatism, as long as it is uniformly included into each allocation. For instance, Figure 35-38 shows that the two ensembles

- k-nearest neighbors and neural network (overall error 5,02%)
- classification tree and neural network (overall error 5,76%)

slightly overestimate the claim payments. In particular, the overestimation is quite stable over the reporting years 1994-1996, that is, they overestimate in the same direction, with the same magnitude, regardless of the reporting year. If the company reserves using one of the two aforementioned ensembles, it will most probably overestimate the amount by around 6-8% (we do not consider the lower but older - thus less significant - errors for the year 1993). For our purpose, it could really be the best compromise, including a reasonable prudence margin.

Once the best ensembles are selected, the last step is the estimation of the reserve on the test dataset, that is, the allocation for the claims reported in 1997 and 1998 (see Subsection 5.1). A further assumption is immediately necessary: given that the reporting year is a categorical variable, the new reporting years are not included in our models, so they will be replaced by the last year available, that is, 1996 - the most significant one in terms of timing. The results are stored in Figure 40. As specified in the column fields, remember that these two reporting years contain information for a limited number of closing delay categories: 0, 1 and 2 for 1997, while 0 and 1 for 1998. This is because the dataset was extracted at year end 1999. Whatever the ensemble is, the overall claim amounts are materially overestimated for both of the years. A total 46M claim amount in 1997 is predicted to be about

		Reporting year 1997 (closing delay 0-2 years)	*Reporting year 1998* (closing delay 0-1 years)
	Actual claim amount	**46.006.511**	**9.683.722**
k-nearest neighbors & regression tree	Predicted amount	63.174.500	22.272.094
	Delta	17.167.988	12.588.372
classification tree & regression tree	Predicted amount	64.860.982	24.113.611
	Delta	18.854.471	14.429.888
k-nearest neighbors & neural network	Predicted amount	68.002.405	22.774.882
	Delta	21.995.894	13.091.160
classification tree & neural network	Predicted amount	68.079.346	23.632.134
	Delta	22.072.834	13.948.412

Fig. 40: Summary results on the test dataset (reporting years 1997 and 1998)

50% greater, between 63M and 68M. In 1998, this delta reach about 150%! Actually, this is not surprising, if we take into account the performance of both k-nearest neighbors and classification tree in predicting closing delay (see Subsection 5.2). Basically, the ensembles correctly predict amounts, but they tend to allocate them in wrong closing delay categories. More specifically, some claims are allocated by lower delays - 0 and 1 - whereas they should be allocated by higher delays - 2 and 3. Additionally, we do not have got many actual claims by higher delay categories yet, so this bias is not balanced out by them, as opposed to the reporting years 1993-1996. However, this happens in those years as well, if we separate claims by reporting year. In Figure 41-44), this effect is represented by the gap between the dashed lines (actual cumulative amount payed) and the four colored lines, representing the four ensembles.

On the one hand, the low-closing-delay payments tend to be overestimated, but the overall amount per reporting year still converges to the correct one. As it's said, keeping in mind that a prudence margin is always required, such ensembles may be still used for reserving purposes. On the other hand, it would be legitimate to let the company reduce such a prudence margin. Ideally, it should own enough data to get better predictions for the closing delay, in order to reduce the gaps shown in Figure 41-44.

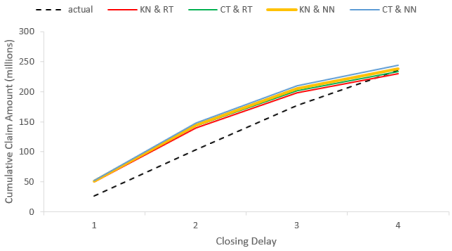

Fig. 41: Cumulative amount in 1993

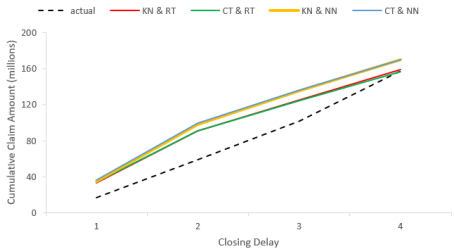

Fig. 42: Cumulative amount in 1994

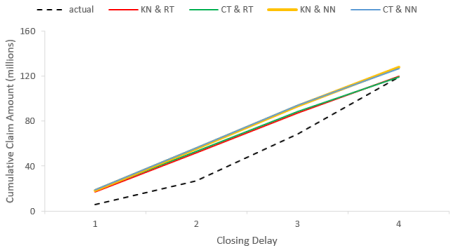

Fig. 43: Cumulative amount in 1995

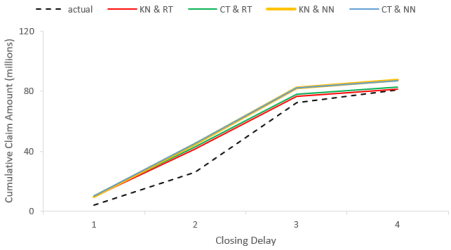

Fig. 44: Cumulative amount in 1996

6. Limitations, extensions, and conclusions

In this paper, we demonstrated the great potential of machine learning in solving a traditional actuarial problem such as claim reserving in non-life. Nonetheless, we focused on a specific application to automobile bodily injury claim data. As a consequence, no general conclusion may be drown (which is typical when it comes with machine learning), but of course the ensembles we used can be easily adapted to different datasets. And this is possible thank to the unique flexibility of these tools.
On the other hand, we should point out some major drawbacks we faced during the analysis:

- data availability is the crucial constraint, and it may happen that a dataset is extremely useful for some target variable, while being very poor for others (for instance, compare closing delay performance in Subsection 5.2 and payment amount performance in Subsection 5.3);

- poor predictions for the closing delay is not a problem if claim reserves are evaluated as at REPORTING year, but it would if they were eval-

uated as at PAYMENT year (this is actually another way to state the problem we faced in Subsection 5.4);

- a complete analysis should include greater closing delays, whose claims tend to be as rare as severe and expensive, as well as zero-amount claims, which would reduce the total reserve, but we couldn't because of data availability issues (see Subsection 5.1);

- IBNYR reserve might be a relevant component of the total claim reserve, but it is not considered in this paper (see Section 2 for further details).

As regards the last bullet point, we should probably build a complete different model for IBNYR prediction. And it would not be strictly "individual", because the company does not get any individual claim data before the reporting date of the claim itself. If we want to use machine learning techniques for this purpose, we should rather rely on different data. More specifically, cross-sectional data related to the policyholder (age, family, address, habits, etc.), and external information like economic environment (unemployment rate, inflation rate, financial distress, etc.), weather conditions, natural catastrophes (storm, flood, earthquake, etc.), and so on. Of course, such additional data may be useful for better prediction of the RBNYS reserve as well.
A last remark is important to conclude the paper. Even if data is materially informative, traditional parametric methods could still outperform nonparametric tools, or return comparable results. For instance, see the payment amount predictions for closing delays 0 and 1 in Figure 31-34. Actually, gamma regression predictions seem more accurate than those of regression tree and neural network. That's just an example of the fact that machine learning is NOT generally superior as compared to traditional methods. The strength of machine learning relates to the ability to catch intricate dependencies among data, which is however not always necessary. More importantly, a greater effort in predicting such dependencies could paradoxically lead to worse performance on regular data. This is also clear in Figure 31-34: better predictions for rare, severe claims (closing delays 3 and 4), but slightly worse predictions for numerous, regular claims (closing delays 0 and 1). In fact, if data is actually regular enough to fulfill the regression assumptions about residuals and multicollinearity, there is no reason to use other tools: regression would return the best performance by definition.
In spite of it, it is worth noting how these new methods can be convenient for

non-life companies. Although we only used basic machine learning tools and combined them together, we have still got very accurate predictions per reporting year. The process may be improved and automatized, little by little, until traditional triangle-based models will be completely dismissed. This evolution is going to outcome two major results: instantaneous, automatic, accurate reserve estimation, and a brand new field for non-life reserving actuaries.

Acknowledgements. The author wishes to thank his promoters and supervisors Prof. Susanna Levantesi (University "La Sapienza", Rome) and Dr. Joseph Lo (Institute and Faculty of Actuaries, London) for their continuous support and precious feedbacks since the very first draft of this paper.

References

[1] K. Antonio, R. Plat, *Micro-level stochastic loss reserving for general insurance*, Scandinavian Actuarial Journal 2014/7, 649-669, 2014.

[2] R. L. Bornhuetter, R. E. Ferguson, *The actuary and IBNR*, Proceedings of the Casualty Actuarial Society 59, 181-195, 1972.

[3] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone, *Classification and Regression Trees*, Wadsworth Statistics/Probability Series, 1984.

[4] P. De Jong, G. Z. Heller, *Generalized linear models for insurance data*, Cambridge University Press, 2008.

[5] C. Dutang, A. Charpentier, *Package 'CASdatasets'*, 4-5, 2016.

[6] T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning. Data Mining, Inference, and Prediction*, Springer Series in Statistics, 2009.

[7] M. Hiabu, C. Margraff, M. D. Martinez-Miranda, J. P. Nielsen, *The link between classical reserving and granular reserving through double chain ladder and its extensions*, British Actuarial Journal 21/1, 97-116, 2016.

[8] G. James, D. Witten, T. Hastie, R. Tibshirani, *An Introduction to Statistical Learning. With Applications in R*, Springer Texts in Statistics, 2015.

[9] A. H. Jessen, T. Mikosch, G. Samorodnitsky, *Prediction of outstanding payments in a Poisson cluster model*, Scandinavian Actuarial Journal 2011/3, 214-237, 2011.

[10] K. Larsen, *Generalized naive Bayes classifiers*, SIGKDD Explorations, 7(1), 76-81, 2005.

[11] T. Mack, *Distribution-free calculation of the standard error of chain ladder reserve estimates*, ASTIN Bulletin 23/2, 213-225, 1993.

[12] M. D. Martinez-Miranda, J. P. Nielsen, R. J. Verrall, M. V. Wüthrich, *The link between classical reserving and granular reserving through double chain ladder and its extensions*, Scandinavian Actuarial Journal 2015/5, 383-405, 2015.

[13] M. Pigeon, K. Antonio, M. Denuit, *Individual loss reserving with the multivariate skew normal framework*, ASTIN Bulletin 43/3, 399-428, 2013.

[14] G. Shmueli, N. R. Patel, P. C. Bruce, *Data Mining for Business Intelligence*, Wiley, 160-240, 2010.

[15] G. Taylor, G. McGuire, J. Sullivan, *Individual claim loss reserving conditioned by case estimates*, Annals of Actuarial Science 3/1-2, 215-256, 2008.

[16] R. Trippi, E. Turban, *Neural networks in finance and investing*, McGraw-Hill, 1996.

[17] R. J. Verrall, M. V. Wüthrich, *Understanding reporting delay in general insurance*, Risks 4/3, 25, 2016.

[18] M. V. Wüthrich, *Machine learning in individual claims reserving*, Swiss Finance Institute, Research Paper Series 16-67, 2016.

[19] M. V. Wüthrich, *Neural networks applied to Chain-Ladder reserving*, SSRN Manuscript ID 2966126, 2017.

[20] M. V. Wüthrich, C. Buser, *Data analytics for non-life insurance pricing*, ETH Zurich, Lecture Notes, 2016.

[21] M. V. Wüthrich, M. Merz, *Stochastic claims reserving manual: advances in dynamic modeling*, SSRN Manuscript ID 2649057, 2015.