Machine Learning, Regression Models, and Prediction of Claims Reserves

Mathias Lindholm, Richard Verrall, Felix Wahl and Henning Zakrisson

Abstract

The current paper introduces regression based reserving models that allow for separate RBNS and IBNR reserves based on aggregated discrete time data containing information about accident years, reporting years, and payment delay, since reporting. All introduced models will be closely related to the cross-classified over-dispersed Poisson (ODP) chain-ladder model. More specifically, two types of models are introduced (i) models consisting of an explicit claim count part, where payments, in a second step, are modelled conditionally on claim counts, and (ii) models defined directly in terms of claim payments without using claim count information. Further, these general ODP models will be estimated using regression functions defined by (i) tree-based gradient boosting machines (GBM), and (ii) feed-forward neural networks (NN). This will provide us with machine learning based reserving models that have interpretable output, and that are easy to bootstrap from. In the current paper we will give a brief introduction to GBMs and NNs, including calibration and model selection. All of this is illustrated in a longer numerical simulation study, which shows the benefits that can be gained by using machine learning based reserving models.

Keywords. Claims reserving, Reported But Not Settled Claims, Incurred But Not Reported Claims, Gradient Boosting Machines, Neural Networks

1. INTRODUCTION

Claims reserving is a major component of the assessment of e.g. solvency and capital adequacy. The uncertainty of the reserves has become an important part of this process, and many statistical models and approaches have been proposed in the literature. Underpinning all of these are the assumptions about the claims processes, and they are often used in conjunction with management information and intervention. This may require judgments to be made about how much to trust the data, what to change from what the data tells you before simulating possible future outcomes, and so on. In this paper, we seek to explore some of these issues using different regression based reserving models combined with machine learning techniques. In order to assess how well these approaches work, we have used the neural network based simulation machine calibrated to Swiss insurance data introduced in [12].

To be more specific, the methodological starting point of the present paper is regression based reserving models that allow for separate reported but not settled (RBNS) reserves and incurred but not settled (IBNR) reserves that make use of count data. Examples of such models are the ones introduced in [26, 22, 27, 17]. Something all these models have in common is that they rely on a description of detailed claim dynamics, but the resulting reserve predictors may be defined only in terms of incremental yearly accident year and development year data for payments and number of reported claims. In particular, the assumptions about individual claim dynamics result in aggregated incremental payments that can be expressed as *conditional* general(ized) linear models, conditional on observed claim counts. Intuitively this means that the observed claim counts will function as exposures. It is also important to stress that due to the detailed constructive motivations used in [26, 22, 27, 17] all model parameters have clear interpretations that are easy to communicate to non-actuaries. Further, due to the conditional linear model structure it is possible to estimate all parameters using e.g. (quasi-) likelihood theory or generalized least squares techniques. Without going into details, the models from [26, 22, 27, 17] allow us to produce

- RBNS reserves, by predicting future remaining payments stemming from the already observed claim counts,
- IBNR reserves, by first predicting the expected number of IBNR claim counts, and then assuming that claim payments behave like the previously described RBNS payments.

In the paper [27], payments are assumed to occur according to a discrete time Poisson process, which allows us to obtain an over-dispersed Poisson (ODP) model for the incremental payments, together with an ODP model for the claim counts. If one wishes, this latter model may be parametrized so that it coincides with a standard cross-classified Poisson chain-ladder model, see e.g. [24]. Thus,

- the model from [27] may be expressed in terms of two standard ODP regression models,
- separate RBNS and IBNR reserves are straightforward to produce,
- all parameters have meaningful interpretations that can be communicated to non-experts.

The main contribution of the present paper is that we will use the ODP regression formulation of the model from [27] together with the underlying modelling ideas to formulate regression based machine learning reserving models that

(a) produce separate RBNS and IBNR reserves,

(b) are interpretable and, to some extent, possible to combine with expert judgement,

(c) are easy to bootstrap.

The methodological focus will be on tree-based gradient boosting machines (GBM) and neural networks (NN), which will be given a brief introduction in this paper, mainly following the exposition in [7]. The overall ambition is to provide a smooth transition from using more standard Poisson regression type models to more complex algorithmic versions, and to describe how these more complex methods may be calibrated, evaluated, interpreted and compared with more standard models. With this said, our focus will primarily be on how to define models and combine these with complex algorithmic regression functions, and how these can be estimated, discussing underlying principles. All numerical illustrations will be based on simulated data produced using a neural network model calibrated to Swiss insurance data, see [12], which will make all our results reproducible.

The disposition of the paper is as follows. Section 1.1 gives a literature overview, Section 2 introduces the regression based reserving models that will be used, and which will be combined with machine learning methods in Section 3. In Section 4 a longer simulation study is used to illustrate how the regression based machine learning models can be calibrated and how to evaluate their performance. Concluding remarks are given in Section 5, and supplementary analyses and technical remarks are given in the appendix.

1.1 Literature overview

The modelling approach taken in the present paper can be thought of as being on an intermediate level compared with pure "macro" models, such as chain-ladder technique models (e.g. [19, 24]), and pure "micro" (individual claim) models, such as point process models (e.g. [2, 23, 16, 1]). For a deeper discussion on pros and cons with using individual level information in claims reserving, see e.g. [1, 25]. The current paper is primarily focused on modelling based on information on the level used in [4], where not only information about accident years and reporting years is used, but also information concerning payment delay since reporting – still, working in discrete time. We believe that this level of granularity is the first natural extension when moving away from pure (discrete-time) macro models. Further, from a practical point of view, this level of information is easy to deal with, still corresponding to standard grouping and aggregation of data, which is much less detailed and complex

to handle than continuous-time micro level data. Due to this, the modelling approach taken here will remain close to those used for standard macro reserving models, and in particular be close to the cross-classified ODP (ccODP) chain-ladder model, see [24].

When turning to machine learning techniques, focus will be on tree-based gradient boosting machines (GBM), see e.g. [9], and feed-forward neural networks (NN), see e.g. [13], and general references that cover both techniques (and much more) are e.g. [14, 7]. Both of these techniques belong to the class of "supervised learning" methods that can be thought of as methods to perform flexible curve (or surface) fitting, subject to a chosen loss function. This makes these methods natural to combine with the ODP regression models we have in mind, since given an ODP model structure, flexible machine learning based regression functions can be used to approximate the true, unobservable mean function, by choosing an ODP likelihood as loss function. How this is can be done in general for GBMs and NNs is discussed in e.g. [7, Ch. 17.3, Ch. 18]. When turning to reserving, [11] uses this idea and introduces a feed-forward neural network extension of the ODP chain-ladder model from [24], and describe how to calibrate and "train" the NN based on partially observed claims data. See also [30] for a condensed discussion on how to combine GLMs and NNs. As mentioned previously, there is a close connection between the ccOPD chain-ladder model and the so-called collective reserving model (CRM) from [27], which will make it natural to use similar techniques in the current setting. Other approaches working with deep neural networks on aggregated data can be found in e.g. [29] which discusses an extension of Mack's distribution-free chain-ladder model, [15] which uses aggregated data and gated recurrent units, and [10] which is closer to the ccODP model but also include a neural network model for count data. An example of an individual loss reserving model using deep neural networks is given in [28]. For examples of other machine learning techniques applied to individual loss reserving, see e.g. [18, 5] which uses trees, and [6] which uses tree-based gradient boosting machines. For a more general discussion of individual loss reserving and machine learning techniques, see e.g. [25] and the references therein.

2. REGRESSION BASED RESERVING MODELS

The basic building block in the present paper is regression models, and in particular over-dispersed Poisson (ODP) regression models. That is, if we let $\boldsymbol{Y}$ denote the response of interest, and let $\boldsymbol{c} = (c_1, \ldots, c_p)'$ be an arbitrary covariate vector such that $\boldsymbol{c} \in \boldsymbol{C}$, we then say that $\boldsymbol{Y} \mid \boldsymbol{c} \sim \text{ODP}(\boldsymbol{\mu}(\boldsymbol{c}; \boldsymbol{\beta}), \boldsymbol{\rho})$ with *link-function* $\boldsymbol{u}(\cdot)$, if

$$\boldsymbol{u}(\mathbb{E}[\boldsymbol{Y} \mid \boldsymbol{c}]) = \boldsymbol{\mu}(\boldsymbol{c}; \boldsymbol{\beta}),$$

where $\mu(\boldsymbol{c};\boldsymbol{\beta}) > 0$ is some arbitrary mean function, and

$$\mathbb{E}[Y \mid \boldsymbol{c}] = \text{Var}(Y \mid \boldsymbol{c})/\rho.$$

Note that we will sometimes stress the dependence on the parameters by using the notation $\mathbb{E}[Y \mid \boldsymbol{c};\boldsymbol{\beta}]$ and $\text{Var}(Y \mid \boldsymbol{c};\boldsymbol{\beta},\rho)$.

Remark 1

(a) The "Poisson" part of ODP comes from that if $Y \mid \boldsymbol{c} \sim \text{Po}(\mu(\boldsymbol{c};\boldsymbol{\beta}))$*, it holds that* $\mathbb{E}[Y \mid \boldsymbol{c}] = \text{Var}(Y \mid \boldsymbol{c})$*. From this perspective, the parameter* ρ *introduces an additional degree of freedom which allows for "over-dispersion". For more on ODP models, see e.g. [20] or [8, 3], in an actuarial context.*

(b) ODP models may be estimated using quasi-likelihood theory so that $\mu(\boldsymbol{c};\boldsymbol{\beta})$ *may be estimated with a* standard *Poisson likelihood. The over-dispersion parameter* ρ *may thereafter be estimated using Pearson (or deviance) residuals.*

Concerning reserving models more specifically, as mentioned in the introduction the so-called "collective reserving model" (CRM) from [27] will serve as the starting point for our analyses. Let $X_{i,j}$ denote the total amount of payments from claims that occurred during accident year i, $i = 1,\dots,m$, that where made during development year j, $j = 0,\dots,m-1+d$, where d corresponds to the maximal delay of a payment since the time since reporting. Further, let $N_{i,j}$ denote the total number of claims from accident year i, $i = 1,\dots,m$, that are reported j, $j = 0,\dots,m-1$, years later. Moreover, let $\mathcal{N}_0 := \sigma\{N_{i,j} : i + j \leq m, i = 1,\dots,m, j = 0,\dots,m-1\}$ and $\mathcal{N} := \sigma\{N_{i,j} : i = 1,\dots,m, j = 0,\dots,m-1\}$. This allows us to define our baseline model: This allows us to define our baseline model:

Model 1 *The CRM based on* $X_{i,j}$ *and* $N_{i,j}$ *data from [27] can be written on the form*

$$X_{i,j} \mid \mathcal{N} \sim \text{ODP}\left(\sum_{k=0}^{j\wedge d} \psi_{i,j-k,k} N_{i,j-k},\ \varphi\right),$$

and

$$N_{i,j} \sim \text{ODP}\left(\nu_{i,j},\ \phi\right),$$

that is,

$$\mathbb{E}[X_{i,j} \mid \mathcal{N}] = \sum_{k=0}^{j\wedge d} \psi_{i,j-k,k} N_{i,j-k} = \text{Var}(X_{i,j} \mid \mathcal{N})/\varphi,$$

and

$$\mathbb{E}[N_{i,j}] = \nu_{i,j} = \text{Var}(N_{i,j})/\phi.$$

All $\boldsymbol{X_{i,j}}$ are assumed to be conditionally independent, given $\boldsymbol{\mathcal{N}}$, and all $\boldsymbol{N_{i,j}}$ are assumed to be independent.

Remark 2

(a) Note that $\mathbb{E}[X_{i,j} \mid \mathcal{N}]$ is given by a sum, which makes it impossible to estimate the $\psi_{i,j,k}$s using anything but the identity link-function, whereas the parameters of the $N_{i,j}$ model can be estimated using any link-function.

(b) Going from the micro-level assumptions to this formulation of the CRM relies on a number of assumptions, which can be summarised as that all individual claims occur independently, all claims with the same accident year and reporting year have the same payment dynamics, and all payments occur independently between claims. For more on these assumptions, see [27]. For the purposes of this paper, the important thing is to think about the model, what the parameters mean in practice and how they can be estimated and used to produce reserve estimates etc.

(c) Based on the individual claim dynamics from [27], the interpretation of $\psi_{i,j,k}$ is that it corresponds to the average amount paid for a claim that occurred during accident year i that was reported j years after the accident occurred, and paid k periods after reporting. Thus, $\psi_{i,j,k}$ may be represented as $\psi_{i,j,k} = \mu_{i,j,k}\lambda_{i,j,k}$, where $\mu_{i,j,k}$ denotes the expected size of a single payment for a claim that occurred in year i, was reported j years later, and paid k years after reporting, and $\lambda_{i,j,k}$ denotes the expected number of claim payments in analogy with $\mu_{i,j,k}$. Further, from the definition of Model 1 there is a single *over-dispersion parameter φ. From [27] it follows that φ equals*

$$\varphi := \frac{\sigma^2_{i,j,k} + \mu^2_{i,j,k}}{\mu_{i,j,k}},$$

implying that $\sigma^2_{i,j,k} := \mu_{i,j,k}(\varphi - \mu_{i,j,k}) \geq 0$ must hold, i.e. it is not possible to choose both $\mu_{i,j,k}$ and $\sigma^2_{i,j,k}$ freely – either, will in this sense, determine the other when assuming a constant φ.

(d) Note that by having a single φ which is independent of the indices i, j, and k, means that from a quasi-likelihood perspective, all the $\psi_{i,j,k}$s can (in theory) be estimated using a standard Poisson *likelihood, and in a second step φ may be estimated using e.g. Pearson or deviance residuals.*

(e) *The model for $\boldsymbol{X}_{i,j}$ conditional on $\mathcal{N}$ given by Model 1 corresponds to an ODP model with identity link function expressed in terms of $\psi_{i,j,k}$s, and may be estimated using standard statistical software such as* R*. However, the fully general parametrization in terms of $\psi_{i,j,k}$ does not allow for estimation given $\boldsymbol{X}_{i,j}$. Instead, one is either forced to consider more restricted models, such as*

$$\psi_{i,j,k} = \mu\lambda_{j,k},$$

or

$$\psi_{i,j,k} = \alpha_i\beta_j\gamma_k \tag{1}$$

or to consider data on a more granular level, which we will get back to further on. Moreover, given that the $\psi_{i,j,k}$s have been estimated using maximum quasi-likelihood, to separate these into their constituent components, e.g. $\psi_{i,j,k} = \mu\lambda_{j,k}$, there is need of additional information, such as e.g. an estimate of μ. Similarly, for the count model, the perhaps most common parametrisation is obtained by assuming $\nu_{i,j} = \alpha_i\beta_j$ with $\sum_j \beta_j = 1$ we may estimate the parameters using a log-link function, i.e. $\log(\mathbb{E}[N_{i,j}]) = \widetilde{\alpha}_i + \widetilde{\beta}_j$, which exactly corresponds to the cross-classified over-dispersed Poisson chain-ladder model of [24].

Concerning reserve estimators, let $R_i^{\mathcal{R}}$ denote the outstanding RBNS claim payments for accident year i and let $R_i^{\mathcal{I}}$ denote the corresponding outstanding IBNR claim payments, i.e.

$$R_i := R_i^{\mathcal{R}} + R_i^{\mathcal{I}}.$$

From Eq. (6) – (9) in [27] it follows that

$$h_i^{\mathcal{R}}(\boldsymbol{\psi}; \mathcal{N}_0) := \mathbb{E}\left[R_i^{\mathcal{R}} \mid \mathcal{N}_0\right] = \sum_{j=m-i+1}^{m-1+d} \sum_{k=j-m+i}^{j\wedge d} \psi_{i,j-k,k} N_{i,j-k}, \tag{2}$$

$$h_i^{\mathcal{I}}(\boldsymbol{\psi}, \boldsymbol{\nu}) := \mathbb{E}\left[R_i^{\mathcal{I}} \mid \mathcal{N}_0\right] = \sum_{j=m-i+1}^{m-1+d} \sum_{k=0\vee(j-m+1)}^{(j-m+i-1)\wedge d} \psi_{i,j-k,k}\nu_{i,j-k}, \tag{3}$$

which, by replacing all unknown model parameters with estimators, gives us the following *computable* RBNS- and IBNR-reserve estimators

$$\widehat{R}_i^{\mathcal{R}} := h_i^{\mathcal{R}}(\widehat{\boldsymbol{\psi}}; \mathcal{N}_0) \tag{4}$$

and

$$\widehat{R}_i^{\mathcal{I}} := h_i^{\mathcal{I}}(\widehat{\boldsymbol{\psi}}, \widehat{\boldsymbol{\nu}}). \tag{5}$$

Although Eq. (2) and (3) are slightly more general than the formulation in [27], the proofs are identical, and it is also possible to calculate RBNS and IBNR reserve process variances explicitly following the arguments in [27], see Appendix C.2 for details.

Remark 3

(a) Eq. (2) *and* (3) *are expressed in terms of* $\psi_{i,j,k}$*s whose index combinations have not yet been observed. Consequently, apart from that the general formulation of Model 1 is not possible to estimate (see Remark 2(c) and (e)), the model is neither possible to use for prediction unless further structural assumptions are made such as e.g.* $\psi_{i,j,k} = \mu\lambda_{j,k}$ *or* $\psi_{i,j,k} = \alpha_i\beta_j\gamma_k$.

(b) The aggregate level formulation of the CRM from Model 1 is not sufficiently detailed in order to be able to derive Eq. (2) *and* (3)*. In particular, unless the micro-level assumptions discussed in Remark 2(b) are fulfilled,* it is not possible *to claim that the reserves given by* (4) *and* (5) *actually correspond to proper RBNS and IBNR reserves.*

As commented on in Remark 2(c) and (e) it is not possible to allow freely varying individual claim payment distributions, and one aspect of this (relating to Remark 2(d) and the assumption of a single over-dispersion parameter) is that such models may be impossible to estimate. This, however, is partly a consequence of having too many parameters in relation to modelling based on aggregated $X_{i,j}$ level data, and partly due to that a too flexible parametrization does not allow us to observe all data needed w.r.t. specific (i, j, k)-combinations. Thus, by using slightly less aggregated data, introducing $X_{i,j,k}$, which denotes the total amount of payments from accident year i, that come from claims that are reported with j years delay, and that are paid k years after reporting, it is natural to define the following model:

Model 2 *The CRM based on* $X_{i,j,k}$ *and* $N_{i,j}$ *data from [27] can be written on the form*

$$X_{i,j,k} \mid \mathcal{N} \sim \text{ODP}\left(\psi_{i,j,k}N_{i,j},\ \varphi\right),$$

and

$$N_{i,j} \sim \text{ODP}\left(\nu_{i,j},\ \phi\right),$$

that is,

$$\mathbb{E}[X_{i,j,k} \mid \mathcal{N}] = \psi_{i,j,k}N_{i,j} = \text{Var}(X_{i,j,k} \mid \mathcal{N})/\varphi,$$

and

$$\mathbb{E}[N_{i,j}] = \nu_{i,j} = \text{Var}(N_{i,j})/\phi.$$

All $X_{i,j,k}$ *are assumed to be conditionally independent, given* $\mathcal{N}$*, and all* $N_{i,j}$ *are assumed to be independent.*

But, a closer inspection of the derivations in [27] (as well as in [26, 22, 17]) $X_{i,j,k}$ level data is actually what is used in intermediate steps. This level of granularity has also been used in e.g. [4] and is still considerably less demanding to use than the continuous time micro-data needed in e.g. [23, 1].

Remark 4

(a) Model 2 uses less aggregated data than Model 1, see Remark 2(b), and these two models will not *produce the same parameter estimates. Concerning parametrisations, note that Model 2 can be expressed as*

$$\log(\mathbb{E}[X_{i,j,k} \mid N_{i,j}]) = \log(N_{i,j}) + \widetilde{\psi}_{i,j,k},$$

or

$$\mathbb{E}[X_{i,j,k} \mid N_{i,j}] = N_{i,j}\, \mathrm{e}^{\widetilde{\psi}_{i,j,k}},$$

making it natural to estimate the $\psi_{i,j,k}$s an ODP model with log-link function and the $\log(N_{i,j})$s as offsets, conditionally on the $N_{i,j}$s.

(b) Model 2 will produce the same theoretical RBNS and IBNR reserves as Model 1. The same holds true for the process variances, which are identical with those for Model 1. For more on this, see Appendix C.2.

However, note that by working with $X_{i,j,k}$ data it is no longer necessary to include $N_{i,j}$ count data information in order to be able to produce separate RBNS and IBNR reserves, also recall Remark 3(b), since

$$R_i^{\mathcal{R}} := \sum_{j=0}^{m-i} \sum_{k>m-(i+j)} X_{i,j,k},$$

and

$$R_i^{\mathcal{I}} := \sum_{j=m-i+1}^{m-1} \sum_{k} X_{i,j,k}.$$

This last observation suggests to introduce the following model:

Model 3 *Let $X_{i,j,k}$ be defined as an over-dispersed model according to*

$$X_{i,j,k} \sim \mathrm{ODP}\left(\psi_{i,j,k},\ \varphi\right),$$

with

$$\mathbb{E}[X_{i,j,k}] = \psi_{i,j,k} = \mathrm{Var}(X_{i,j,k})/\varphi,$$

where all $X_{i,j,k}$ are assumed to be independent.

Note that Model 3 does not only have very simple RBNS and IBNR reserve predictors, being sums of suitably indexed $\psi_{i,j,k}$s, but also the corresponding process variances are easily obtained due to that all $X_{i,j,k}$s are assumed to be independent. This independence assumption may, of course, be questioned, since in reality the underlying claims dynamics should correspond to something similar to the micro-level assumptions that underpin the CRM model (Model 1). Nevertheless, Model 3 defines a flexible statistical model in its own right, and whose performance will be evaluated in Section 4. Still, as for Model 1 and Model 2, Model 3 will need additional parameter restrictions in order to be able to be used in practice.

Before proceeding further, note that Model 1 - Model 3

(i) have enough flexibility to be able to capture e.g. "inflation" or "calendar year" effects using e.g. $\psi_{i,j,k} := \mu_{i+j}\lambda_{j,k}$ (although these parameters may be hard to estimate in practice due to the number of parameters, and hard to extrapolate for prediction purposes),

(ii) are straightforward to bootstrap, see Appendix C.2 for details.

Concerning the problems with estimating $\psi_{i,j,k}$ s and extrapolating these for reserve predictions, these problems can at least partly be addressed by replacing the $\psi_{i,j,k}$ s with general functional forms that can be estimated using regression based machine learning techniques.

3. MACHINE LEARNING BASED RESERVING MODELS

In Section 2 three different over-dispersed Poisson models have been introduced and discussed, Model 1 - Model 3. All of the models are flexible w.r.t. parametrisation and estimation, and may be used to produce separate RBNS and IBNR reserves using simple formulas. Still, the models' most general formulations will have too many parameters for reliable estimation, but the general $\psi_{i,j,k}$ parametrisation will not lend itself to extrapolation, see Remark 2(c) and (e), and Remark 3(a). Moreover, even though it would be possible to estimate all $\psi_{i,j,k}$ s, these estimators will likely be unstable. We may instead use functional forms, but then the question is, *which* functional forms will serve as appropriate approximations.

In the current section these questions will be addressed by using tree-based gradient boosting machines (GBM) and neural network (NN) regression functions combined with the ODP models from Section 2.

We will start by discussing these questions in a reserving context without explic-

itly stating any particular machine learning technique. The specific GBM and NN estimated models will be discussed in Section 3.3 and 3.4 below.

3.1 General considerations

The general focus will be on reserving models based on $X_{i,j,k}$-level data with regression functions $\psi(i, j, k; \boldsymbol{\theta})$ and $\nu(i, j; \boldsymbol{\theta})$ corresponding to general functional forms defined in terms of a parameter vector $\boldsymbol{\theta}$, which we will try to approximate using complex algorithmic techniques. Further, all models that we will consider will be ODP models parametrised using log-link functions, unless stated otherwise. In practice, this simply means that the $\psi_{i,j,k}$s and $\nu_{i,j}$s from Section 2 are defined according to $\psi_{i,j,k} := \psi(i, j, k; \boldsymbol{\theta})$ and $\nu_{i,j} := \nu(i, j; \boldsymbol{\theta})$, and that a specific link-function is being used. That is, Model 2 may be parametrized according to

$$\begin{cases} X_{i,j,k} \mid N_{i,j} & \sim \text{ODP}(\log(N_{i,j}) + \psi(i, j, k; \boldsymbol{\theta}),\ \varphi), \\ N_{i,j} & \sim \text{ODP}(\nu(i, j; \boldsymbol{\theta}),\ \phi), \end{cases}$$

where

$$\mathbb{E}[X_{i,j,k} \mid N_{i,j}] = N_{i,j} \exp\{\psi(i, j, k; \boldsymbol{\theta})\},$$

(recall Remark 4(a)) and

$$\mathbb{E}[N_{i,j}] = \exp\{\nu(i, j; \boldsymbol{\theta})\}.$$

Similarly, Model 3 can be parametrized as

$$X_{i,j,k} \sim \text{ODP}(\psi(i, j, k; \boldsymbol{\theta}),\ \varphi),$$

with

$$\mathbb{E}[X_{i,j,k}] = \exp\{\psi(i, j, k; \boldsymbol{\theta})\}.$$

These parametrizations of Model 2 and Model 3 are very close to the regression function parametrizations that will be introduced and used when fitting the GBMs and NNs discussed in Section 3.2. Note that using this specific parametrization will not change the interpretability of e.g. ψ, see Remark 2(c). Furthermore, this fact will also hold when these regression functions are estimated by using the functional forms described by the machine learning methods. Consequently, due to the possibility to interpret the underlying model structure, it will be possible to include expert opinion into this type of machine learning based regression models, e.g. by scaling estimated ψs based on expert opinions.

At this point we have not yet discussed any particular machine learning techniques in detail, but the overarching idea is that these techniques will allow us to

model complex functional relationships such as higher order non-linear relationships between covariates without explicitly stating *how* these should look. In general these methods involve a lot of parameters defining the functional forms that are being used to approximate the true regression functions (ψ and ν). Due to this, it is of great importance to discuss how to avoid overfitting, which is done in the next section.

3.2 Estimation and calibration of machine learning models

Before going into details on specific machine learning methods and reserving, let us take a step back to the general regression setting that was starting point of Section 2. That is, let $Y \mid \boldsymbol{c} \sim \text{ODP}(\mu(\boldsymbol{c};\boldsymbol{\beta}), \rho)$ where $\mu(\boldsymbol{c};\boldsymbol{\beta})$ is some unknown functional form that we want to approximate, which in particular means for a specfic link-function $u(\cdot)$ it holds that

$$\mu(\boldsymbol{c};\boldsymbol{\beta}) = u(\mathbb{E}[Y \mid \boldsymbol{c};\boldsymbol{\beta}]).$$

Moreover, recall from Remark 1 that when estimating the mean function in an ODP model it is possible to do this separately from the over-dispersion parameter ρ by using a standard Poisson-likelihood. Consequently, what we will do next is to use machine learning techniques to

- approximate the unknown $\mu(\boldsymbol{c};\boldsymbol{\beta})$ function by using a flexible function class $f(\boldsymbol{c};\boldsymbol{\gamma})$,
- do the optimization of $f(\boldsymbol{c};\boldsymbol{\gamma})$ using a loss function corresponding to a Poisson likelihood.

The machine learning techniques to be discussed below will introduce flexible functional forms $f(\boldsymbol{c};\boldsymbol{\gamma})$, where the parameter vector $\boldsymbol{\gamma}$ will be (potentially very) high-dimensional and it is, therefore, important to avoid overfitting. The general approach to tackle overfitting of complex algorithmic methods is to use out-of-sample validation, see e.g. [7, Ch. 12]. This is done by splitting the available data into two sub-sets: one used for training ("parameter estimation") and one used for validating the predictive performance of the trained model. Further, the numerical procedures that will be introduced below to estimate $\boldsymbol{\gamma}$ will be based on iterative numerical updating procedures. A problem with these procedures when it comes to complex algorithmic methods is that, unless you stop the iterative numerical procedure used for approximating $\mu(\boldsymbol{c};\boldsymbol{\beta})$ (or in our reserving models the functions $\psi(\cdot)$ and $\nu(\cdot)$) early, you will likely end up with a fully saturated model - a model with a unique parameter for each observed data point. In the extreme situation $\boldsymbol{\gamma}$ may have more parameters than observations (which will be the case in our reserving situation discussed in

Section 4). This premature stopping is what is often referred to as "early stopping", and can be thought of as an informal regularization technique, see e.g. [7, Ch. 18.2]. Consequently, what often is done in practice, is that the number of optimization steps used are determined based on the trained model's out-of-sample performance based on the validation data. Unless done properly, from a statistical perspective this may result in a kind of cheating – you are *not* allowed to *estimate* a parameter, in this context the number of iterations, based on validation data. Still, given a sufficient amount of data you can always split your data set into three parts: one part used for training, one part used for determining the number of steps that the optimization procedure should be run, corresponding to a pseudo-validation data set, and a final third sub-set used for proper out-of-sample validation once the model has been properly trained (without any prior peeking!). This procedure allows us to evaluate different parametrizations and configurations of our machine learning models and serves as a generic way of doing model selection. Further, since the models of interest belong to the family of ODP models, it is natural to evaluate the performance w.r.t. deviance residuals. This corresponds to that the parameter vector $\boldsymbol{\gamma}$ is optimized w.r.t. the standard ODP likelihood loss function. That is, this choice corresponds to standard maximum likelihood optimization, but w.r.t. complex parametric function $f(\boldsymbol{c}; \boldsymbol{\gamma})$.

When trying to implement these ideas in a reserving context the first problem is that we ideally would like to have fully developed payment data to be used to evaluate the prediction of outstanding claim payments. This is rarely the case. In order to at least partly circumvent this issue we adopt the procedure from [11]:

(i) Split the *un-aggregated* individual claim data into two parts,

(ii) Depending on the model that you want to train, aggregate the separate individual data sets into $\boldsymbol{X}_{i,j}$, $\boldsymbol{X}_{i,j,k}$, and $\boldsymbol{N}_{i,j}$ form. Let $\boldsymbol{X}_{i,j}^{(t)}$ denote "training" data and let $\boldsymbol{X}_{i,j}^{(v)}$ denote "validation" data, and analogously for $\boldsymbol{X}_{i,j,k}$ and $\boldsymbol{N}_{i,j}$.

(iii) Train your models on $\boldsymbol{X}_{i,j}^{(t)}$, $\boldsymbol{X}_{i,j,k}^{(t)}$, and $\boldsymbol{N}_{i,j}^{(t)}$ data, validate the model performance based on using $\boldsymbol{X}_{i,j}^{(v)}$, $\boldsymbol{X}_{i,j,k}^{(v)}$, and $\boldsymbol{N}_{i,j}^{(v)}$ data.

Remark 5

(a) The above algorithm is only described in terms of splitting the historic data in two, but you may, of course, make another implicit split of the data set used to construct e.g. $\boldsymbol{X}_{i,j}^{(t)}$ *to be able to carry out the above described three step procedure.*

(b) *By using the above algorithm we will train our models based on observed in-sample data, and we will use another set of in-sample data not used for training to construct "out-of-sample" validation data. This is most likely sub-optimal, but the best we can hope for.*

(c) *It is important that the mechanism used for splitting the original data sets into sub-sets does not create severe imbalances in terms of overall exposures per accident year. The procedure used in [11] corresponds to a random 50 / 50 split of the original individual claim data.*

We will return to Remark 5(b) when discussing GBM and NN models in more detail.

Next, we will describe two specific choices of functions $f(\boldsymbol{c};\boldsymbol{\gamma})$ using GBMs and NNs.

3.3 Tree-based gradient boosting machines

The current section will give a brief introduction to regression trees and tree based gradient boosting. A comprehensive introduction to these subjects can be found in e.g. [14, 7], which form the basis for the current exposition. More technical parts of the exposition can be found in Appendix C.1.

To start off, we consider the situation with data is in the form of pairs $(y_i, \boldsymbol{c}_i)$, $i = 1, \ldots, m$, where y_i corresponds to the observed response i with corresponding covariate vector $\boldsymbol{c}_i := (c_{i,1}, \ldots, c_{i,p})' \in \boldsymbol{C}$. As a notational convention all references to $\boldsymbol{c} = (c_1, \ldots, c_p)'$ are with respect to a general covariate vector. In the current paper we will focus on binary regression trees. A binary regression tree is, like any other regression model, a model that takes a covariate vector $\boldsymbol{c}$ and parameter vector $\boldsymbol{\gamma}$ as input and produces an expected value of $Y \mid \boldsymbol{c}$ as output. More specifically, we will focus on ODP models, which means that given a link-function $u(\cdot)$, we let

$$r(\boldsymbol{c};\boldsymbol{\gamma}) := u(\mathbb{E}[Y \mid \boldsymbol{c};\boldsymbol{\gamma}]),$$

and where the distribution of $Y \mid \boldsymbol{c}$ will imply a specific loss function to be used when estimating $\boldsymbol{\gamma}$ – in our case a Poisson likelihood. More specifically, in its simplest form, a binary regression tree of "depth" k will partition $\boldsymbol{C}$ into 2^k different regions $\mathcal{A}_j$, $j = 1, \ldots, 2^k$, where each region is assigned a single value δ_j, which means that $r(\boldsymbol{c};\boldsymbol{\gamma})$ may be represented according to

$$r(\boldsymbol{c};\boldsymbol{\gamma}) := \sum_{j=1}^{2^k} \delta_j \mathbb{1}_{\{\boldsymbol{c}\in\mathcal{A}_j\}}, \tag{6}$$

where $\boldsymbol{\gamma}$ is the vector containing all δ_js and all additional parameters needed to define the $\mathcal{A}_j$s. Consequently, given that the $\mathcal{A}_j$s are known, in our ODP-regression situation, we would get that the δ_js are obtained by optimizing

$$\delta_j := \arg\min_{\delta} \sum_{i:\boldsymbol{c}_i \in \mathcal{A}_j} L(y_i, r(\boldsymbol{c}_i; \boldsymbol{\gamma})),$$

where

$$r(\boldsymbol{c}_i; \boldsymbol{\gamma}) = \delta_j, \qquad i : \boldsymbol{c}_i \in \mathcal{A}_j,$$

and where $\boldsymbol{L}$ corresponds to the Poisson log-likelihood function[1]. Further, to see how to construct the $\mathcal{A}_j$s, we will use the "tree" interpretation of this regression procedure. That is, the reason why this is referred to as a regression "tree" is because the procedure to arrive at the 2^k partitioning splits of $\boldsymbol{C}$ can be represented as a binary decision tree consisting of k binary decisions, where each decision is based on a single component of the covariate vector $\boldsymbol{c}$. That is, the $\mathcal{A}_j$ regions can be thought of as being constructed using a sequence of k binary decisions expressed in terms of $\mathcal{A}_{l,j}, l = 1, \ldots, 2^j, j = 1, \ldots, k$, and the $\mathcal{A}_j$ regions are often referred to as "leaves". An example of a binary decision tree of depth 2 is shown in Figure 1, where $\boldsymbol{c} = (c_1, c_2)$ and where

$$\mathcal{A}_{l,j} := \mathcal{A}(\pi_{l,j}, \kappa_{l,j}) = \{c_{\pi_{l,j}} \leq \kappa_{l,j}\}.$$

Further, Figure 1 implies that it will be computationally complex to estimate all δ_js and all $\mathcal{A}_{l,j}$s simultaneously. As a consequence of this, a so-called "greedy" algorithm is used. This is a sequential algorithm where the tree is allowed to grow from depth 1 to k, where the $\mathcal{A}_{l,j}$s, together with the associated δ_js, are optimized for the *next* depth level of the tree. That is, given that you have reached depth s of the tree, you *only* optimize the $\mathcal{A}_{s,j}$s and the associated δ_js as if these would be the terminal depth of the tree, leaving the previously estimated $\mathcal{A}_{l,j}, l < s$, fixed. This depth-by-depth level optimization is continued until you reach depth k, which gives you your final $\mathcal{A}_j$ and δ_js. Note that in practice $\mathcal{A}_{l,j} = \mathcal{A}(\pi_{l,j}, \kappa_{l,j}) = \{c_{\pi_{l,j}} \leq \kappa_{l,j}\}$, meaning that you optimize w.r.t. *which* covariate component to base your split on ($\pi_{l,j}$), and *which* covariate threshold to base your split on ($\kappa_{l,j}$). This procedure allows the same covariate to appear on different levels in the same tree, which is seen in Figure 1. Also note that each $\mathcal{A}_{l,j}$ is associated with *two* δ-values, depending on which binary decision that was made based on $\mathcal{A}_{l,j}$. For more on this, see Appendix C.1 and the previously mentioned references.

The above described procedure for how to construct a tree will always produce a tree with 2^k leaves. It is, however, common to use so-called "pruning", where the

[1]Note that the objective function used for optimizing δ_j corresponds to the sample average estimator of $\mathbb{E}[L(Y, r(\boldsymbol{c}; \boldsymbol{\gamma})) \mid \boldsymbol{c}]$.

δ-values associated with leaves estimated based on e.g. few observations are removed. Further, trees, as the one depicted in Figure 1, are easy to interpret in terms of the $\mathcal{A}_j$ regions, and trees allow for non-trivial interactions between the covariates. Moreover, trees scale nicely when having many covariates, i.e. $\boldsymbol{c} = (c_1, \ldots, c_p)$ with large p, but trees that are estimated with large depth may be unstable w.r.t. overfitting, introducing large estimation error variance. Due to this, techniques such as random forests and bagging are used, which are based on averaging trees of lower depth, where not all covariates are allowed to be used in each optimization step when building the trees.

Another alternative to building a single complex tree-based regression model is to use a sequential learning procedure known as gradient boosting. Unlike forests which average a large number of smaller (approximately uncorrelated) regression trees, tree-based gradient boosting instead uses trees of low depth, e.g. $k \leq 10$, to approximate the pointwise gradient of the loss function given the current fit. That is, let $\widehat{G}_0(\boldsymbol{c}) = 0$, and for each $b, b = 1, \ldots, \beta$, calculate

$$g_i = -\frac{\partial}{\partial z} L(y_i, z)\Big|_{z=\widehat{G}_{b-1}(\boldsymbol{c}_i)}, i = 1, \ldots, m,$$

and fit a tree with low depth to these gradients according to

$$\widehat{\gamma}_b = \arg\min_{\gamma} \sum_{i=1}^{m} (g_i - r(\boldsymbol{c}_i; \gamma))^2.$$

The fit is thereafter updated according to

$$\widehat{G}_b(\boldsymbol{c}) = \widehat{G}_{b-1}(\boldsymbol{c}) + \epsilon r(\boldsymbol{c}; \widehat{\gamma}_b),$$

where $\epsilon > 0$ corresponds to the so-called shrinkage factor or learning rate, which after β iterations results in

$$\widehat{G}_\beta(\boldsymbol{c}) := G_\beta(\boldsymbol{c}; \epsilon, \widehat{\gamma}_b, b = 1, \ldots, \beta) = \epsilon \sum_{b=1}^{\beta} r(\boldsymbol{c}; \widehat{\gamma}_b).$$

The above described procedure corresponds to Algorithm 17.4 in [7], and this means that our gradient boosting machine predictor $f^{\text{GBM}}(\boldsymbol{c}; \widehat{\gamma})$ is given by

$$f^{\text{GBM}}(\boldsymbol{c}; \widehat{\gamma}) := \epsilon \sum_{b=1}^{\beta} r(\boldsymbol{c}; \widehat{\gamma}_b).$$

In Section 4 the `R`-package `gbm` will be used, and we refer to the documentation of this software for the precise implementation of the exact GBM configuration that is

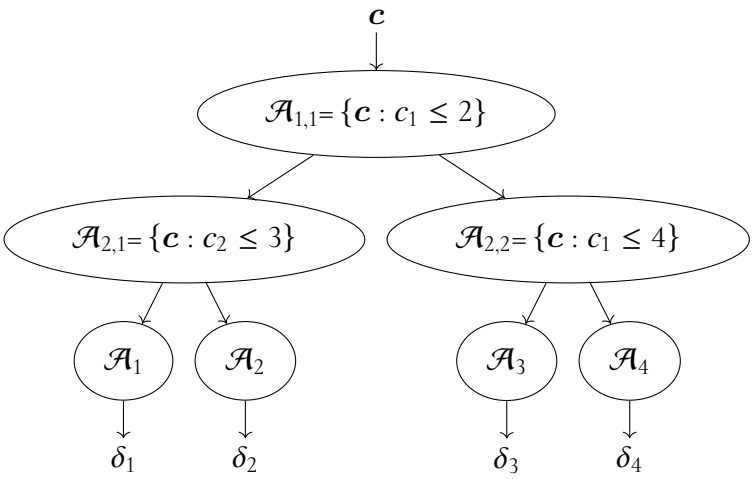

Figure 1: Example of tree of depth 2, where $\mathcal{A}_1 = \{\boldsymbol{c} : \boldsymbol{c} \in \mathcal{A}_{1,1} \cap \mathcal{A}_{2,1}\} = \{\boldsymbol{c} : \boldsymbol{c} \in \{c_1 \leq 2, c_2 \leq 3\}\}$, and analogously for $\mathcal{A}_2$, $\mathcal{A}_3$, and $\mathcal{A}_4$.

being used. Further, as discussed in Section 3.2, how to choose the learning rate ϵ, the depth of the trees used for gradient approximations, and the number of iterations β, can be done by splitting the data into training and validation sets. We will return to this in more detail in Section 4.

3.3.1 Reserving models

From the general definition of Model 2, using log-link functions according to the discussion on parametrizations in Section 3.1 and Remark 4(a), it follows that its GBM estimated version will produce the following estimates

$$\begin{cases} \widehat{\mathbb{E}}[X_{i,j,k}|N_{i,j}] &= N_{i,j}\exp\{f_{\psi}^{\text{GBM}}(i,j,k;\widehat{\boldsymbol{\theta}})\} \\ &= N_{i,j}\exp\{\epsilon\sum_{b=1}^{\beta} r_{\psi}(i,j,k;\widehat{\gamma}_b)\} \\ \widehat{\mathbb{E}}[N_{i,j}] &= \exp\{f_{\nu}^{\text{GBM}}(i,j;\widehat{\boldsymbol{\theta}})\} \\ &= \exp\{\epsilon\sum_{b=1}^{\beta} r_{\nu}(i,j;\widehat{\gamma}_b)\} \end{cases} \qquad \text{(M2 - GBM)}$$

and where $\widehat{\varphi}$ and $\widehat{\phi}$ corresponds to estimates based on e.g. Pearson or deviance residuals. Note that this is all the information that is needed in order to produce separate RBNS and IBNR reserves based on Eq. (4) and (5). Further, when it comes to bootstrapping the GBM-estimated version of Model 2, we will use Model 2, but using the "plug-in" estimates from (M2 - GBM). For more details on this, see Appendix C.2.

Analogously, the GBM estimated version of Model 3 will produce

$$\begin{aligned}\widehat{\mathbb{E}}[X_{i,j,k}] &= \exp\{f_{\psi}^{\text{GBM}}(i,j,k;\widehat{\boldsymbol{\theta}})\} \\ &= \exp\{\epsilon \sum_{b=1}^{B} r_{\psi}(i,j,k;\widehat{\gamma}_b)\}.\end{aligned} \qquad \text{(M3 - GBM)}$$

3.4 Neural networks

As in Section 3.3 we start with the general setting with a response $\boldsymbol{Y}$ that we want to regress on $\boldsymbol{c} \in \boldsymbol{C}$, and the neural network (NN) model will define a general regression function parametrised using a specific link-function $u(\cdot)$ according to

$$f^{\text{NN}}(\boldsymbol{c};\boldsymbol{w}) = u(\mathbb{E}[Y \mid \boldsymbol{c};\boldsymbol{w}]),$$

defining the mean of, in our reserving context, an ODP model. The reason for using "$\boldsymbol{w}$" instead of the previous "$\boldsymbol{\gamma}$", is because $\boldsymbol{w}$ will correspond to weights. We will here focus on so-called *feed-forward* neural networks and base our exposition on [14, 13, 7]. A feed-forward neural network is most easily understood from an example as the one given in Figure 2. Based on this figure it is natural to give $f^{\text{NN}}(\boldsymbol{c};\boldsymbol{w})$ a recursive representation, which leads us to the following definition of $f^{\text{NN}}(\boldsymbol{c};\boldsymbol{w})$ for a general feed-forward neural network with λ layers:

$$\begin{aligned} f^{\text{NN}}(\boldsymbol{c};\boldsymbol{w}) &= a^{(\lambda)}, \\ a_j^{(l)} &= g^{(l)}(w_{j,0}^{(l-1)} + \sum_{i=1}^{p_{l-1}} w_{j,i}^{(l-1)} a_i^{(l-1)}), \quad 2 \le l \le \lambda - 1,\ j = 1,\dots,p_l, \\ a_j^{(1)} &= c_j, \quad j = 1,\dots,p, \end{aligned} \qquad (7)$$

where the ws are weights to be estimated, p_l corresponds to the number of "neurons" (or nodes) in layer l, and the $g^{(l)}$-functions are the (parameter-free) so-called "activation functions", that often have a sigmoid shape, such as the hyperbolic tangent function (tanh). Further, the first layer is called the "input" layer, the last layer is called the "output" layer, and all intermediate layers are called "hidden" layers. A feed-forward neural network with more than 2 hidden layers is what usually is referred to as being a "deep" (feed-forward) neural network. Further, from the above it is clear that this type of model, consisting of a large number of parameters and iterative composition of functions, will be very flexible making them so-called "universal approximators" (Cybenko's universality theorem). Moreover, due to that $f^{\text{NN}}(\boldsymbol{c};\boldsymbol{w})$ is expressed in terms of compositions of $g^{(l)}$-functions, it is possible to obtain explicit gradients using so-called "backpropagation", see e.g. [7, Alg. 18.1]. By using these gradients the parameter estimates can be updated iteratively. Usually not all input data

is used in each gradient update step, but rather the gradient update is based on a sample of input data ("batch") chosen at random. This is primarily used if input data is very large. Due to this the concept of "epochs" has been introduced, where an epoch corresponds to that all input data has been used for a parameter estimate update. That is, if all data is used in each step of the gradient updating procedure, an epoch is the same as a standard iteration. If batches are used, several gradient updates will be done within each epoch.

The flexibility of NN models is partly due to the large number of parameters, often thousands, which will make the models heavily overparametrized, with the risk of overfitting if you run too many epochs (iterations). The problem with overparametrization can be addressed by using regularization techniques, by adding e.g. Ridge or Lasso penalization to the likelihood, see e.g. [14, 7]. A technique related to regularization that will be used in the current paper is to use what is known as "dropout", which corresponds to that a fraction of all weights for each layer in the network are randomly set to zero in each epoch.

Further, recall that the starting point of the current paper is GLM models. One way to improve the performance of a feed-forward NN in this context is to use the approach taken in [11, 30] where one first fit a GLM model, and use these parameters as offsets (or non-trainable parameters) in an NN model. That is, the NN model will essentially be fitted to the residuals from the GLM model, which may be thought of as so-called regression or residual boosting the GLM model using a NN model, see e.g. [7, Ch. 16.7, Ch. 17.2]. Given that the GLM model provides a reasonable fit, the NN model will likely need less training. Still, this procedure will *not* provide initial weights $\boldsymbol{w}$.

Concerning other aspects of tuning of NN models, you can choose the number of layers, the number of neurons (or nodes) in a layer, activation functions (that often are different for different layers), and the number of epochs used for training. All of these decisions can be evaluated using the techniques discussed in Section 3.2 by splitting the data into training and validation sets. We will comment more on this in Section 4.

3.4.1 Reserving models

In analogy with the GBM estimated models from Section 3.3.1, using log-link functions, it follows that the NN estimated version of model 2 will produce

$$\begin{cases} \widehat{\mathbb{E}}[X_{i,j,k}|N_{i,j}] &= N_{i,j} \exp\{f_{\psi}^{\mathrm{NN}}(i,j,k;\widehat{\boldsymbol{w}})\} \\ \widehat{\mathbb{E}}[N_{i,j}] &= \exp\{f_{\nu}^{\mathrm{NN}}(i,j;\widehat{\boldsymbol{w}})\} \end{cases} \qquad \text{(M2 - NN)}$$

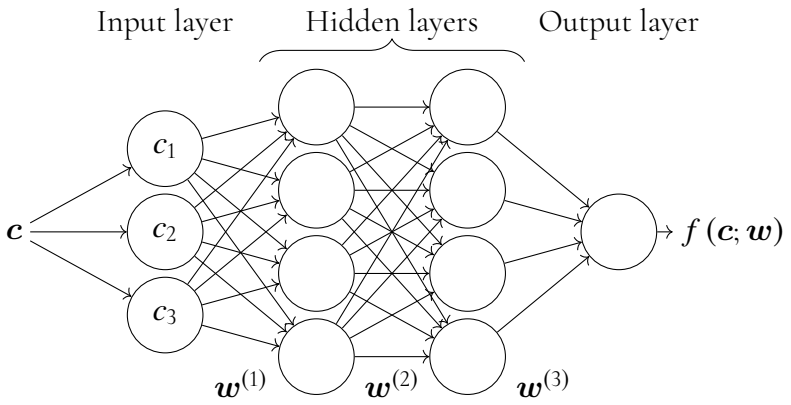

Figure 2: Example of feed-forward neural network taking $\boldsymbol{c} = (c_1, c_2, c_3)'$ as input covariates with $\lambda = 4$ layers, where 2 are hidden layers with $p_2 = p_3 = 4$ neurons, where $\boldsymbol{w}^{(l)}$, $l = 1, 2, 3$, corresponds to weights.

with $f_\bullet^{\text{NN}}(\cdot\,; \widehat{\boldsymbol{w}})$ from (7), and where $\widehat{\varphi}$ and $\widehat{\phi}$ correspond to estimates based on e.g. Pearson or deviance residuals. Again, as discussed in Section 3.3.1 this is sufficient information in order to be able to produce separate RBNS and IBNR reserves, as well as to produce bootstrapped reserves.

As before, the NN estimated version of Model 3 will produce

$$\widehat{\mathbb{E}}[X_{i,j,k}] = \exp\{f_\psi^{\text{NN}}(i, j, k; \widehat{\boldsymbol{w}})\}. \qquad \text{(M3 - NN)}$$

4. NUMERICAL ILLUSTRATION

In this section, we give an illustration of how we, in practice, can use the machine learning techniques described in this paper. For this, we consider six different portfolios of simulated data generated by the individual claims history simulation machine of [12]. These six portfolios will be referred to as Lines of Business (LoBs) 1-6 and are identical to those in [11]. For a detailed description of these LoBs and their characteristics, see Section 2.3 and the appendix in [11]. For `R`-code showing how to generate the individual-level data, see Listing 1 in the same paper. We will not describe the data and simulation machine in detail here. In short, the simulation machine uses neural networks calibrated on Swiss insurance data to simulate synthetic claims histories. The 6 LoBs are generated independently of each other, with the differences between them being that the first three use one stochastic generator and the last three another one. The difference between the LoBs within the same group, i.e. LoBs 1-3 and 4-6, are that their underlying features, which are the size, growth, and covariate structure of the portfolios, are different. The synthetic data

that we will look at here can be simulated following Listing 1 of [11] using seed 75, implemented using the `R`-package `keras`.

As mentioned in Section 3.2, to alleviate overfitting, we perform early stopping in training our GBMs and NNs, which we illustrate further on in this section. To do this, we need to split the data into a training and a validation set, and we do this following [11] by, for each LoB and AY, dividing the individual claims into two datasets of equal size according to Listing 2 in [11]. As is described in [11], the two datasets should be of equal size since parameters may be volume-dependent, see also the discussion in Section 3.2 above. In Listing 1 of the present paper, we have included code that aggregates the individual-level data from the simulation machine into data corresponding to the $\boldsymbol{X_{i,j,k}}$s and $\boldsymbol{N_{i,j}}$s.

Table 1 shows the number of individual claims and the total payments generated for each LoB, together with the proportion of these that are RBNS. From this table, we can note that extended reporting and payment delays are not a prominent feature of the data generated by this simulation machine since most claims are RBNS at the current time. Still, the simulated data contain claims that have longer reporting and payment delays, see e.g. Figure 3.

Table 1: Number of generated individual claims per LoB.

	LoB 1	LoB 2	LoB 3	LoB 4	LoB 5	LoB 6
Number of individual claims	250,040	250,197	99,969	249,683	249,298	99,701
Percent RBNS	99.4%	99.4%	99.2%	99.2%	99.2%	99.1%
Percent RBNS in last AY	93.6%	93.7%	93.3%	91.5%	91.5%	91.6%
Total payments	285,989	278,621	108,345	429,344	437,728	171,482
Total outstanding payments	39,689	37,037	16,878	71,630	72,548	31,117

Given that we simulate data, we know all payments, both current and future. Therefore we will be able to compare the reserves of the different models and methods to what the real outstanding amounts are. One metric we will use to assess the performances is the relative bias (error). The relative bias of a prediction $\widehat{\boldsymbol{R}}$ of the true reserve $\boldsymbol{R}$ is defined as $\frac{R-\widehat{R}}{R}$. In Table 2, we show the reserves of four benchmark models, the chain-ladder (CL) technique, the standard CRM using $\boldsymbol{X_{i,j,k}}$-data (Model 2) with a log-link and the parametrization $\log \psi_{i,j,k} = \alpha_k$, the CRM using the parametrization $\log \psi_{i,j,k} = \alpha_i + \beta_j + \gamma_k$ — from now on referred to as the generalized CRM (GCRM) — and the neural network of [11] (GRWNN). The standard CRM severely underestimates the true reserve, which is not necessarily surprising since the CRM assumes that payments are only dependent on the payment delay. Therefore, this underestimation could be due to there being some accident year or

reporting delay effect in the data that the standard CRM cannot capture. We see that the simple extension to the GCRM already alleviates this problem to a large extent and performs similarly to the CL. Although the neural network (GRWNN) yields the smallest biases across the board; however, it does not allow for the computation of individual RBNS and IBNR reserves. It is also not as interpretable as the conditional ODP models discussed in this paper, which can be connected to detailed individual-level assumptions. In Table 7 in the appendix, a version of Table 2 for IBNR and RBNS reserves are shown, and it is clear that the GCRM performs better than the standard one. It seems that we, in general, underestimate RBNS reserves while overestimating IBNR reserves.

Table 2: True reserve compared to CL, the CRM and [11] (GRWNN) reserves. Relative bias (error) of the predictions in the parentheses.

	LoB 1	LoB 2	LoB 3	LoB 4	LoB 5	LoB 6
True reserves	39,689	37,037	16,878	71,630	72,548	31,117
CL reserves	38,569	35,460	15,692	67,574	70,166	29,409
	(-2.82)	(-4.26)	(-7.02)	(-5.66)	(-3.28)	(-5.49)
CRM reserves	32,485	29,901	13,040	55,782	59,390	24,403
	(-18.15)	(-19.27)	(-22.74)	(-22.12)	(-18.14)	(-21.58)
GCRM reserves	38,293	35,117	15,448	66,961	69,397	29,104
	(-3.52)	(-5.18)	(-8.47)	(-6.52)	(-4.34)	(-6.47)
GRWNN	39,233	35,899	15,815	70,219	70,936	30,671
	(-1.15)	(-3.07)	(-6.30)	(-1.97)	(-2.22)	(-1.43)

It is difficult to say why we see these underestimated RBNS and overestimated IBNR reserves, except for the CRM. For the CRM, it is clear that only taking payment delay into account leaves a lot of room for error. If there is some accident year effect, for instance, then we would have no way of capturing it. The GCRM, on the other hand, can catch this potential accident year effect. Nonetheless, it still underestimates the reserve for all LoBs. This underestimation could be due to many different reasons. For instance, as [11] notes, there may be some interaction that the cross-classified model structure cannot capture. That is, there may be some interaction effect between, for instance, payment and reporting delay that affects how large payments are. This suspicion is strengthened by Figure 3, which shows the average development of the payments from an individual claim as a function of the payment delay and the first three periods of reporting delay for each LoB. If there is no interaction between reporting and payment delay, the curves in this figure should overlap, which they do not. The cross-classified model structure does not allow us to capture this difference in patterns. In addition to interaction effects, there could also be some

calendar year or inflation effect, which these cross-classified model structures likely will not capture.

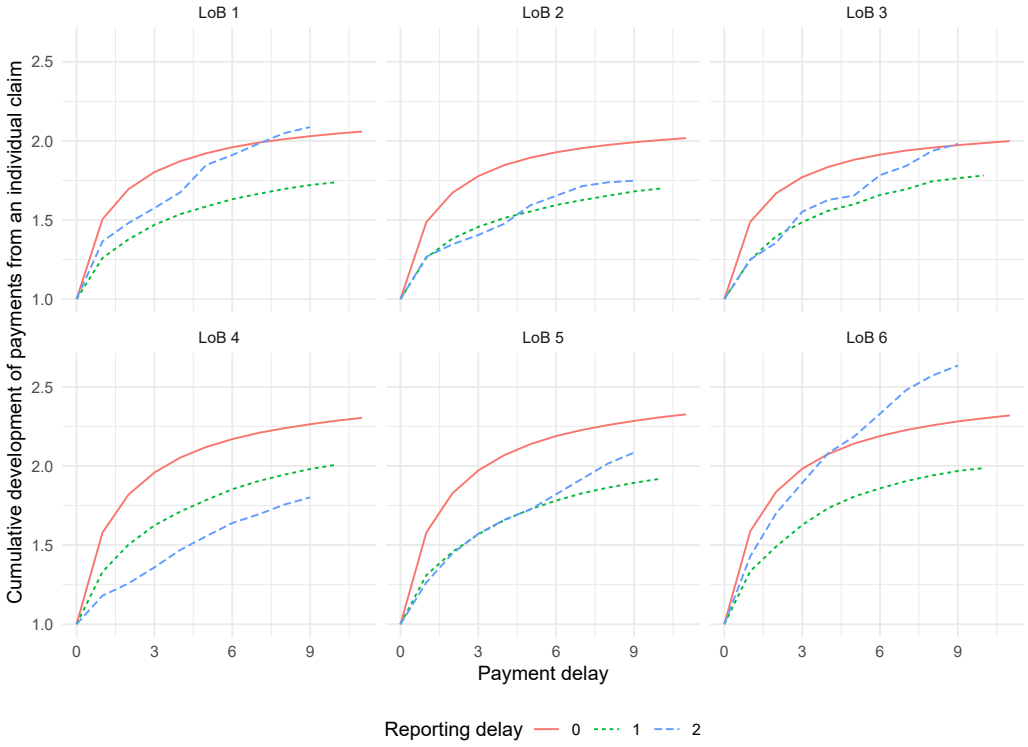

Figure 3: Average cumulative development of the payments from an individual claim as a function of payment delay, stratified on LoB and the first three periods of reporting delay, for all claims in all accident years.

In Figure 4, we show heatmaps of the relative biases of the predicted payments from the two versions of the CRM in each combination of accident year and development year for LoB 1. The CRM overestimates payments in the upper left triangle, while the opposite is true in the bottom right triangle. The GCRM has a much weaker, but similar pattern. A potential explanation is that the GCRM has picked up on some accident year effect, while not being able to capture some calendar year effect. To illustrate this further, Figure 5 shows how the average total size of the payments for individual claims evolves over the accident years. We see that there is a clear trend upwards for all of the LoBs. This trend upwards could be due to some type of inflation effect that the (generalized) CRM cannot capture.

We now move on to the machine learning methods, i.e. the GBM and NN. We

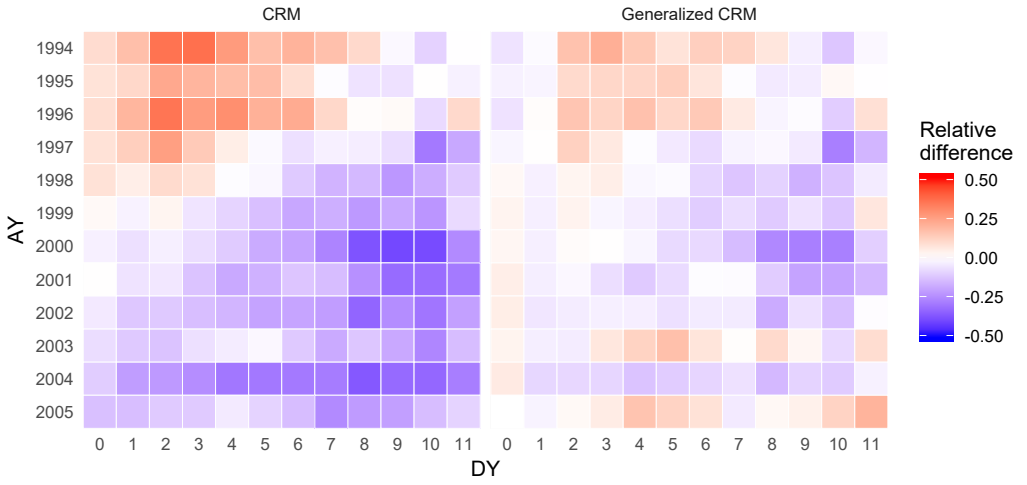

Figure 4: Heatmap for the relative biases (errors) of the prediction within a specific accident year and development year combination for LoB 1 using the CRM and GCRM.

will look at tunings of these algorithms that are, in some sense, standard. For the neural networks, we use the architecture of [11]. Their code, see Listing 4 in [11], can be used without modification for the model of the number of reported claims. However, for the payment part of the model, we have to adapt it to our situation with more granular data. This adaptation is straightforward to implement given the already available code from [11], which uses the `R`-package `keras`; see Listing 3 in the appendix.

For the number of epochs in training the neural networks, we set an upper limit at 10,000. If we consider LoB 1 and the number of reported claims part of Model M2 - NN, running 10,000 epochs takes around 47 seconds while it takes 174 for the payment part of the model[2] Therefore, in the worst-case scenario, we run 20,000 epochs taking around 220 seconds, which is still a feasible amount of computation time. However, if we want to compute an MSEP using bootstrapping, then we would need to repeat this fitting procedure several times. Assume we are satisfied with 100 simulations in the bootstrap, a rather small number for estimating the MSEP, which may be skewed and heavy-tailed, the bootstrap would take ca 6 hours to run. If we increase the number of bootstrap simulations to a thousand, the run-time would be about two and a half days for a single line of business. Now, given a faster computer,

[2]These and all other computations in this section are performed on a stationary PC with an Intel Core i7-7700 processor @ 3.60Ghz and 8GB RAM.

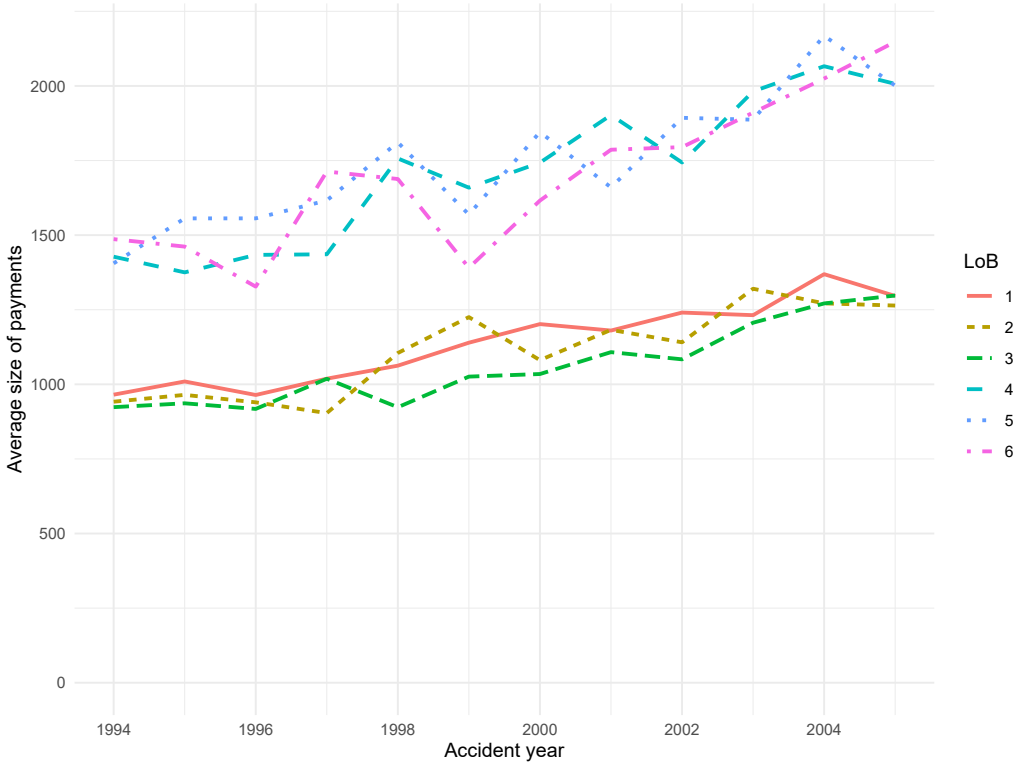

Figure 5: Average size of payments within the different combinations of reporting and payment delay for each accident year.

this number would naturally be smaller. For this illustration, we will stick to a maximum of 10,000 epochs, and simply state that the performance of the neural networks might be a lower bound on how well they could perform given more computation time.

For the GBMs, we start with the standard tuning in the `gbm`-function of the `gbm`-package in `R` and perform a rudimentary tuning of these by slightly changing one tuning parameter at a time and investigating how this affects the validation loss and actual out-of-sample performance (lower right triangle). This latter out-of-sample check cannot be done in practice. Nonetheless, since we can simulate data, we can investigate how tuning these parameters seem to perform over a number of simulated data sets generated by the same underlying dynamics.

The standard set of parameters in the `gbm`-function is a tree depth of 1 with a shrinkage factor of 0.1, a bagging fraction of 0.5, and a minimum of 10 observations per node. We will call this the *standard tuning* and now discuss how our tuning re-

lates to it and how we arrive at it. First of all, we set the maximum number of trees to 10,000 and then decide how many are needed based on the validation loss when fitting the models to the training data. For the GBMs, for both the payment part of the model and the number of reported claims part, our tunings seem to suggest that we get similar results with a smaller shrinkage of 0.01 as with 0.1, although then we need about ten times more trees. This can be seen in the upper left graph of Figure 6, where we illustrate the training and validation loss as a function of the number of trees in the GBM for the payment part of Model M2 - GBM applied to LoB 5 with our tuning using these two shrinkage factors. We see that the results are almost identical, with the larger shrinkage factor allowing us to reach the minimum validation loss faster. Whichever shrinkage factor we choose, the algorithm is reasonably quick. Fitting 10,000 trees takes ca half a second, which is feasible even within a bootstrap — increasing the number of trees to 100,000 increases the computation time tenfold, which should not be a problem in practice. Although, seeing as the results are almost identical, to keep computation times within reason, we settle for a shrinkage factor of 0.1. In practice, if the validation loss looks volatile, one usually decreases the shrinkage factor, making the loss more stable (smooth). It should be noted that the computation time here is orders of magnitude smaller than for the neural networks.

In the upper right graph of Figure 6, we illustrate the effect of varying the bagging fraction. In our case, bagging does not seem to help us out, only resulting in noisy deviance paths without, seemingly, decreasing the validation loss or yielding better out-of-sample performance. We, therefore, set the bagging fraction to 1, i.e. no bagging at all. Further, our data is quite small, and we seem to be helped by decreasing the minimum observation per node to 1. This can be a problem since it is possible that we base predictions on parameters estimated using only one observation, and therefore possibly introducing a large amount of variance in the predictions. However, the aggregated data that we use is small, especially taking into account that reporting and payment delays are so fast, and we thus have many zeroes – in particular for the claim counts. It can, therefore, be the case that we miss effects in the data by forcing ten observations per node, i.e. we introduce bias. How to decide on this trade-off between bias and variance is up to the practitioner, possibly by minimizing an MSEP. Based on the bottom left graph of Figure 6, there does not seem to be much of a difference anyway, so we decide to allow for one observation per node.

Finally, the depths of the trees are slightly more complicated. Looking at the validation data in the bottom right graph of Figure 6, we would be led to believe that depth 2 is better than 1. This seems to be the case for the number of reported claims. However, for the payment part of the model, this then leads to awful out-of-sample (lower triangle) performance, which would seem to indicate that the GBM overfits interactions in the observed data that are not representative of future payment pat-

terns. This is not obvious from the training and validation loss, although the distinct decrease in the training loss, which is not seen in the validation loss, may be seen as an indication of overfitting. Further, the reason for illustrating the tuning for LoB 5 is that not all of the other LoBs has the pattern where a tree depth of 2 yields a lower validation loss than a tree depth of 1. For some LoBs, the validation loss becomes worse when using a tree depth of 2.

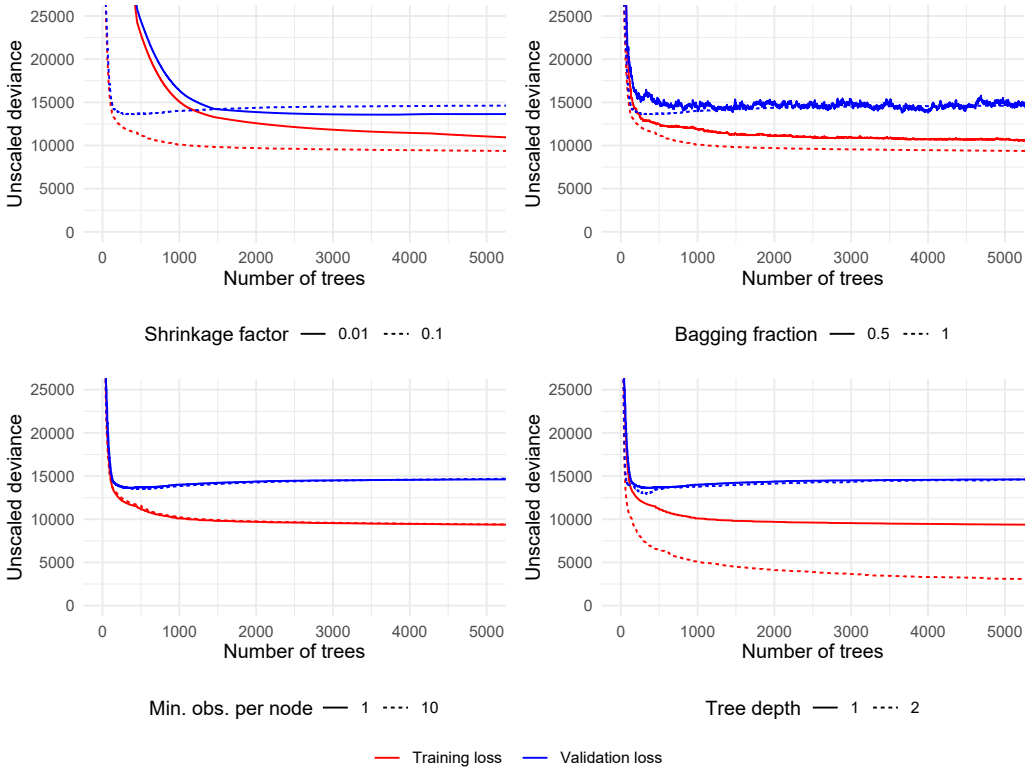

Figure 6: Training and validation loss for our tuning of the GBM for the payment part of Model 2 when varying the shrinkage factor (upper left),the bagging fraction (upper right), the minimum number of observations per node (lower left), and the tree depth (lower right).

Finally, it is straightforward to include transformations of the data when fitting the GBMs. As we have seen, there is a potential inflation (or calendar year) effect in the data, see Figure 4. Therefore, in addition to including the accident year, the reporting delay, and the payment delay as covariates, we also include the calendar time of reporting, i.e. the sum of the accident year and the reporting delay. Technically the GBM should be able to capture this transformed feature by itself; however, including it does seem to help performance quite a bit. Nonetheless, the GBM without this

inflation feature still performs well.

Another part of the calibration of machine learning methods is early stopping, discussed in Section 3.2, i.e. the choice of the number of trees in the GBMs and the number of epochs in training the NNs. [11] describe how to choose the number of epochs when training a neural network, which we follow closely. For this reason, we refrain from repeating what they have already described, and simply add that in addition to what they do, we choose the number of epochs by finding the minimum validation loss using a central simple moving average with window size 100. We use this moving average since the validation loss becomes volatile when using the dropout in the NNs. Without showing a graph, we note that the convergences are very slow for the incurred claims part of the NN models. The number of epochs at which the minimum (moving average) validation losses are attained can be seen in Table 3, in which we report (for Model 2) the number of epochs used for the NNs and the number of trees for the GBMs. Almost all LoBs need close to the maximum of 10,000 epochs and allowing for more epochs may yield better results than those we will acquire. For the payment part of the model, the minimum losses are reached much earlier for most LoBs.

Now let us look at the analogous choice of the number of trees for the GBMs. In Figure 7, we illustrate these choices for Model 2 by showing the training and validation loss (unscaled deviance) as a function of the number of trees for our and the standard tuning. The dotted black lines indicate the minimum of the validation loss as well as its position for our tuning. It is clear that our tuning and the standard tuning yield similar results, with our tuning generally being more stable and achieving slightly smaller losses. It is also clear that the validation losses for each LoB reach a point where they do not decrease any more and, most often, start to increase. This point is where we begin to overfit by adding more trees. Thus, this will be the number of trees that will be used in the GBM model when making the final prediction of the outstanding payments based on all data (training + validation). As was mentioned above, Table 3 shows these numbers of trees.

Table 3: Number of epochs used in training the NNs and the number of trees used in the GBMs for Model 2.

Number of epochs/trees	LoB 1	LoB 2	LoB 3	LoB 4	LoB 5	LoB 6
Payment part NN	8,825	405	225	9,895	331	227
Incurred claims part NN	9,950	9,704	9,357	9,946	9,751	1371
Payment part GBM	279	200	251	171	357	206
Incurred claims part GBM	635	383	912	310	360	467

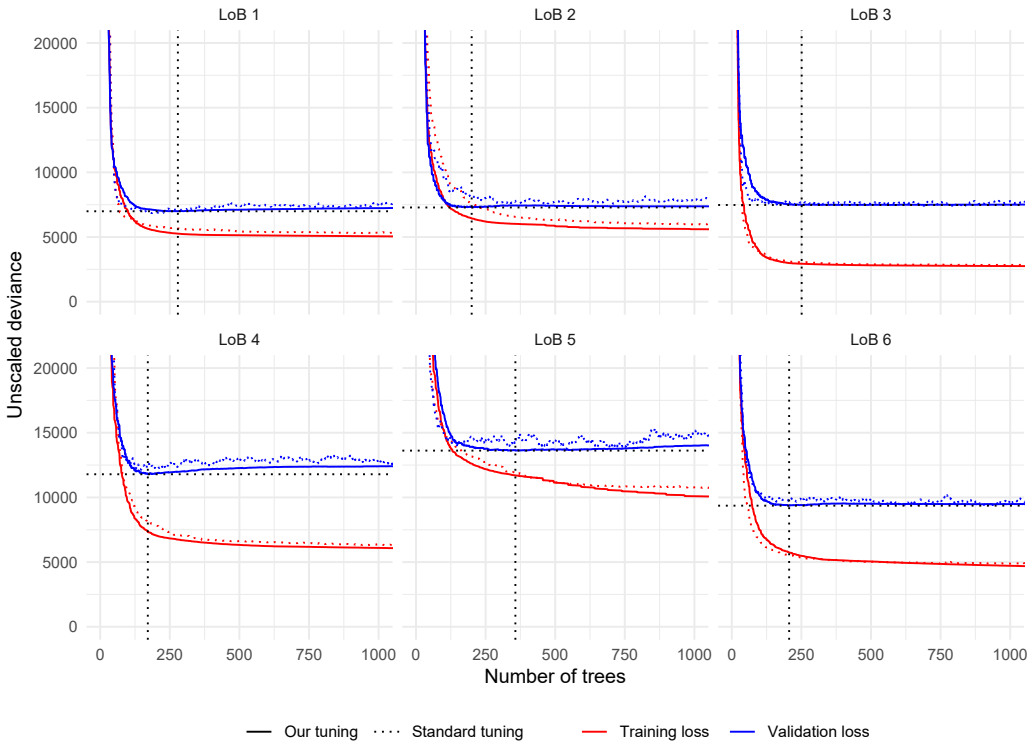

Figure 7: Out-of-sample validation analysis for over-fitting (determining the number of trees) in the GBM for Model 2. The dotted vertical and horizontal lines show the minimum of the validation loss and the number of trees at which it is obtained for our tuning.

While GBMs are still non-standard in the insurance industry, they are surprisingly accessible given, for instance, familiarity with GLMs and `R`. Listing 2 in the appendix shows how to use the GBMs in practice when fitting Model 2. This short snippet of code is a lot more accessible than the corresponding code for the neural networks using the `R`-package `keras`. Note that the `keras` code in Listing 3 is only for the aggregated payments.

Table 4 shows the reserves produced by these machine learning methods, as well as the benchmark models and the true reserves. Both machine learning methods seem to be performing well, especially the GBM for Model 2. In Figure 8, we show a heatmap of the relative biases of the predictions generated by the GBM for Model 2. There is not any visible pattern remaining in the data, which is in line with its excellent performance. The only cell with a notable error is the bottom-right cell; however, due to the quick reporting of claims, not many payments are made in that cell, and

the importance of predicting it with a small margin of error is not crucial.

Table 4: True reserve compared to benchmark models and the GBMs and NNs. Relative biases of the reserve predictions in the parentheses.

	LoB 1	LoB 2	LoB 3	LoB 4	LoB 5	LoB 6
True reserves	39,689	37,037	16,878	71,630	72,548	31,117
CL reserves	38,569	35,460	15,692	67,574	70,166	29,409
	(-2.82)	(-4.26)	(-7.02)	(-5.66)	(-3.28)	(-5.49)
CRM reserves	32,485	29,901	13,040	55,782	59,390	24,403
	(-18.15)	(-19.27)	(-22.74)	(-22.12)	(-18.14)	(-21.58)
GCRM reserves	38,293	35,117	15,448	66,961	69,397	29,104
	(-3.52)	(-5.18)	(-8.47)	(-6.52)	(-4.34)	(-6.47)
GRWNN	39,233	35,899	15,815	70,219	70,936	30,671
	(-1.15)	(-3.07)	(-6.30)	(-1.97)	(-2.22)	(-1.43)
GBM (Model 2)	39,697	37,253	16,508	72,679	71,828	31,941
	(0.02)	(0.58)	(-2.19)	(1.46)	(-0.99)	(2.65)
GBM (Model 2) without inflation	38,324	37,053	16,327	73,386	70,486	32,100
	(-3.44)	(0.04)	(-3.26)	(2.45)	(-2.84)	(3.16)
GBM (Model 3)	40,114	35,729	15,761	69,448	72,418	30,061
	(1.07)	(-3.53)	(-6.62)	(-3.05)	(-0.18)	(-3.39)
NN (Model 2)	41,587	37,587	15,680	71,155	71,309	28,984
	(4.78)	(1.48)	(-7.10)	(-0.66)	(-1.71)	(-6.86)
NN (Model 3)	39,757	38,719	16,245	70,916	74,600	28,943
	(0.17)	(4.54)	(-3.75)	(-1.00)	(2.83)	(-6.99)

The conclusions based on these reserves are, of course, only based on one dataset. Therefore, we simulate new datasets for each seed 1-100, and for each of these, repeat the reserve computations. We do this for the CL, CRM, GCRM, and the best performing GBM (Model M2 - GBM) and NN (Model M2 - NN). Figure 9 shows boxplots of the relative biases of these reserves for the simulated data using seeds 1-100. All of these data sets have been generated using the same underlying model, and are hence i.i.d., and we see that LoB 3 and 6 are those with most variation, probably due to the lower portfolio sizes generating fewer claims. Further, recall Figure 3, which tells us that, e.g., the realised simulated reporting delays may fluctuate quite a bit between simulations. In the appendix, Figures 10 and 11 show the corresponding graphs for the RBNS and IBNR reserve predictions. The GBM model seems to be the one with the least amount of bias; however, the NN has a predictor with smaller variance. Which of these should be deemed most important is to be decided by the practitioner using the model. However, we usually pick models based on the mean squared error of prediction, as this naturally balances the importance of bias and variance. Table 5

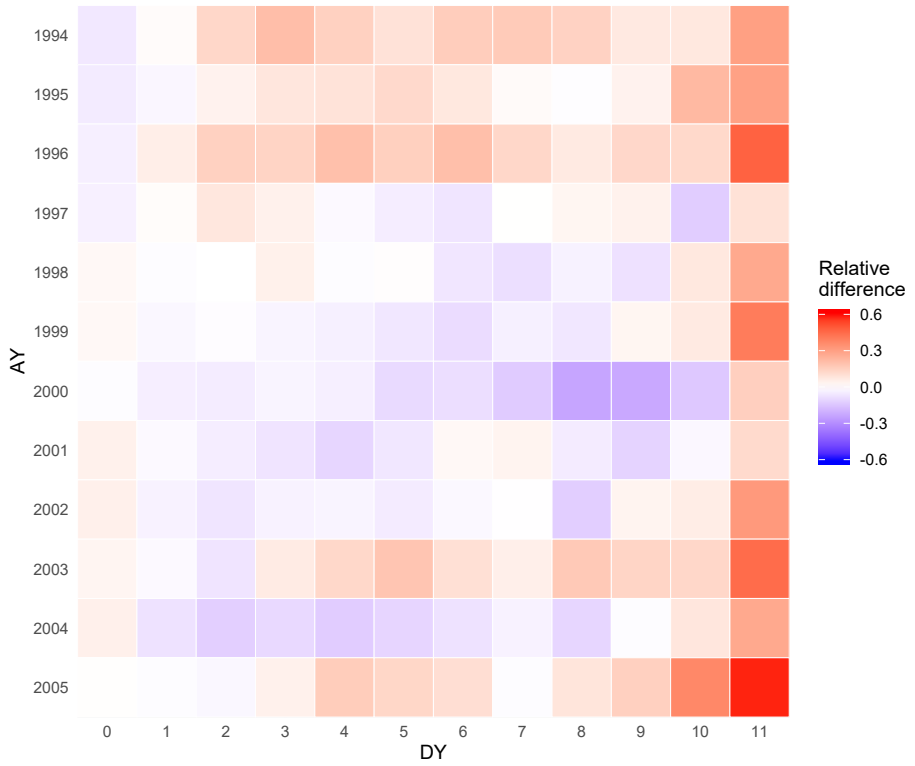

Figure 8: Heatmap for the relative biases (errors) of the prediction within a specific accident year and development year combination for LoB 1 using the GBM.

shows the average bias and the (true) root mean squared error of prediction across the simulations. The NN yields the smallest root mean squared error of prediction (RMSEP) except for LoB 3 and 6 where CL has slightly smaller RMSEP. Nonetheless, CL does not allow for separate RBNS and IBNR predictions, speaking in favor of the NN. It is, of course, also important to note that we have not allowed ourselves to run more than 10,000 epochs and that we have not performed any tuning of the neural networks, which could yield even better predictions.

4.1 Conditional mean squared error of prediction

We end this section by estimating the conditional MSEP for a subset of the models in this paper. First, for some predictor $\widehat{\boldsymbol{R}}$ of the outstanding payments $\boldsymbol{R}$, the

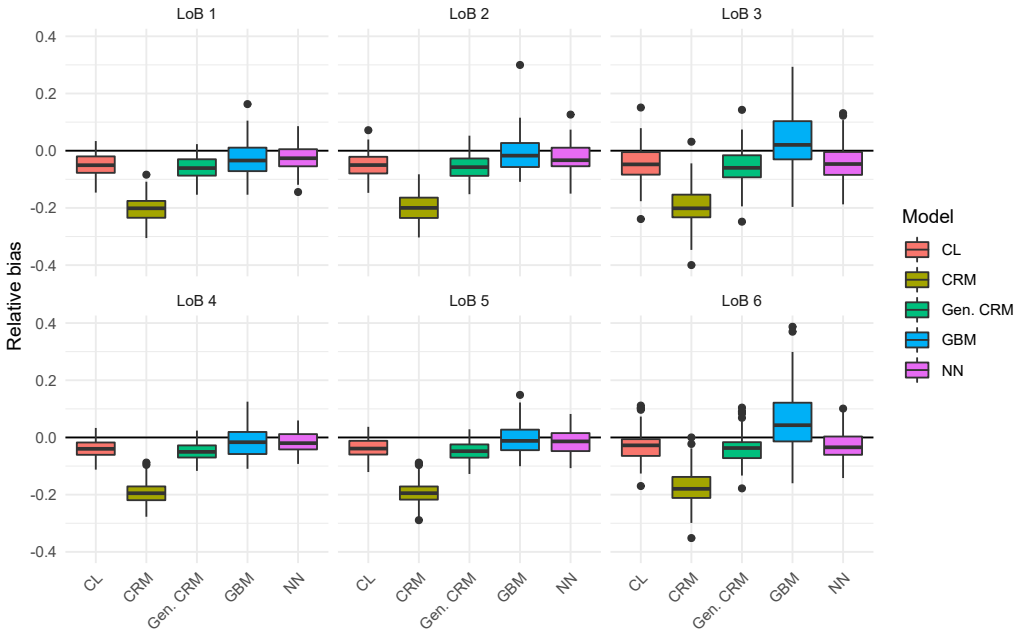

Figure 9: Boxplots of relative biases from the 100 simulations.

Table 5: Biases and RMSEP from the simulations using seeds 1-100 for the CL, CRM, GCRM, and the GBM and NN for Model 2.

bias	LoB 1	LoB 2	LoB 3	LoB 4	LoB 5	LoB 6
CL	-2,035	-1,945	-886	-2,927	-2,724	-1,051
CRM	-8,196	-8,017	-3,606	-14,310	-14.133	-5,889
GCRM	-2,452	-2,328	-1,073	-3.672	-3.507	-1,384
GBM	-1,162	-553	664	-937	-497	1,878
NN	-1,077	-1,109	-840	-1,212	-1,352	-1,064
RMSEP	LoB 1	LoB 2	LoB 3	LoB 4	LoB 5	LoB 6
CL	2,649	2,587	1,507	3,716	3,775	2,045
CRM	8,453	8,284	3,885	14,652	14,574	6,311
GCRM	2,979	2,878	1,630	4,342	4,375	2,231
GBM	2,767	2,574	1,854	3,897	4,019	3,832
NN	2,252	2,225	1,517	2,769	3,391	2,107

conditional MSEP is defined as

$$\begin{aligned}\mathrm{MSEP}(R,\widehat{R}|\mathcal{N}_0) &:= \mathbb{E}\left[\left(R-\widehat{R}\right)^2\middle|\mathcal{N}_0\right] \\ &= \mathrm{Var}\,(R|\mathcal{N}_0) + (R-\mathbb{E}[\widehat{R}|\mathcal{N}_0])^2,\end{aligned}$$

where the first term is usually referred to as the process variance and the second term as the estimation error. Given the ODP assumptions, it is straightforward to compute the process variance analytically and to utilize a bootstrap to estimate the estimation error. We detail how the process variances can be computed analytically and give a cursory description of the bootstrap in Appendix C.2. We refer the reader to [21] for further information on how to bootstrap CRM-type models.

The estimators of the conditional MSEPs for the CL, the GCRM, and the NN and GBM for Model 2, are given in Table 6. It should be noted that these MSEPs are estimated by analytical calculations and parametric bootstrapping that assume the fitted models to be the true ones – recall the discussion in Section 3.3.1 and Section 3.4.1, when introducing Model M2 - GBM and Model M2 - NN. Remember that Table 5 shows the true MSEPs based on simulating new datasets form the simulation machine, and are therefore taking model error into account. However, it is essential to note that these are unconditional MSEPs and are therefore not directly comparable to the estimated conditional ones in Table 6. Nonetheless, the conditional and unconditional MSEPs are likely not going to differ considerably, and comparing the two tables indicates that the machine learning methods give much more accurate representations of their true MSEPs than the CL, CRM, and GCRM do.

Table 6: Root mean squared error of prediction based on analytical calculation of the process variance and bootstrapping of the estimation error for the CL, the GCRM, and the GBM and NN for Model 2. n_{boot} is the number of bootstrap samples used in estimating the estimation error. Note that for the NNs we only use 100 due to the computation time being much longer than for the other models. The numbers in the parentheses are run-times (in seconds) of the bootstrap simulations.

	LoB 1	LoB 2	LoB 3	LoB 4	LoB 5	LoB 6	n_{boot}
CL	1,092 (3.4)	1,313 (3.4)	460 (3.3)	2,145 (3.3)	1,900 (3.3)	995 (3.2)	1,000
GCRM	1,044 (16)	1,147 (15)	670 (15)	1,691 (15)	2,430 (15)	1,254 (15)	1,000
GBM	1,854 (33)	2,691 (24)	1,210 (37)	5,802 (20)	4,715 (33)	2,890 (25)	1,000
NN	1,828 (9,060)	2,676 (4,800)	733 (4,600)	3,053 (9,700)	3,895 (4,900)	1,425 (910)	100

5. CONCLUDING REMARKS

In the present paper we have introduced regression based reserving models that can produce separate RBNS and IBNR reserves based on aggregated data, and described how their regression functions can be modelled using complex algorithmic

machine learning techniques, including calibration and model selection. Our focus has been on GBMs and feed-forward NNs, trying to use "standard" tuning to as a wide extent as possible. Still, we have described how more detailed tuning can be conducted, which may be needed in in real world applications. In the numerical illustration of Section 4, focus has been on GBM models, since we are not aware of work in this direction based on aggregated claims data. Concerning the NN models, we have used the architecture (number of hidden layers, number of neurons per layer, etc.) from [11], but the techniques discussed for choosing different epochs applies to how to choose different architectures as well. For more on this, see e.g. [10]. One obstacle in the reserving context is that model estimation and calibration is carried out based on partially observed claims data, making it a risk to overfit to historical claims development patterns. This problem is not restricted to machine learning techniques, but is a general problem, although the risk may be higher when using very flexible complex models in these situations. In the current paper we have used the training and validation setup from [11], see also Section 3.2 above together with Remark 5. An alternative could be to use other splits of data, such as removing the last diagonal from the training data.

The overall conclusion from the paper based on Section 4 is that by using off-the-shelf software and standard tuning, machine learning techniques can be used to improve the predictive performance also in aggregated claims reserving modelling, still producing interpretable output that can be communicated to non-experts, and whose regression functions (e.g. the $\psi_{i,j,k}$s) may be altered based on expert opinions, see the discussion in Section 3.1.

A. NUMERICAL ILLUSTRATION APPENDIX

Table 7: RBNS and IBNR reserves. Relative biases of the reserve predictions in the parentheses.

IBNR reserves	LoB 1	LoB 2	LoB 3	LoB 4	LoB 5	LoB 6
True	1,596	1,537	603	3,593	2,739	1,048
CRM	1,867 (16.99)	1,821 (18.42)	921 (52.60)	3,823 (6.39)	4,001 (46.09)	1,817 (73.48)
GCRM	1,846 (15.68)	1,800 (17.09)	1,012 (67.64)	3,569 (-0.67)	3,479 (27.05)	1,643 (56.83)
GBM (Model 2)	1,643 (2.96)	1,652 (7.42)	957 (58.51)	3,197 (-11.02)	2,894 (5.67)	1,447 (38.11)
GBM (Model 3)	2,501 (56.75)	2,114 (37.48)	1,133 (87.76)	4,504 (25.35)	4,667 (70.41)	2,044 (95.08)
NN (Model 2)	1,780 (11.55)	1,773 (15.29)	955 (58.23)	3,293 (-8.34)	3,292 (20.20)	1,535 (46.55)
NN (Model 3)	1,889 (18.37)	1,624 (5.64)	930 (54.15)	3,407 (-5.17)	3,479 (27.05)	1,621 (54.74)
RBNS reserves	LoB 1	LoB 2	LoB 3	LoB 4	LoB 5	LoB 6
True	38,093	35,500	16,275	68,037	69,810	30,069
CRM	30,619 (-19.62)	28,080 (-20.90)	12,119 (-25.53)	51,959 (-23.63)	55,390 (-20.66)	22,586 (-24.89)
GCRM	36,447 (-4.32)	33,317 (-6.15)	14,436 (-11.29)	63,392 (-6.83)	65,918 (-5.57)	27,461 (-8.67)
GBM (Model 2)	38,054 (-0.10)	35,601 (0.28)	15,551 (-4.44)	69,482 (2.12)	68,934 (-1.25)	30,495 (1.41)
GBM (Model 3)	37,613 (-1.26)	33,615 (-5.31)	14,628 (-10.11)	64,944 (-4.55)	67,751 (-2.95)	28,017 (-6.83)
NN (Model 2)	39,807 (4.50)	35,814 (0.88)	14,725 (-9.52)	67,861 (-0.26)	68,017 (-2.57)	27,448 (-8.72)
NN (Model 3)	37,868 (-0.59)	37,095 (4.49)	15,315 (-5.90)	67,509 (-0.78)	71,121 (1.88)	27,322 (-9.14)

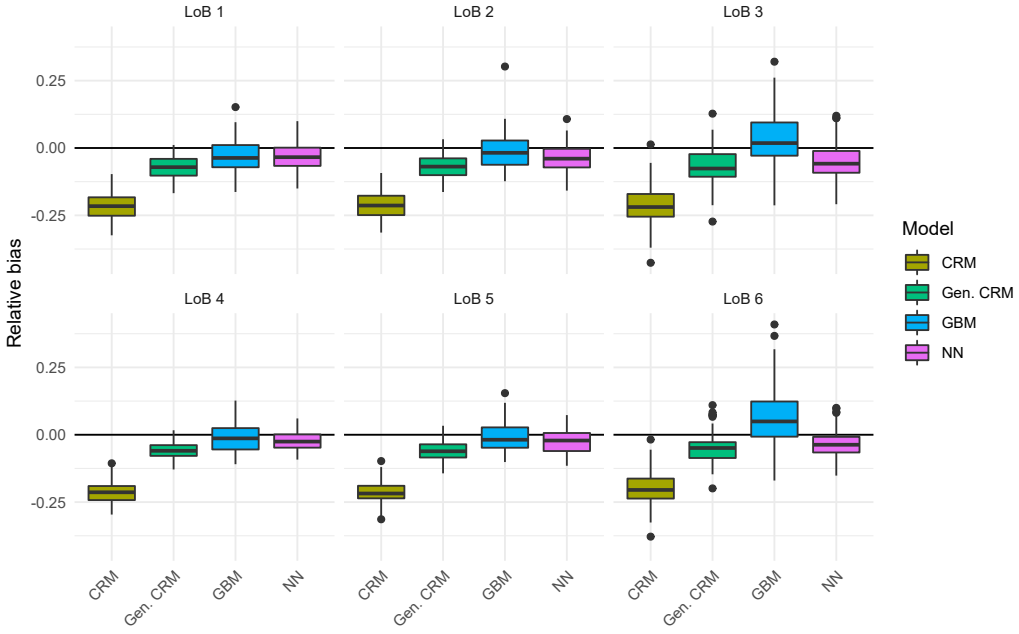

Figure 10: Boxplots of RBNS relative biases from the 100 simulations.

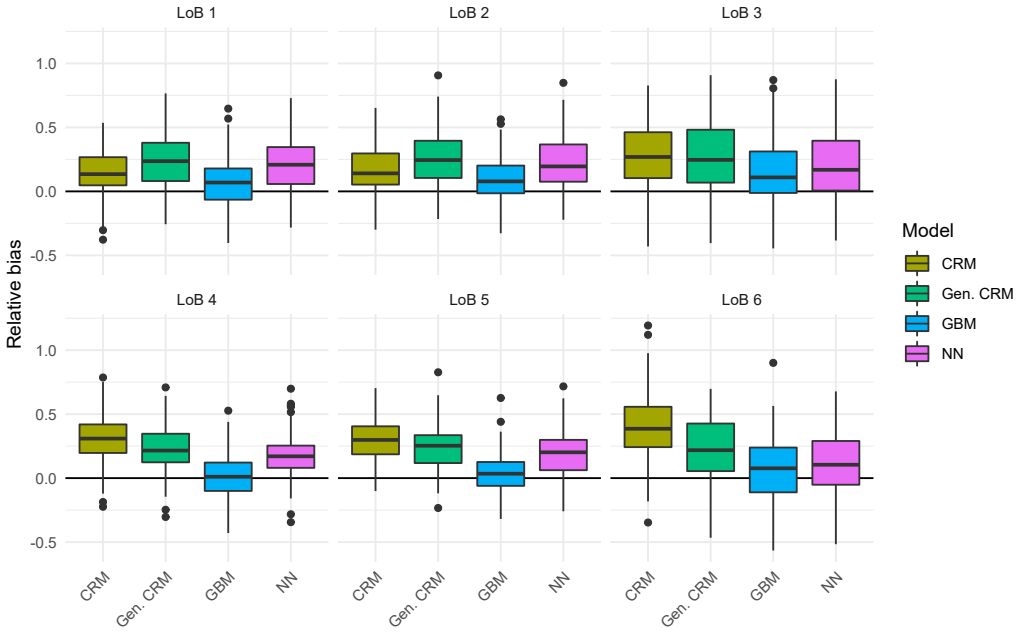

Figure 11: Boxplots of IBNR relative biases from the 100 simulations.

B. R-CODE

Listing 1: Aggregation of micro data. The data.frame `output` is generated by Listing 1 in GRW.

```
library("dplyr")
library("reshape2")
library("tidyr")

# Aggregate the micro data
df_aggregated <- output %>%
  dplyr::mutate(N = 1) %>%
  dplyr::select(LoB, AY, RepDel, N, dplyr::starts_with('Pay')) %>%
  dplyr::group_by(LoB, AY, RepDel) %>%
  dplyr::summarise_all(sum) %>%
  reshape2::melt(id = c("LoB", "AY", "RepDel", "N"),
                 variable.name = "PayDel",
                 value.name = "paid") %>%
  dplyr::mutate(PayDel = as.numeric(substr(PayDel, start = 4, stop = 5)),
                PayDel = PayDel - RepDel) %>%
  dplyr::filter(PayDel >= 0)

# Add cells without any reported claims
dat_X <- df_aggregated %>%
  tidyr::complete(LoB = 1:6,
                  AY = 1994:2005,
                  RepDel = 0:11,
                  PayDel = 0:11) %>%
  dplyr::group_by(LoB, AY, RepDel) %>%
  tidyr::fill(N) %>%
  dplyr::mutate(paid = ifelse(is.na(paid), 0, paid),
                N = ifelse(is.na(N), 0, N)) %>%
  dplyr::filter(PayDel + RepDel <= 11)

# Create dataset for only the number of reported claims
dat_N <- dat_X %>%
  subset(select = c(LoB, AY, RepDel, N)) %>%
  dplyr::distinct()
```

Listing 2: Fitting the GBMs for Model 2 to LoB 1.

```
library("gbm")

# Fit GBM to the number of reported claims for LoB 1
m_N_GBM <- gbm(
  formula = N ~ AY + RepDel,
  data = dat_N %>% dplyr::filter(LoB == 1, AY + RepDel + PayDel <= 2005),
  distribution = "poisson",
  interaction.depth = 2,
  shrinkage = 0.1,
  bag.fraction = 1,
  n.trees = 10000,
  n.minobsinnode = 1
)

# Fit GBM to the claims payments for LoB 1
m_X_GBM <- gbm(
  formula = paid ~ offset(log(N)) + AY + RepDel + PayDel + I(AY + RepDel),
  data = dat_X %>% dplyr::filter(LoB == 1, AY + RepDel + Paydel <= 2005),
  distribution = "poisson",
  interaction.depth = 1,
  shrinkage = 0.1,
  bag.fraction = 1,
  n.trees = 10000,
  n.minobsinnode = 1
)
```

Listing 3: Neural network architecture of GRW adapted to the payment part of Model 2.

```
# Declare features
AccYear <- layer_input(shape = c(1), dtype = "int32", name = "AccYear")
RepDel <- layer_input(shape = c(1), dtype = "int32", name = "RepDel")
PayDel <- layer_input(shape = c(1), dtype = "int32", name = "PayDel")
LogN <- layer_input(shape = c(1), dtype = "float32", name = "LogN")

# Define the embedding layers
AY_embed <- AccYear %>%
  layer_embedding(
    input_dim = 12, output_dim = 1, trainable = FALSE, input_length = 1,
    weights = list(array(alpha_ODP_train, dim = c(12, 1))), name = "AY_embed"
  ) %>%
  layer_flatten(name = "AY_flat")

RepDel_embed <- RepDel %>%
  layer_embedding(
    input_dim = 12, output_dim = 1, trainable = FALSE, input_length = 1,
    weights = list(array(beta_ODP_train, dim = c(12, 1))), name = "RepDel_embed"
  ) %>%
  layer_flatten(name = "RepDel_flat")

PayDel_embed <- PayDel %>%
  layer_embedding(
    input_dim = 12, output_dim = 1, trainable = FALSE, input_length = 1,
    weights = list(array(gamma_ODP_train, dim = c(12, 1))), name = "PayDel_embed"
  ) %>%
  layer_flatten(name = "PayDel_flat")

# Concatenate the embedding layers and add them to the CC model part
concate0 <- list(AY_embed, RepDel_embed, PayDel_embed) %>%
  layer_concatenate(name = "concate0")
CC0 <- list(AY_embed, RepDel_embed, PayDel_embed) %>% layer_add(name = "CC0")

# Define the 3 hidden layers of the NN model part
NN0 <- concate0 %>%
  layer_dense(units = 20, activation = "tanh", name = "hidden1") %>%
  layer_dropout(0.1, name = "dropout1") %>%
  layer_dense(units = 15, activation = "tanh", name = "hidden2") %>%
  layer_dropout(0.1, name = "dropout2") %>%
  layer_dense(units = 10, activation = "tanh", name = "hidden3") %>%
  layer_dropout(0.1, name = "dropout3")
```

```
# Define the bCCNN model using the skip connection for the CC part
Response <- list(CC0, NN0, LogN) %>% layer_concatenate(name = "concate1") %>%
  layer_dense(units = 1, activation = k_exp, name = "Response",
              weights = list(array(c(1, rep(0, 10), 1), dim = c(10 + 1 + 1, 1)),
                             array(intercept_ODP_train, dim = c(1))))

# Define and compile the model
model <- keras_model(inputs = c(AccYear, RepDel, PayDel, LogN),
                     outputs = c(Response))
model %>% keras::compile(optimizer = optimizer_rmsprop(), loss = "poisson")

fit <- model %>% keras::fit(x = dat_train_lst, y = y_train, epochs = 1000,
                            batch_size = batch_size,
                            validation_data = validation_data)
```

C. TECHNICAL DETAILS

C.1 Details on fitting a regression tree

Let us consider depth l of a tree that is being estimated, which implies that there are $j = 1, \ldots, 2^l$ decision sets $\mathcal{A}_{l,j} = \mathcal{A}(\pi_{l,j}, \kappa_{l,j})$ to be determined together with 2^{l+1} δ-values. That is, for each $\mathcal{A}_{l,j}$ there are two δ-values, $\delta_{l,j}^{(1)}$ which is assigned if the condition is fulfilled, and $\delta_{l,j}^{(0)}$ if the decision is not fulfilled. Let

$$\mathcal{I}_{l,j} = \{i : \text{ observation } i \text{ should be evaluated using } \mathcal{A}_{l,j}, i = 1, \ldots, m\},$$

where m corresponds to the maximal number of observations. This allows us to express the optimization problem for depth level l of the tree according to

$$\min_{\boldsymbol{\pi}_l, \boldsymbol{\kappa}_l} \min_{\boldsymbol{\delta}_l^{(1)}, \boldsymbol{\delta}_l^{(0)}} \sum_{j=1}^{2^l} \sum_{i \in \mathcal{I}_{l,j}} \left(\mathbb{1}_{\{\boldsymbol{c}_i \in \mathcal{A}_{l,j}\}} L(y_i, \delta_{l,j}^{(1)}) + \mathbb{1}_{\{\boldsymbol{c}_i \notin \mathcal{A}_{l,j}\}} L(y_i, \delta_{l,j}^{(0)}) \right).$$

This optimization step is repeated until the pre-defined tree depth k is reached.

The final $\mathcal{A}_j$ regions are obtained by taking the intersection of all $\mathcal{A}_{l,j}$ sets following the branches of the tree leading to this specific leaf. See also Figure 1.

C.2 Details on computing the conditional MSEP

In this section, we detail the estimation of the conditional MSEP by describing how one can, given the ODP assumptions, analytically compute the process variance and bootstrap to get an estimator of the estimation error.

Let us first detail how we can compute the process variances analytically, focusing on Model 2. By the ODP assumption, it holds that the process variances of the outstanding RBNS payments, i.e. those $X_{i,j,k}$s with indices $i + j \leq m$, are given by

$$\mathrm{Var}(X_{i,j,k}|\mathcal{N}_0) = \mathrm{Var}(X_{i,j,k}|N_{ij}) = \varphi\psi_{i,j,k}N_{i,j}. \tag{8}$$

Since all cells are, conditionally on the N_{ij}s, independent by assumption, it is straightforward to compute the RBNS variances by summing over appropriate indices. The RBNS variance for accident year i is

$$\begin{aligned}\mathrm{Var}(R_i^{\mathcal{R}}|\mathcal{N}_0) &= \sum_{j=0}^{m-i}\sum_{k>m-i-j}\mathrm{Var}(X_{i,j,k}|\mathcal{N}_0)\\ &= \varphi\sum_{j=0}^{m-i}\sum_{k>m-i-j}\psi_{i,j,k}N_{i,j}.\end{aligned} \tag{9}$$

The accident years are independent, so we can get the variance of the total outstanding RBNS payments by summing these variances over all accident years.

The IBNR variances are only slightly more complicated to calculate. By variance decomposition, the process variances of the outstanding IBNR payments, i.e. those $X_{i,j,k}$s with indexes $i + j > m$, are

$$\begin{aligned}\mathrm{Var}(X_{i,j,k}|\mathcal{N}_0) &= \mathrm{Var}(X_{i,j,k})\\ &= \mathrm{Var}(\mathbb{E}[X_{i,j,k}|N_{i,j}]) + \mathbb{E}[\mathrm{Var}(X_{i,j,k}|N_{i,j})]\\ &= \psi_{i,j,k}^2\,\mathrm{Var}(N_{i,j}) + \varphi\psi_{i,j,k}\mathbb{E}[N_{i,j}]\\ &= \psi_{i,j,k}^2\phi\nu_{i,j} + \varphi\psi_{i,j,k}\nu_{i,j}\\ &= (\psi_{i,j,k}\phi + \varphi)\psi_{i,j,k}\nu_{i,j}.\end{aligned} \tag{10}$$

Since we are not conditioning on the N_{ij}s for the IBNR claims, $X_{i,j,k}$ and $X_{i,j,k'}$ are dependent. Therefore we may not add these variances together as we did for the RBNS claims. Instead, the process variance for the outstanding IBNR payments in

accident year i is given by

$$\begin{aligned}
\mathrm{Var}(R_i^I \,|\mathcal{N}_0) &= \sum_{j=m-i+1}^{m-1} \mathrm{Var}\left(\sum_k X_{i,j,k}\right) \\
&= \sum_{j=m-i+1}^{m-1} \left(\mathbb{E}\left[\mathrm{Var}\left(\sum_k X_{i,j,k}\middle| N_{i,j}\right)\right] + \mathrm{Var}\left(\mathbb{E}\left[\sum_k X_{i,j,k}\middle| N_{i,j}\right]\right)\right) \\
&= \sum_{j=m-i+1}^{m-1} \left(\mathbb{E}\left[\sum_k \mathrm{Var}\left(X_{i,j,k}\middle| N_{i,j}\right)\right] + \mathrm{Var}\left(\sum_k \psi_{i,j,k} N_{i,j}\right)\right) \\
&= \sum_{j=m-i+1}^{m-1} \left(\mathbb{E}\left[\sum_k \varphi\psi_{i,j,k} N_{i,j}\right] + \left(\sum_k \psi_{i,j,k}\right)^2 \phi\nu_{i,j}\right) \\
&= \sum_{j=m-i+1}^{m-1} \left(\varphi + \phi \sum_k \psi_{i,j,k}\right) \nu_{i,j} \sum_k \psi_{i,j,k} \qquad (11)
\end{aligned}$$

As for the RBNS variances, these IBNR variances can be added together to get the variance of the total outstanding payments from IBNR claims. Finally, since the RBNS and IBNR claims are independent, to get the variance of the total outstanding payments, we add the RBNS and IBNR variances together. Now we have all the ingredients to compute the process variance, and we, therefore, move on to estimating the estimation error using a parametric bootstrap.

For the parametric bootstrap, we follow the algorithms in Section 6.2 in [21] to get bootstrap samples of the RBNS and IBNR reserve predictions in CRM-type models. These algorithms, adapted to our situation, is given at the end of this section. To follow these algorithms, we need to simulate new in-sample data. There are many possible ways of simulating from these ODP models. A standard approach is to use the following (which is used in e.g. [10] and [11]): If

$$\phi N_{i,j} \sim \mathrm{Po}(\nu_{i,j}/\phi),$$

then

$$N_{i,j} \sim \mathrm{ODP}(\nu_{i,j}, \phi).$$

That is, we may simulate from a Poisson distribution with mean $\nu_{i,j}/\phi$ and then multiply the observations with the dispersion parameter ϕ to get a random sample from $\mathrm{ODP}(\nu_{i,j}, \phi)$. For this, however, we need an estimator of ϕ. Here we will use the standard one based on the Pearson statistic (see e.g. p. 328 in [20]) which, for the

number of reported claims part of the model, is given by

$$\widehat{\phi} := \frac{1}{n - p_\nu} \sum_{i+j \le m} \frac{(N_{i,j} - \widehat{\nu}_{i,j})^2}{\widehat{\nu}_{i,j}},$$

where p_ν is the number of parameters ($p_\nu = 2m - 1$ for a cross-classified structure), and n is the number of observations in the upper left triangle. The estimator $\widehat{\varphi}$ is defined analogously as

$$\widehat{\varphi} := \frac{1}{n - p_\psi} \sum_{i+j+k \le m} \frac{(X_{i,j,k} - \widehat{\psi}_{i,j,k} N_{i,j})^2}{\widehat{\psi}_{i,j,k} N_{i,j}},$$

where n now is the sample size of the $X_{i,j,k}$s and p_ψ the number of parameters used in estimating the $\psi_{i,j,k}$s, which would be $3m - 2$ in a cross-classified model structure.

By computing the process variances according to (9) and (11), and bootstrapping the estimation error using the below algorithms, we have all the ingredients needed to estimate the conditional MSEP. To compute the estimation error for the total reserve, we use the bootstrap samples acquired in Step 6 in the two algorithms below by computing the sample average

$$\frac{1}{B} \sum_{b=1}^{B} (\widehat{R}^{\mathcal{R}} + \widehat{R}^{\mathcal{I}} - (\widehat{R}^{\mathcal{R},*}_{(b)} + \widehat{R}^{\mathcal{I},*}_{(b)}))^2$$

The following algorithm follows the underlying principles from [21]:

Algorithm RBNS

Step 1. *Estimation of the parameters.* Estimate the payment part parameters of the model using the original data to get the estimators $\widehat{\psi}_{i,j,k}$ and $\widehat{\varphi}$.

Step 2. *Bootstrapping the data.* Keep the same counts $N_{i,j}$, but generate new bootstrapped aggregated payments $\{X^*_{i,j,k} : i+j+k \le m\}$ by simulating from $\text{Po}(\widehat{\psi}_{ijk} N_{ij} / \widehat{\varphi})$ and multiplying by $\widehat{\varphi}$.

Step.3 *Bootstrapping the parameters.* Compute the estimators $\widehat{\psi}^*_{i,j,k}$ using $\{N_{i,j} : i + j \le m\}$ and the bootstrap data $\{X^*_{i,j,k} : i + j + k \le m\}$.

Step 4. *Bootstrapping the RBNS predictions.* Using the original incurred claims $\{N_{i,j} : i + j \le m\}$ and the bootstrap parameters $\widehat{\psi}^*_{i,j,k}$, compute the RBNS reserve prediction $\widehat{R}^{\mathcal{R},*}$ according to (4).

Step 5. *Monte Carlo approximation.* Repeat steps 2-4 $\boldsymbol{B}$ times to get an approximate bootstrap distribution of the RBNS reserve from the bootstrapped $\{\widehat{R}^{\mathcal{R},*}_{(b)}\}_{b=1}^{B}$.

Algorithm IBNR

Step 1. *Estimation of the parameters.* Estimate the parameters of the model using the original data to get the estimators $\widehat{\nu}_{i,j}$, $\widehat{\psi}_{i,j,k}$, $\widehat{\phi}$, and $\widehat{\varphi}$.

Step 2. *Bootstrapping the data.* Generate new bootstrapped data $\{N^*_{i,j} : i+j+k \leq m\}$ and $\{X^*_{i,j,k} : i+j+k \leq m\}$ by simulating the $X^*_{i,j,k}$s exactly as described in Step 2 of the RBNS algorithm above and the $N^*_{i,j}$s by simulating from $\text{Po}(\widehat{\nu}_{i,j}/\widehat{\phi})$ and multiplying by $\widehat{\phi}$.

Step.3 *Bootstrapping the parameters.* Compute the estimators $\widehat{\psi}^*_{i,j,k}$ using $\{N_{i,j} : i+j \leq m\}$ and the bootstrap data $\{X^*_{i,j,k} : i+j+k \leq m\}$, and compute the estimators $\widehat{\nu}^*_{i,j}$ using $\{N^*_{i,j} : i+j \leq m\}$.

Step 4. *Bootstrapping the IBNR predictions.* Using the bootstrap parameters $\widehat{\nu}_{i,j}$ and $\widehat{\psi}_{ijk}$, compute the IBNR reserve prediction $\widehat{R}^{\mathcal{I},*}$ according to (5).

Step 5. *Monte Carlo approximation.* Repeat steps 2-4 $\boldsymbol{B}$ times to get an approximate bootstrap distribution of the IBNR reserve from the bootstrapped $\{\widehat{R}^{\mathcal{I},*}_{(b)}\}_{b=1}^{B}$.

REFERENCES

[1] Katrien Antonio and Richard Plat. Micro-level stochastic loss reserving for general insurance. *Scandinavian Actuarial Journal*, 2014(7):649–669, 2014.

[2] Elja Arjas. The claims reserving problem in non-life insurance: Some structural ideas. *ASTIN Bulletin: The Journal of the IAA*, 19(2):139–152, 1989.

[3] John T Bonsignore, Joseph O Marker, Julie Sims, Yisheng Bu, Gary V Nickerson Greg Taylor, Sandie Cagley, Bruce E Ollodart, Gary G Venter, David R Clark, Dianne M Phelps, et al. The analysis and estimation of loss & alae variability: A summary report. 2005.

[4] H Bühlmann, Rene Schnieper, and Erwin Straub. Claims reserves in casualty insurance based on a probabilistic model. *Bulletin of the Swiss Association of Actuaries*, 80:21–45, 1980.

[5] Massimo De Felice and Franco Moriconi. Claim watching and individual claims reserving using classification and regression trees. *Risks*, 7(4):102, 2019.

[6] Francis Duval and Mathieu Pigeon. Individual loss reserving using a gradient boosting-based approach. *Risks*, 7(3):79, 2019.

[7] Bradley Efron and Trevor Hastie. *Computer age statistical inference*, volume 5. Cambridge University Press, 2016.

[8] Peter D England and Richard J Verrall. Stochastic claims reserving in general insurance. *British Actuarial Journal*, 8(3):443–518, 2002.

[9] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[10] Andrea Gabrielli. A neural network boosted double overdispersed poisson claims reserving model. *ASTIN Bulletin: The Journal of the IAA*, 50(1):25–60, 2020.

[11] Andrea Gabrielli, Ronald Richman, and Mario V Wüthrich. Neural network embedding of the over-dispersed poisson reserving model. *Scandinavian Actuarial Journal*, pages 1–29, 2019.

[12] Andrea Gabrielli and Mario V Wüthrich. An individual claims history simulation machine. *Risks*, 6(2):29, 2018.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. Available at http://www.deeplearningbook.org.

[14] Trevor Hastie, Jerome Friedman, and Robert Tibshirani. *The elements of statistical learning*. Springer series in statistics New York, NY, USA:, 2008.

[15] Kevin Kuo. Deeptriangle: A deep learning approach to loss reserving. *Risks*, 7(3):97, 2019.

[16] Christian Roholte Larsen. An individual claims reserving model. *ASTIN Bulletin: The Journal of the International Actuarial Association*, 37(01):113–132, 2007.

[17] Mathias Lindholm and Richard Verrall. On distribution-free reserving and partial information. *Stockholm university research reports in Mathematical statistics*, 2019:14, 2019.

[18] Olivier Lopez, Xavier Milhaud, Pierre-E Thérond, et al. Tree-based censored regression with applications in insurance. *Electronic journal of statistics*, 10(2):2685–2716, 2016.

[19] Thomas Mack. Distribution-free calculation of the standard error of chain ladder reserve estimates. *ASTIN Bulletin: The Journal of the IAA*, 23(2):213–225, 1993.

[20] Peter McCullagh and John A Nelder. *Generalized linear models*. Chapman & Hall/CRC, 1989.

[21] María Dolores Martínez Miranda, Bent Nielsen, Jens Perch Nielsen, and Richard Verrall. Cash flow simulation for a model of outstanding liabilities based on claim amounts and claim numbers. *ASTIN Bulletin: The Journal of the IAA*, 41(1):107–129, 2011.

[22] María Dolores Martínez Miranda, Jens Perch Nielsen, and Richard Verrall. Double chain ladder. *ASTIN Bulletin: The Journal of the IAA*, 42(1):59–76, 2012.

[23] Ragnar Norberg. Prediction of outstanding liabilities in non-life insurance 1. *ASTIN Bulletin: The Journal of the IAA*, 23(1):95–115, 1993.

[24] Arthur E Renshaw and Richard J Verrall. A stochastic model underlying the chain-ladder technique. *British Actuarial Journal*, 4(4):903–923, 1998.

[25] Greg Taylor. Loss reserving models: Granular and machine learning forms. *Risks*, 7(3):82, 2019.

[26] Richard Verrall, Jens Perch Nielsen, and Anders Hedegaard Jessen. Prediction of rbns and ibnr claims using claim amounts and claim counts. *ASTIN Bulletin: The Journal of the IAA*, 40(2):871–887, 2010.

[27] Felix Wahl, Mathias Lindholm, and Richard Verrall. The collective reserving model. *Insurance: Mathematics and Economics*, 87:34–50, 2019.

[28] Mario V Wüthrich. Machine learning in individual claims reserving. *Scandinavian Actuarial Journal*, 2018(6):465–480, 2018.

[29] Mario V Wüthrich. Neural networks applied to chain–ladder reserving. *European Actuarial Journal*, 8(2):407–436, 2018.

[30] Mario V Wüthrich and Michael Merz. Yes, we cann! *ASTIN Bulletin: The Journal of the IAA*, 49(1):1–3, 2019.