A comparative Analysis Between Various Machine Learning Models and Generalized Linear Models

Jan Mikael Yousif

Masteruppsats 2023:3
Försäkringsmatematik
Februari 2023

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Mathematical Statistics
Stockholm University
Master Thesis **2023:3**
http://www.math.su.se

A comparative Analysis Between Various Machine Learning Models and Generalized Linear Models

Jan Mikael Yousif*

February 2023

Abstract

In recent decades there has been a vast improvement in computational power which has lead to an increased demand for advanced modelling techniques such as Gradient Boosting Machines and Neural Networks. This thesis studies if the mentioned models have the ability to predict the claim frequency for an insurance portfolio more accurately than a traditional Generalized Linear Model (GLM). By training and Cross-Validating the mentioned models the thesis shows that the Machine- Learning models do perform better than the GLM for a data-source from a real insurance portfolio. The improvements that the Machine-Learning models resulted in were initially expected to be of greater magnitude but showed to only have a slight difference from the GLM. This is largely explained by the GLM already being a good predictor for the specified data. It is also observed that further advantages can be obtained by partitioning the data by the feature levels where the models performs the best. By training each model on the partitioned data, the weaknesses in each model are minimized while the strengths are more highlighted. Mixing models across partitioned data however leads to a large cost in interpretability which is often an important factor to consider when building price strategies in real-world applications.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: janyousif@gmail.com. Supervisor: Filip Lindskog.

Before any material of this thesis is presented I would like to dedicate my personal gratitude to my supervisor, Filip Lindskog!

The assistance he provided during the course of this semester has been invaluable due to his expertise and genuine will for his students to thrive. During the time it took to study, analyse and write the material in this report Filip never once refrained from giving a helping hand, regardless of what day of the week or what holiday it was and for that I truly think that he deserves to be thanked. I would also like to thank my wife, Gabriella, and my family for the tremendous support during this time.

Contents

1 Introduction

1.1 Background

Insurance as we know it is an old concept that has been an important part of any thriving society during the past centuries but there are also historical references of institutions which might be loosely regarded as insurance institutions that dates as far back as the Babylonian era (roughly 1500 BC). The age of enlightenment of the 17th and 18th century gave grounds for insurers to accept that actuarial science could provide means to conduct better business. Behind this innovation was a belief that the world and its possible future states could be predicted by analysing reoccurring patterns. The history of insurance is quite interesting, it seems as whenever a society reaches a prosperity level to the point where food is no longer a scarce resource, that same society will also have a rising demand for insurance-like protection. For more on the insurance history we refer the interested reader to [16]. In modern times insurance is widely accessible to most consumers and there are also various solutions that tailor to the individuals needs. Insurance policies can cover almost anything of value such as homes, lives, vehicles, livestock, electronics, pets, income, businesses and a wide array of special-case contracts.

Insurance is often viewed as a means of protecting valuable assets such as vehicles and estates. However, in reality, it can be considered an indirect necessity that is hard to ignore since high-value possessions are often financed through loans. As a result, the borrower is left in a position where damages to these assets becomes difficult to repair with personal funds. Insurance policies help to spread the risk between individual policyholders by utilizing the law of large numbers; it states that the the average of a given sample will converge towards the samples expected value as the sample size grows. The key idea for managing an insurance portfolio is that as the number of active policies increases, the average cost of the portfolio will reach a plateau. As a result, the cost per capita decreases and stabilize over the time as the number of policies increases.

Pricing has always been a question of great importance to any business and the insurance industry is no exception. In insurance however, pricing is not only the key ingredient to the revenue stream but it is also the most important tool for ensuring that each policyholder pays a fair price in regards to the risk that his/her asset is associated with. A primitive but somewhat effective way to price a portfolio is to average the cost per policy and apply a flat rate for the entire portfolio but as the portfolio grows it usually start to form varying levels of risk for different segments within the data, this leads to the question if it truly is fair to price policyholders with lower level of risk the same rate as the policyholders who add more strain to the collective. An example could be homeowners who deliberately purchase properties near areas that are known for flooding versus home-owners who purchase properties in more stable regions. The question of fairness could possibly be discarded as an ethical discussion but once we add competitive aspects to the equation we instead observe a growing demand for pricing that better reflects the risks of the policyholders. While it is true that an insurance company in a monopoly has little need for price discrimination, since the consumers only choice would be to pay the price anyway, the playing field changes when insurance companies compete for market shares. If one company were to suddenly decrease the premiums of low-risk segments making changes in the protective aspects of the policy, then given enough time, the affected consumers are likely to sign up for the cheaper policy. This will force all other companies to either start matching their prices or to risk falling into adverse selection spirals.

During the late-parts of the 20:th century insurance companies often used to price their portfolios by assessing the risk for general data-segments through the usage of Generalized Linear Models, however, the depth of the modelling process was often limited to the technology of the time period. The analytical abilities of the day are of course also limited to what technology has to offer but as the power of computing has improved greatly during the past decades so has the possibilities for large-scale analysis on great amounts of data. Generalized Linear Models has been the standard pricing tool during the 21st century and the modelling framework has proven itself to be robust, predictive, stable over time and easy to interpret which are some key attributes that are of importance for an insurance business. However, alongside an exponential increase in computing power of the past decades a demand for more advanced methods of modelling have risen, namely the demand for tariffs that are based on Machine-Learning algorithms. The increased demand is partly explained by the idea of whoever that has the best pricing strategy also has the best likelihood of minimizing the risk of the portfolio.

1.2 Aim & Purpose

This thesis studies the effects of using advanced Machine-Learning frameworks such as Neural Networks and Gradient Boosting Machines for an insurance portfolio in comparison to Generalized Linear Models in order to see if any statistical advantages in favour of Machine-Learning models can be found. While it is commonly accepted in a more general sense that Machine-Learning models are more appropriate for modelling data with high complexity one also have to account for external factors such as regulations that might pose a problem for applying the advanced methods in reality. Regulatory frameworks will be largely ignored in this thesis since the question of interest is of a statistical kind, while this thesis places a large focus on how well models perform in comparison to each other there will also be some general discussions regarding the pros and cons of advanced modelling techniques.

The Machine-Learning models in this thesis are often regarded as black-box models since they are typically not easy to interpret. The Generalized Linear Model generally predicts data through linear transformations while Machine-Learning model will often sequentially weight input data until some desired output is produced, depending on what model architecture is used the complexity of the weighting sequence can vary. The mentioned difference in architecture is a common critique towards Machine-Learning models. One can ask why it is important to have deep understanding of the path that the data takes before being transformed into predictions and the answer to this is quite clear; low understanding of what your model does can often lead to explanatory factors being missed out or being exaggerated. While linear models in general offer multitude of analytical tools that help bridge the gaps between the input and output in order to help the analyst understand deeper relationships within the underlying data the same cannot be said for Machine-Learning models. There is some validity in the criticism towards Machine-Learning models since a general rule of thumb is that the analyst should understand the models thoroughly before drawing conclusions but in all fairness this might sometimes be easier said than done. From the other hand, however, a probably equally valid counter-criticism could be that the in-between analysis perhaps is not that important for some types of data if the black-box model performs significantly better. As an example, if a certain well-trained Neural Network has a 98% likelihood to adequately predict the contents of an image, does it really matter if the architecture hard to interpret?

There is a natural duality that appears when modelling data, from one perspective it might be interesting to push the limits and see what improvements advanced modelling can produce while a counter-perspective is that one should also bear in mind that different kinds of models can vary in performance for different sets of data. It should in other words be kept in mind that the model should fit the data and not the other way around. A simple straight line, perhaps even drawn by hand or interpolated between two well-placed points might also be the best predictor depending on how the data is comprised. This thesis will argue for both sides of the spectrum and seek to bridge the gap between advanced and simpler modelling techniques in order to show that benefits from both methods can be found.

2 Dictionary

Some of the terms used in this thesis is commonly used in the insurance sector and might not be considered common knowledge for a general reader. The contents in this section seeks to summarize and explain some general language terms and language that will be of interest in this thesis.

Insurer - An institution that offers financial security in case of damage to valuable assets.

Policy - A contract between the policy holder and the insurer that specifies the terms of the financial protections.

Portfolio - A group of active policies.

Claim Frequency - The average number of claims per time unit for an arbitrary insurance portfolio.

Claim Severity - The average claim size for an arbitrary insurance portfolio.

Pure Premium/Risk Premium - The average cost per policy for an arbitrary insurance portfolio.

Tariff - A set of rules that defines the price for a policy.

Duration - The number of insured years for a given policy.

Premium - The price payed by the policyholder.

Covariate/Feature - Statistical variables used to explain a response variable.

3 Theory

3.1 Generalized Linear Models (GLM)

The Generalized Linear model is the standard model architecture that is used not only in the insurance industry but also in many other fields of application. The simplicity of the model structure allows for intuitive and understandable results while simultaneously allowing for a wide range of variety in tools that help improve the fit.

Model Assumptions:

Exposure (w)	Respones (X)	Key Ratio Y=X/w
Duration	No. Claims	Claim Frequency
Duration	Claim Cost	Risk Premium
No. Claims	Claim Cost	Claim Severity
Earned Premium	Claim Cost	Loss Ratio

Table 3.1 - Key Exposures

The following assumptions are referred to section 1.2 of [1].

Assumption 1 - Policy Independence:

Let n be the number of unique policies in a portfolio. For any key ratio in Table 3.1 let X_i denote the response for policy i. Then $X_1, \cdots, X_n$ are assumed to be independent from each other. Although this assumption is an important part of the GLM some examples where this is not the case can be identified. The textbook example explains vehicle collision involving two cars in which both are insured in at the same insurance company, it follows quite logically that the two separate claims that get registered in the data will be dependent on the other. This phenomenon is usually rare and of no importance to the overall portfolio.

Assumption 2 - Time Independence:

Let n specify the number of disjoint time intervals between two periods of time. For any response in Table 3.1, let X_i denote the response in time interval i. Then $X_i, \cdots, X_n$ are independent

Assumption 3 - Homogeneity:

Consider any two policies in the same tariff cell with the same exposure. For any response in Table 3.1, let X_i denote the response for policy i. Then X_1 and X_2 will share the same probability distribution.

3.1.1 Exponential Dispersion Models

The most common method of pricing an insurance portfolio involves Exponential Dispersion models (EDM for short) which can be described as generalized cases of the normal distribution, see Section 2.1 in [1]. The probability distribution of an EDM is defined as the following expression.

$$f_{Y_i}(y_i;\theta_i,\phi) = \exp\left\{\frac{y_i\theta_i - b(\theta_i)}{\phi/\omega_i} + c(y_i,\phi,\omega_i)\right\}, \tag{1}$$

where θ_i is a parameter that is allowed to depend on i and the dispersion parameter $\phi > 0$ is the same for all i (see Section 2.1 in [1]). The cumulant function $b(\theta_i)$ is assumed to be twice differentiable and $b'(\theta_i)$ is invertible, given a choice of such a function we then get a family of probability distributions, further sections in this thesis will explore how EDM:s can be applied for Poisson- and Gamma Distributions.

3.1.2 Poisson Case

Section 2.1.1 in [1] shows that if we let X_i denote the number of claims in a tariff cell with duration w_i and the expectation μ_i at $w_i = 1$, we then have that the expected value will be equal to the product of w_i and μ_i. It then follows that the distribution function for X_i is equal to the following expression.

$$f(x_i;\mu_i) = e^{-w_i\mu_i}\frac{(w_i\mu_i)^{x_i}}{x_i!}. \tag{2}$$

When modelling claims it is usually more convenient to work with the claim frequency rather than the number of claims per se. We define the claim frequency as $Y_i = \frac{X_i}{w_i}$ where we also acknowledge that $X_i = Y_i w_i$, We are then left with the following probability distribution.

$$f_{Y_i}(y_i;\mu_i) = P(Y_i = y_i) = P(X_i = w_i yi) = e^{-w_i\mu_i}\frac{(w_i\mu_i)^{w_i y_i}}{(w_i y_i)!}. \tag{3}$$

Rearranging (3) algebraically we see that $f_{Y_i}(y_i;\mu_i)$ can be written as the following expression.

$$\begin{align} f_{Y_i}(y_i;\mu_i) &= \exp\{-w_i\mu_i\}\ \exp\left\{\log\left(\frac{w_i\mu_i^{w_i y_i}}{(w_i y_i)!}\right)\right\} \tag{4}\\ &= \exp\{-w_i\mu_i + w_i y_i(\log(w_i\mu_i) - \log(w_i y_i!))\} \tag{5}\\ &= \exp\{-w_i\mu_i + w_i y_i\log(\mu_i) - w_i y_i(\log(w_i) - \log(w_i y_i!))\} \tag{6}\\ &= \exp\{w_i(y_i\log(\mu_i) - \mu_i) + w_i y_i\log(w_i) - \log(w_i y_i!)\}. \tag{7}\end{align}$$

If we set $\theta_i = \log(\mu_i)$ and note that $w_i y_i\log(w_i) - \log(w_i y_i!)$ can be rewritten as $c(y_i,\phi,w_i)$ where $\phi = 1$ we end up with a probability function that fits the EDM-structure.

$$f_{Y_i}(y_i;\mu_i) = \exp\{w_i(y_i\theta_i - e^{\theta_i}) + c(y_i,\phi,w_i)\}. \tag{2}$$

3.1.3 Gamma Case

Section 2.1.2 in [1] shows that if we let the X denote the claim cost in a tariff cell with ω number of claims and we can define the claim severity as $Y = \frac{X}{\omega}$, this is often modelled following a gamma distribution. Generally speaking there are more than just one distribution that could fit the claim severity but the gamma case is a standard baseline model. The gamma distribution is defined by the density function,

$$f(x) = \frac{\beta^{\alpha}}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x} \; ; \quad x > 0. \tag{8}$$

Thus, for the claim severity case we here receive the following density function,

$$f_Y(y) = w f_X(wy) = \frac{w \; \beta^{\; w\alpha}}{\Gamma(w\alpha)} y^{w\alpha-1} e^{-w\beta y} \; ; \quad x > 0, \tag{9}$$

and so we have that $Y \sim \Gamma(wa, wb)$. If we set u and ϕ such that $u = \frac{\alpha}{\beta}$ and $\phi = 1/\alpha$ we can rewrite the contents of (9) as follows:

$$\begin{aligned} f_Y(y) &= f_Y(y; \mu, \phi) && (10) \\ &= \frac{1}{\Gamma(w/\phi)} \left(\frac{w}{\mu\phi} \right)^{w/\phi} y^{(w/\phi)-1} e^{-wy/(\mu\phi)} && (11) \\ &= \exp\left\{ \frac{1 - y/\mu - log(\mu)}{\phi/w} + c(y, \phi, w) \right\}, && (12) \end{aligned}$$

where $c(y, \phi, w) = \frac{\log(wy/\phi)}{w/\phi} - \log(y) - \log(\Gamma(w/\phi))$. If we specify θ such that $\theta = -1/\mu$ we are then left with a density function that fits the EDM-structure.

3.1.4 Maximum Likelihood for EDM:s

An important part of modelling insurance data is that the model is expressed in a multiplicative form instead of an additive since multiplicative tariffs are seen as more reasonable when it comes to setting price strategies. The multiplicative transformation is done by a logarithmic link function that is also known as the *Log-Link*. The idea here is to link the general results of the linear model $\eta_i = \sum_j^n x_{ij}\beta_j$ through a logarithmic function $g(\mu_i) = \log(\mu_i)$ such that $g(\mu_i) = \sum_j^n x_{ij}\beta_j$. We refer the reader to section 2.2 in [1] for more details on link functions.

From Section 2.3.2 in [1] we see that it follows from (1) that the log-likelihood for for an EDM is given by

$$l(\theta; \phi, y) = \frac{1}{\phi} \sum_i w_i(y_i\theta_i - b(\theta_i)) + \sum_i c(y_i, \phi, w_i). \tag{13}$$

From (13) we see that the dispersion parameter has no effect on the general result of the maximization with respect to θ so it can disregarded. We can express the likelihood as a function of β_j instead of θ_j by using the inverse of $\mu_i = b'(\theta_i)$ together with the link $g(\mu_i) = \eta_i = \sum_j x_{ij}\beta_j$ to get the following derivative.

$$\begin{aligned}
\frac{\mathrm{d}\, l}{\mathrm{d}\, \beta_j} &= \frac{\sum_i \mathrm{d}\, l}{\mathrm{d}\, \theta_i}\frac{\mathrm{d}\, \theta_i}{\mathrm{d}\, \beta_j} && (14)\\
&= \frac{1}{\phi}\sum_i (w_i y_i - w_i b'(\theta_i))\frac{\mathrm{d}\, \theta_i}{\mathrm{d}\, \beta_j} && (15)\\
&= \frac{1}{\phi}\sum_i (w_i y_i - w_i b'(\theta_i))\frac{\mathrm{d}\, \theta_i}{\mathrm{d}\, \mu_i}\frac{d\mu_i}{\mathrm{d}\, \eta_i}\frac{\mathrm{d}\, \eta_i}{\mathrm{d}\, \beta_j}. && (16)
\end{aligned}$$

Since $\mu_i = b'(\theta_i)$ we have that $\frac{\mathrm{d}\,\mu_i}{d\theta_i} = b''(\theta_i) = \frac{1}{v(\mu_i)}$ and $\frac{\mathrm{d}\,\mu_i}{\mathrm{d}\,\eta_i} = \left(\frac{d\eta_i}{d\mu_i}\right)^{-1} = \frac{1}{g'(\mu_i)}$ (see Section 2.3.2 in [1]). From $\eta_i = \sum_j x_{ij}\beta_j$ we get that $\frac{d\eta_i}{d\beta_j} = x_{ij}$ and we finally get the following result.

$$\frac{\mathrm{d}\, l}{\mathrm{d}\, \beta_j} = \frac{1}{\phi}\sum_i w_i \frac{y_i - \mu_i}{v(\mu_i)g'(\mu_i)} x_{ij}. \qquad (17)$$

The expression in (17) is what is commonly known as the *score function* which yields the ML-estimation through solving the following expression.

$$\frac{1}{\phi}\sum_i w_i \frac{y_i - \mu_i}{v(\mu_i)g'(\mu_i)} x_{ij} = 0 \iff \sum_i w_i \frac{y_i - \mu_i}{v(\mu_i)g'(\mu_i)} x_{ij} = 0, \qquad (18)$$

where $\mu_i = g^{-1}(\eta_i) = g^{-1}\left(\sum_j x_{ij}\beta_j\right)$. When modelling EDM:s we we generally define $v(\mu) = \mu^p$ and $g(\mu_i) = \log(\mu_i)$, see Section 2.1.4 in [1], which gives us the general expression:

$$\sum_i w_i \frac{y_i - \mu_i}{\mu_i^{p-1}} x_{ij} = 0., \qquad (19)$$

Here it is important to remember that μ_i in (19) is a function of β that has to satisfy $\mu_i = g^{-1}(\eta_i)$

3.1.5 Likelihood-Ratio Test

It can be said that one of the more important parts of building a a model is to validate how well it performs on the data that it is trained on. While an arbitrary data source can contain multiple covariates there is often no guarantee that all or even some of the covariates will significantly effect the model performance. A common praxis is to compare how well two models fit the training data and then deciding which one is the most suitable to continue with, logically it should follow that performing repetitive tests should lead to a valuable model given that the data is qualitative. This sort of testing where two models are compared to each other are often known as hypothesis testing where some initial Hypothesis H_0 (also known as the Null Hypothesis) is assumed and will either be dismissed in favour of an alternative hypothesis H_1 or vice versa.

The Likelihood-Ratio Test (LRT for short) is a commonly used as an aid to decide if a covariate should be kept or excluded from a model, by assuming a null hypothesis which states that data is explained by the reduced model the LRT can show if the hypothesis is likely or not. From section 3.1.2 in [1] we define the LRT as follows. Consider two models H_r and H_s such that $H_s \subset H_r$. Let $\hat{\mu}^{(r)}$ be defined as the MLE:s under H_r and vice versa for H_s. The LRT statistic is then given by $D(y, \hat{\mu}^{(s)}) - D(y, \hat{\mu}^{(r)})$ where $D(\cdot)$ is a deviance function properly specified for the a given distribution. For the poisson case we define the LRT as the following expression.

$$D(y, \hat{\mu}^{(s)}) - D(y, \hat{\mu}^{(r)}) = 2 \sum_i w_i y_i \log \left(\frac{\hat{\mu_i}^{(r)}}{\hat{\mu_i}^{(s)}} + \sum_i w_i (\hat{\mu_i}^{(s)} - \hat{\mu_i}^{(r)}) \right). \tag{20}$$

Under general assumptions the LRT is $\chi^2(f_r - f_s)$ distributed where f_r and f_s are the number of non-redundant parameters for said models.

3.1.6 Akaike information Criterion

When two GLM:s (or more) show somewhat comparable results and you are unsure on which one to continue with then a good idea can be to include the Akaike Information Criterion (AIC for short) into the analysis. While the LRT is a good way to measure how well the model is trained it does not account for the model complexity so only using the LRT does not guarantee that overfitting through a large amount of parameters is avoided. The AIC is designed to be a comparative tool that also penalize larger models by including a term that is dependent on the number of covariates. The idea is that if two models show a similar level of performance then the model with the highest AIC value should be discarded. It is however worth to mention that the *AIC* should probably not be used as the sole metric to analyse the goodness of fit since it is a comparative metric that does not give much information on its own. If for example there are multiple models to compare the AIC will only tell which of these models that is the most appropriate, it does not say how well the models actually fit the data so in reality all models might perform poorly and the AIC might still suggest the best model among them.

From section 6.4 in [12] we define the AIC as follows, let n be the number of estimated parameters in a given model. Let $\hat{L}$ be the maximized value of the likelihood function for the given model. The AIC is then defined as follows:

$$AIC = 2n - \ln(\hat{L}). \tag{21}$$

The AIC is a measure that is dependent on the maximum-likelihood estimate which in turn means that its actual magnitude can vary greatly depending on the data that is being modelled, in other words it should be kept in mind that the magnitude of the AIC is only relevant when being compared to the outcome from similar models.

3.2 Machine-Learning Models

While there are many types of model frameworks that can fit into the description of Machine-Learning this thesis will only cover a select few, namely Gradient Boosting Machines (GBM:s) and Neural Networks (NN:s). Even though the term "Machine-Learning" has already been mentioned in previous sections this comment is intended as a small reminder that any reference to it only refers to Gradient Boosting Machines and Neural Networks.

3.2.1 Gradient Descent

Gradient descent is an iterative optimization process that minimizes an arbitrary loss function $\mathcal{L}$ and is often used to train machine-learning models. The idea in its essence is to slightly nudge a given set of model parameters towards some value that minimizes the loss. In section 10.10.1 in [11] the gradient descent method is defined as follows.

Let $f(x_i)$ be model the function to be optimized and let $\mathcal{L}(f) = \sum_i^N \mathcal{L}(y_i, f(x_i))$ be an arbitrary loss function. It can be difficult to specify a loss function for the general case but it should essentially be a function that expresses the deviance between predictions and observations. A fairly common choice is the *MSE*. Consider a vector f containing the predictive parameters of the function $f(x_i)$. By differentiating t he loss function $\mathcal{L}(f)$ with respect to $f(x_i)$ we can tell which direction that the f that minimizes the loss can be found, so if we repeatedly just "push" f with a small step γ (aka a learning rate) towards the desired direction we should in theory, after a predetermined number of iterations, obtain an f that gives an optimal loss. More formally we define the following expression.

$$f_n = f_{n-1} - \gamma g_n, \tag{22}$$

where

$$g_n = \left[\frac{\mathrm{d}\,\mathcal{L}(y_i, f(x_i))}{\mathrm{d}\,f(x_i)}\right]_{f(x_i)=f_{n-1}(x_i)}. \tag{23}$$

A fair question that might be asked is why we subtract γg_n from f in (22). To speak in simpler terms, this is merely to ensure that the nudging is done in the direction that truly heads towards the minimum. If a given g_n is larger than the minimum then naturally it follows that the gradient will be positive and vice versa if the g_n is less than the minimum so in order to adjust f towards the right direction we need to subtract the gradient from it.

in summary we define the gradient descent with the following algorithm.

```
Step 1. Initialize $f_0$ with an arbitrary guess
Step 2.
**while** *Stopping requirement not met* **do**
    1. $g_n = \left[\frac{\mathrm{d}\,\mathcal{L}(y_i,f(x_i))}{\mathrm{d}\,f(x_i)}\right]_{f(x_i)=f_{n-1}(x_i)}$
    2. $f_n = f_{n-1} - \gamma \cdot g_n$
**end**
```

Algorithm 1: Gradient Descent

3.2.2 Tree Based Models

3.2.2.1 Regression Trees

All theory in this section is referred to Section 9.2.2 in [11] unless stated otherwise.

In general terms regression trees model data by partitioning the covariate space into regions and associating each region with a predictive value. Once the rules that define the partitioned regions are set, the model will simply pass new data through a decision tree that decide which region the data belongs to. The concept of tree-based methods are generally speaking simple but also can often prove to be a tool with powerful predictive abilities, but before digging deeper into the regression trees used in this thesis it could be advantageous to first conceptualize the tree-based methods.

A popular tree-based method is the model known as Classification And Regression Tree (CART for short). Let us consider a data source with a response Y and features X_1, X_2. If we define a set of points $t_1, \ldots, t_m$ that partitions the data into a set of regions $R_1, \ldots, R_m$ we can associate certain variations of the feature space to each R_i. What regression trees do is to try and predict the data in each region by the help of arbitrary predictions $c_1, \cdots, c_m$ which are assigned to each region, so what the model essentially is trying to say is that a data point x_{1i}, x_{2j} should be assigned the predictive value c_i because it is located in region R_i. Figure 3.1 illustrates this process with an example of 5 regions R_i and 4 cut-off points t_i for a hypothetical dataset, The left graph illustrates how the data is partitioned while the second graph illustrates how a decision-tree would assign input data accordingly. Since each region is associated with a predicted value c_i any input that is assigned to region R_i will consequently be assigned to the prediction c_i.

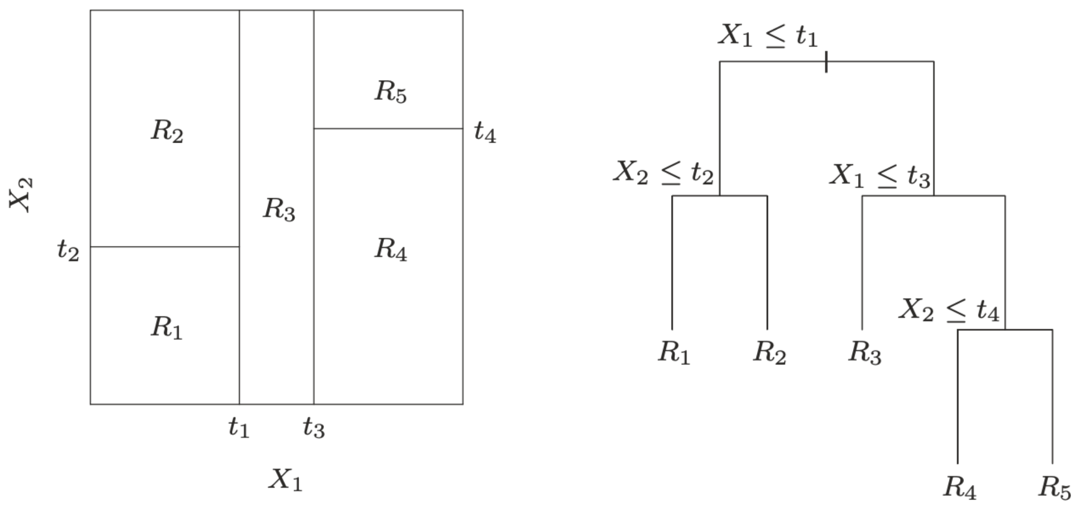


Figure 3.1 - An illustration of partitioned data (section 9.2.2 in [11])

While GLM:s makes predictions by weighting the input with regression coefficients $f(x) = \beta_0 + \sum_i^m x_i\beta_i$ the CART will instead make predictions by regional associations $f(x) = \sum_i^m c_i I(\{X_i, \ldots, X_m\} \in R_i)$ where $I(\cdot)$ is an indicator function for each region.

So far we have only mentioned how Regression Trees work in a general sense but we have yet explained how they are trained. Generally speaking an arbitrary data source will consist of p features and a response variable with potential room for a large amount of partitions, it would be rather impractical having to define all partitions by hand so the algorithm for growing a tree would have to have an automatic process that partitions the data into viable regions If we assume that we have a model $f(x)$ with M regions such that

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m), \tag{24}$$

where x is a p-dimensioned vector. We then get that the best estimation of c_m just amounts to the average of the response in region m.

$$\hat{c}_m = avg(y_i | x_i \in R_m). \tag{25}$$

We then initialize the process with a greedy algorithm that identifies which regions that are the most suitable by partitioning variable j at split point s for all the data, and if we define the partitioned halves $R_1(j,s) = \{X | X_j < s\}$ and $R_2(j,s) = \{X | X_j > s\}$ We then need to seek the variable j at splitting point s that solves the following expression.

$$\min_{\mathbf{j},\mathbf{s}} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right]. \tag{26}$$

For any choice of j and s; we solve the inner optimization problems by $\hat{c}_1 = avg(y_i | x_i \in R_1(j,s))$ and $\hat{c}_2 = avg(y_i | x_i \in R_2(j,s))$ for each partition variable, once the optimal split is identified we simply repeat the process for the resulting partitions until a stopping requirement has been met.

In other words the process simply iterates through all data points and features in order to then partition the data at that particular point, given this process it identifies which split point that minimizes the prediction error for the entire dataset. If one thinks about it for a second it does not take much to realize that such a process generates impractically large trees for datasets with many features and observations, Figure 3.1 illustrates an example with only two features and 5 regions but as the regions and/or features increase the portrayed tree will quickly grow larger. There are many ways to decide on the size of a tree and in Section 9.2.2 in [11] it is argued that one of the better ways is to first allow the tree to grow large, we can name this tree T_0, and then pruning it using a *cost-complexity* value. We define a subtree $T \subset T_0$ to be any tree that can be obtained by pruning T_0 and we index all terminal nodes by m where m represents the region R_m (as illustrated in 3.1), also let $|T|$ be the number of terminal nodes in T. we first define the following measures.

$$N_m = \#\{x_i \in R_m\} \quad \text{(No. Observations in region } R_m\text{)}, \tag{27}$$

$$\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_i} y_i, \tag{28}$$

$$Q_m(T) = \sum_{m=1}^{|T|} (y_i - \hat{c}_m)^2. \tag{29}$$

Using (27), (28) and (29) we then define the cost complexity criterion as follows:

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T|. \tag{30}$$

The idea here is to identify for each α, a sub-tree $T_\alpha \subset T_0$ that minimize $C_\alpha(T)$, it naturally follows that α regulates the tree size where larger values of α implies smaller trees and $\alpha = 0$ implies a fully grown tree. It is important to understand that α is not a parameter that is generated from the training process but instead a parameter that is adaptively chosen to optimize the prediction error, it can be shown that for each α there is a unique smallest sub-tree T_α that minimizes $C_\alpha(T)$. To find this particular tree we need to successively prune the internal nodes in T_0 that produces the smallest per-node increase in $\sum_{m=1}^{|T|} N_m Q_m(T)$ until we reach the root of the tree. This sequential pruning should lead to a sequence of sub-trees and this sequence must contain the sought after T_α (according to Section 9.2.2 in [11]) and to find T_α we simply perform a k-fold cross validation on all sub-trees available and then select the $\hat{\alpha}$ that minimizes the error.

3.2.2.2 Gradient Boosting Machine (GBM)

Gradient Boosting (GBM for short) is a framework that makes use of multiple models that by themselves can be considered weak models. In an illustrative sense it can be compared to a twig that by itself is easy to snap in half but when combined in an array of twigs it becomes significantly stronger. The key idea in the GBM framework is to combine multiple so-called weak learners until they together minimize a loss function $\mathcal{L}$ in a gradient descent process. The weak learners can be defined as almost any sort of model framework, some examples show that GLM:s and even fundamental NN:s can be used as weak learners but this thesis will only cover regression trees as it is the most common method which also have plenty of software support. While the end product of gradient boosting is a predictive model it should be noted that the GBM itself should be regarded as an algorithm since it is a framework that acts on existing model architectures.

Some key requirements for a generic gradient boosting process are the following:

- A weak learner $f(x)$ that is differentiable.
- A proper loss function $\mathcal{L}(y, f(x))$ for the data that describes the deviance predictions and and observations. A common choice of loss function is the *MSE*.
- A specified number of iterations M to limit the training process.

The last point can be clarified a bit, since the gradient boosting process is an iterative algorithm we need to specify when the iterations should cease. In theory one could also use a metric that illustrates the improvement for each iteration and stop the process once a level of satisfaction has been reached but for generic cases a manually defined iteration limit is sufficient.

From sections 10.10.2 and 10.10.3 in [11] we define the gradient boosting process as the following algorithm.

```
Step 1. Initialize $f_0(x)$ with a constant value as $f_0(x) = \arg\min_\gamma \sum_i^n \mathcal{L}(y, \gamma)$
Step 2.
for m=1 to M do
    1. for i=1 to N compute do
        $r_{im} = -\left[\frac{\mathrm{d}\,\mathcal{L}(y_i, f(x_i))}{\mathrm{d}\,f(x_i)}\right]_{f=f_{m-1}}$
    end
    2. Fit a regression tree to the targets $r_{im}$ giving terminal regions $R_{jm}, j = 1, \cdots, J_m$
    3. for j=1 to $J_m$ compute do
        $\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} \mathcal{L}(y_i, f_{m-1(x_i)} + \gamma)$
    end
    4. Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$
end
Step 3. Output $\hat{f}(x) = f_M(x)$
```

Algorithm 2: Gradient boosting

Since GBM:s naturally come with a large interpretability cost we will also introduce the *Relative Importance Factor* which is a measurement that expresses how much each feature contributes to the model as a whole. As mentioned earlier in Section 3.2.2.1, the general method to fit a regression tree is to let it grow into a Large (or full) tree and then prune away branches that does not make satisfactory contributions to the model, this logically implies that some features tend to contribute more than others. Hence, to add some interpretability to the model we use this metric to identify which features that are of importance to the model. From section 10.13.1 in [11] we define the relative importance as follows.

For a single regression tree we define the importance $\mathcal{I}_l^2(T)$ for each feature X_l as

$$\mathcal{I}_l^2(T) = \sum_{t=1}^{J-1} \hat{l}_t^2 I(v(t) = l). \tag{31}$$

The sum is meant to span the $J-1$ internal nodes of the tree and at each node t we know that one of the features $X_v(t)$ is used to partition the data associated with that particular feature. As explained in Section 3.2.2.1 each node is chosen by maximizing an improvement l_t^2 for a particular region and we define the variable importance as the sum of such improvements over all nodes. This is however for a single regression tree only and to expand this metric for an additive framework such as GBM:s we simply average the $\mathcal{I}_l^2$ across all trees. Since this metric is a relative measure it is considered customary to scale the importance factors around the greatest one such that they illustrate their effect as percentages of the total feature importance.

3.2.3 Artificial Neural Networks

3.2.3.1 Plain Neural Networks

Neural Networks (NN for short) are often in some loose sense described as a tree-structured models, largely because they often are mentioned along with GBM:s and similar frameworks, but it is however quite wrong to regard them as trees. A significant difference between trees and NN:s is that the way which data passes through the models is in some sense more fluid in the NN architechture than in the trees. While regression trees sequentially pass data through a set of binary distinctions in order to associate data with regions, the NN will instead feed input data through multiple layers of weights that are all connected to each other.

A common saying made originally by neuroscientist Donald Hebb is that neurons that fire together, also wire together. The idea behind this saying is that the human brain stores information by passing electrical signals through sequences of neurons and the idea behind NN:s is to mimic this behaviour. The generic model structure is designed around a set of artificial neurons that together weight data towards a desired output, since each weight acts differently for different values then the model essentially associates a "memory" for each unique input. Although the science behind how actual neurons work is not perfected we can still see that our artifical ones are quite powerful in remembering data patterns.

From section 6 in [2] we define a general NN as multiple functions that are connected in a chain so that the input from each function is the output from the previous. We specify L as the index of the output layer and we can then express the network function as a nested function of all layers as the following expression.

$$f(x) = f_L(f_{L-1}(f_{L-2}(....f_1(x)))), \quad i = 1, \dots, L, \tag{32}$$

where each function f in the expression corresponds to a layer in the neural network. When we mention the number of layers in a model we are generally referring to the *depth* and when we mention the number of neurons in each layer we are generally referring to the *width.* There are different types of neural networks but the one that is covered in this thesis is a standard *Feed Forward Network*, these types of networks pass data by feeding it through layers of neurons as illustrated in Figure 3.2. One can see that it does not take a large amount of widths and depths before a neural network becomes incomprehensible to the naked eye so the layers between the input f_1 and the output f_L are often hidden from illustrations, this is why they are often known as *hidden layers.*

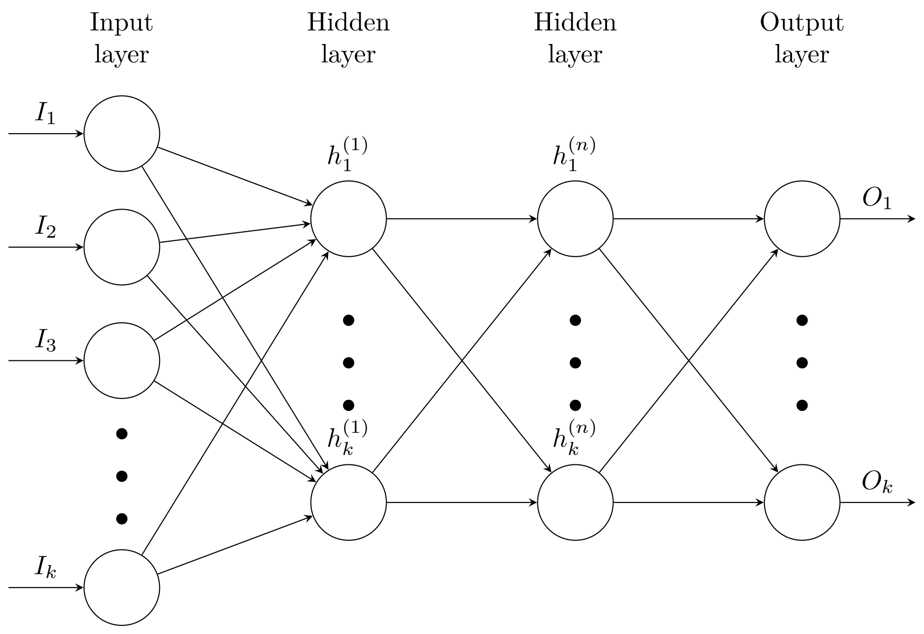


Figure 3.2 - An illustration of a generic Neural Network with n hidden layers and k depth.

Now let us define $x_i = (x_{i1}, \dots, x_{in})$ as the p-dimensional features for the i:th observation. From section 4 in [14] we define the first hidden layer as

$$h^{(1)} = g^{(1)}\left(z^{(1)}\right), \tag{33}$$

$$z^{(1)} = \sum_{k=1}^{n} w_k^{(1)} x^{(1)} + b_k^{(1)}, \tag{34}$$

where $w_k^{(1)}$ are the weights of the first hidden layer, $b_k^{(1)}$ is the bias of the first hidden layer and $g^{(1)}$ is a non-linear function called the activation function with the sole purpose of deciding how much a specific neuron will contribute to the model. There are many types of activation functions available for using, some of them are mentioned in Section 3.2.4. A fairly common one is the sigmoid activation function which is defined as follows:

$$g(x) = \frac{1}{1+e^{-x}}. \tag{35}$$

This activation function will essentially transform the weights of each neuron into values between $[0, 1]$ which in other terms becomes an expression for how much a specific will activate, when close to zero the neuron will have little effect on the prediction and when close to 1 it will have an almost full effect.

But why is the activation function used at all? Imagine for a second that we pass the data through all weights without activating it, this would essentially mean that each time input data is weighted in (34) it simply transforms in a linear manner, this would essentially make the NN behave like a linear regression model. The activation will force the input data to bend into a predetermined non-linear shape (such as a sigmoid) which is why we generally say that NN:s are non-linear. In other words, the activation is what allows the NN to learn from the same complex data patterns that GLM:s often have a hard time to adapt to.

Since a hidden layer is fed its input from the previous layer we then define each layer l as follows.

$$h^{(l)} = g^{(l)}\left(z^{(l)}\right), \tag{36}$$

$$z^{(l)} = \sum_{k=1}^{n} w_k^{(l)} h^{(l-1)} + b_k^{(l)}. \tag{37}$$

3.2.3.2 Estimating Neurons With Back-Propagation.

We have talked about NN:s in a general sense and we have also briefly explained the concept of *Forward Propagation* (FP for short) but what has not yet been addressed is how the model actually learns. While FP is the concept of passing data through the model so that it is weighted into a predictive output, the *Backward Propagation* (BP for short) is the process of passing data backwards in the model so that it can learn from the data. The BP is an optimization method that is based on a *gradient descent* process that minimizes an arbitrary loss function in order to determine how the predictor might be adjusted (as described in Section 3.2.1). As previously mentioned, a common loss function is the MSE or the RMSE, however, this thesis uses the *Poisson deviance* as its loss function (more on the *Poisson deviance* is covered in Section 3.3.1).

The training process is initialized by guessing the values of the networks weights. By propagating data through the model a prediction will be obtained, this prediction might be completely inaccurate in comparison to the observed values but nevertheless we now have an initial guess that can be used in the gradient descent process to optimize the weights and biases. By propagating backwards through the model and nudging each weight in each layer towards a value that minimizes the loss, similar to how it has been done in the previous sections, we end up with a set of neurons that hopefully predicts data accurately. All that we need is a loss function $\mathcal{L}$ that is differentiable with respect to the model parameters, which in this case would be the be the weights that were initially guessed. When repeating this process through a predetermined amount of epochs we essentially fine-tune the weights and biases until they become viable.

Section 6 in [14] shows that what we are essentially seeking is an expression that minimize the loss with respect to the model weights, so if we define the expression $\frac{\mathrm{d}\,\mathcal{L}}{\mathrm{d}\,w}$ we can then expand it with the help of the chain rule and obtain the following expression.

$$\frac{\mathrm{d}\,\mathcal{L}}{\mathrm{d}\,w} = \frac{\mathrm{d}\,\mathcal{L}}{\mathrm{d}\,z^{(l)}} \frac{\mathrm{d}\,z^{(l)}}{\mathrm{d}\,w}, \tag{38}$$

where $z^{(l)}$ corresponds to the non-activated layer l as defined in the previous chapter. Now if we look a bit closer at (38) we can see that the the second term is just an expression for the differentiation of $z^{(l)}$ with respect to the weights and since it corresponds to the output layer it corresponds to the derivative of an arbitrary activation function with respect to the weights, this leaves us with the following expression for the first term in (38).

$$\frac{\mathrm{d}\,\mathcal{L}}{\mathrm{d}\,z^{(l)}} = \frac{\mathrm{d}\,\mathcal{L}}{\mathrm{d}\,h^{(l)}} \frac{\mathrm{d}\,h^{(l)}}{\mathrm{d}\,z^{(l)}}. \tag{39}$$

We need to keep in mind that the BP is the process of propagating the model in the backwards direction which means that each layer is being fed data from the layer in front of it. We rewrite (41), for any given layer l as follows:

$$\frac{\mathrm{d}\,\mathcal{L}}{\mathrm{d}\,z^{(l)}} = \frac{\mathrm{d}\,\mathcal{L}}{\mathrm{d}\,z^{(l+1)}}\frac{\mathrm{d}\,z^{(l+1)}}{\mathrm{d}\,h^{(l)}}\frac{\mathrm{d}\,h^{(l)}}{\mathrm{d}\,z^{(l)}}. \tag{40}$$

Finally we end up with (41):

$$\frac{\mathrm{d}\,\mathcal{L}}{\mathrm{d}\,w^{(l)}} = \frac{\mathrm{d}\,\mathcal{L}}{\mathrm{d}\,z^{(l+1)}}\frac{\mathrm{d}\,z^{(l+1)}}{\mathrm{d}\,h^{(l)}}\frac{\mathrm{d}\,h^{(l)}}{\mathrm{d}\,z^{(l)}}\frac{\mathrm{d}\,z^{(L)}}{\mathrm{d}\,w^{(l)}}. \tag{41}$$

It can be quite easily understood that this expression becomes rather complex as the number of hidden layers increase. Expressing a general NN with three hidden layers would leave us with the following expression.

$$\frac{\mathrm{d}\,\mathcal{L}}{\mathrm{d}\,w} = \frac{\mathrm{d}\,\mathcal{L}}{\mathrm{d}\,z^{(4)}}\frac{\mathrm{d}\,z^{(4)}}{\mathrm{d}\,h^{(4)}}\frac{\mathrm{d}\,h^{(4)}}{\mathrm{d}\,z^{(3)}}\frac{\mathrm{d}\,z^{(3)}}{\mathrm{d}\,h^{(3)}}\frac{\mathrm{d}\,h^{(3)}}{\mathrm{d}\,z^{(2)}}\frac{\mathrm{d}\,z^{(2)}}{\mathrm{d}\,h^{(2)}}\frac{\mathrm{d}\,h^{(2)}}{\mathrm{d}\,z^{(1)}}\frac{\mathrm{d}\,z^{(1)}}{\mathrm{d}\,w}. \tag{42}$$

Now that we have a gradient to descend we simply plug (42) into the gradient descent framework, as demonstrated in Section 3.2.1, where the all the weights and biases in the model will be systematically nudged towards a value that minimizes the loss. More specifically the weights w will be sequentially adjusted according to $w = w - \alpha\frac{\mathrm{d}\,\mathcal{L}}{\mathrm{d}\,w}$ where α is the learning rate of choice. it is hard to say in explicit terms what a right choice of learning rate is since a poor choice can lead to overfitting when α is too small and to underfitting when α is too big, it is often recommended try multiple rates and see which one performs the best for a specific set of data.

3.2.3.3 Combined Actuarial Neural Network (CANN)

The CANN model is an architechture that uses a *Skip-Connection* in order to embed an already existing GLM into the NN:s training process, Section 3.2.5 expands more on the *Skip-Connection.* From section 5.1.4 in [4] we define the CANN model as the following. Consider a GLM with an appropriately chosen link function for the trained Tweedie model, also assume that it has been trained to the point where it is regarded to be an efficient estimator. Now consider a Neural Network with L layers, $W^{(l)}$ and $b^{(l)}$ to be a matrix containing all weights and a vector containing the biases for layer l. If we also define $\hat{y}_i^{GLM}$ to be the linked prediction of a GLM model at a given set of inputs, given this GLM prediction, NN and a suitable activation function $g(\cdot)$ we can define the output layer of the CANN as in (43). It is important to keep in mind that the GLM prediction is identity mapped from the input layer directly to the output layer with a *Skip-Connection* which essentially means that it does not interact with any of the hidden layers, an illustration of the skip-layer in a CANN model can be seen in Section 3.2.5 - Figure 3.3

$$h^{(L)} = h(W^{(L)}h^{(L-1)} + b^{(L)} + \hat{y}_i^{GLM}). \tag{43}$$

The idea of the CANN is to incorporate the GLM into the NN architecture in such a way that the NN is initialized and trained around it without altering the GLM predictions in the training process. in other words, if we were to nullify its weights and biases as in (44),

$$W^{(L)}h^{(L-1)} + b^{(L)} = 0, \tag{44}$$

then we would simply be left with the initial GLM model and the CANN would then produce the same results as the GLM given that the activation function in the final layer is equal to the inverted link of the GLM. The output layer of the CANN would then result as follows:

$$h^{(L)} = g(W^{(L)}h^{(L-1)} + b^{(L)} + \hat{y}_i^{GLM}) = h(\hat{y}_i^{GLM}). \tag{45}$$

Embedding the GLM into the NN architecture in this way will will simply allow the NN to be trained around an already efficient model which in turn means that the a fully trained CANN will try to explain the residuals that the GLM is unable to do on its own. Once again we would like to remind that since the CANN makes use of Tweedie models one also needs to adjust the loss function that is used in the training process. A plain NN often uses the *MSE* whereas the CANN with a nested Tweedie model should instead use the appropriate deviance function for the given model which in our particular case is the *Poisson Deviance*.

3.2.3.4 The link Function in the CANN framework.

Since the foundation of the CANN model is (in this particular case) a GLM we need to address how the link-function fits into the CANN framework. Generally speaking when modelling the claim frequency we work with GLM:s that are linked with the log-link and one can easily wonder how this will fit into the structure of the NN. It turns out that we can define the activation function in the output layer $h^{(L)}$ as the inverse to the log-link without disturbing the internal mechanics of the NN itself. In the previous section we defined the CANN framework as a Neural Network that embeds a static GLM so if we simply expand the log-link to that same framework we can express $h^{(L)}$ as in (47). Bear in mind that the expressions below share the same definitions as in the previous chapters. Also bear in mind that while the activation function of the output layer should be defined in accordance to the GLM link it is not required that all the hidden layers share the same activation.

$$\begin{aligned} y_i^{CANN} &= \exp\{W^L h^{(L-1)} + b^L + < x_i; \beta^{MLE} >\} & (46)\\ &= \exp\{W^L h^{L-1)} + b^L\} \exp\{< x_i; \beta^{MLE} >\}, & (47)\end{aligned}$$

where $< x_i; \beta^{MLE} >$ are the log-linked results from the GLM for some feature combination x_i. The results in (47) essentially gives us an expression where the the two models have their own weight to the final prediction. It goes without saying that if the GLM is a strong predictor then the NN-part of the model will have a more neutral value since the BP process already has a strong foundation to start with. In other words we can express y_i^{CANN} as follows:

$$\begin{aligned} y_i^{CANN} &= \exp\{z^{(L)}\} \cdot \exp\{\hat{y}_i^{GLM}\} & (48)\\ &= y_i^{NN} y_i^{GLM}. & (49)\end{aligned}$$

3.2.4 Activation Functions

The activation function is generally speaking the key ingredient in the neural network that makes the NN non-linear and allows it to be a versatile tool that can be applied for a wide variety of datasets. Since it forces the passing data to adapt to a predetermined function the NN will then have a better chance to fit irregular data patterns. There are various functions that can be used to activate neurons, 3.2 lists some of the more popular functions that are often used for practical as well as educational purposes.

Activation Function Name	Formula
Rectified Linear Unit	$f(x) = max(0, x)$
Hypoerbolic Tangent	$f(x) = \frac{e^x - x^{-x}}{e^x + e^{-x}}$
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$
ArcTan	$f(x) = tan^{-1}(x)$

Table 3.2 - a summary of popular activation functions

3.2.5 Skip-Connection

A *Skip-Connection* is any type of connection in the NN that allows for the data to break natural path and skip specific layers such that the input from one layer is the output from layer $l-n$ where $l-1 \neq n$. In a plain and unaltered neural network the input from layer l is defined as the output from layer $l-1$ while the *Skip-Connection* is fed from a layer of choice. Some sources, such as [8], even shows that skipping layers can prove to have a an advantage for modelling certain types of data such as images for image recognition but this thesis will not cover such cases.

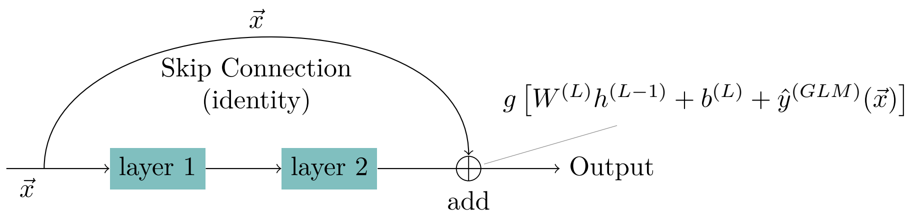

$\vec{x}$
Skip Connection
(identity)
$g\left[W^{(L)}h^{(L-1)} + b^{(L)} + \hat{y}^{(GLM)}(\vec{x})\right]$
layer 1
layer 2
Output
$\vec{x}$
add

Figure 3.3 - An illustration of the skip-connection in a two-layered CANN.

3.3 Model Comparison

3.3.1 Root Mean Squared Error & Poisson Deviance

The mean squared error is a measure that summarizes the prediction error for a model. We define the MSE as the following expression,

$$MSE(y, \hat{y}) = \frac{1}{n} \sum_{i}^{n} (y_i - \hat{y}_i)^2. \tag{50}$$

One can easily see why using a measure like the MSE is advantageous in comparison to general goodness-of-fit tests. The MSE is generally speaking very versatile and works with practically every type of model regardless of how advanced it is. The strength in the MSE lies in its simplicity since all that it really does is to calculate the average difference between observations and predictions without any requirements or assumptions other than simply having a trained model and available for testing. By square-rooting the expression in (50) we then get the RMSE.

$$RMSE(y, \hat{y}) = \sqrt{\frac{1}{n} \sum_{i}^{n} (y_i - \hat{y}_i)^2}. \tag{51}$$

The poisson deviance has been lightly mentioned in previous sections and in this thesis it might be more appropriate to use a function that takes account for certain weights in the data. We define the poisson deviance directly from Section 3.1 from [1] as the following expression.

$$D(y, \hat{\mu}) = 2 \sum_{i} w_i \left[y_i \log\left(\frac{y_i}{\hat{\mu}_i}\right) - y_i - \hat{\mu}_i \right]. \tag{52}$$

3.3.2 K-Fold Cross Validation

Cross-Validation is a model validation framework which is commonly used to assess the predictive strength that a model might posses. While a model that poorly fits the data also often lack predictive power it may still be the case that it predicts data in the near vicinity of a desired result simply by sheer luck. The reason to this can be one of many such as overfitting, biased training-data, poor data samples and a multitude of other reasons. By *Cross-Validation* however, we aim to get a wider look of how a model performs across a an entire dataset, we do so by subsetting the data into smaller sets and using some of the subsets as training data and other parts as validating data. This simple method of subsetting allows the model in question to be trained on some data while simultaneously having no bias for the the remaining data.

There are many forms of cross validation but the one used in this thesis is called *K-fold cross validation*. The idea is to subset the data into K subsets (or folds) and then using $K-1$ subsets as training-data while the remaining subset is used as test data. This is done by iterating through all subsets such that each subset gets to act as a test-set while the rest of the subsets act as training-sets (as illustrated in figure 3.4), for each iteration the *RMSE* is calculated and its final average will then be an indicator of how well a model perform on the data as a whole. It is also worth mentioning that one should also keep an eye out for subsets with deviating *RMSE:s* since that could be a segment worth of further investigation.

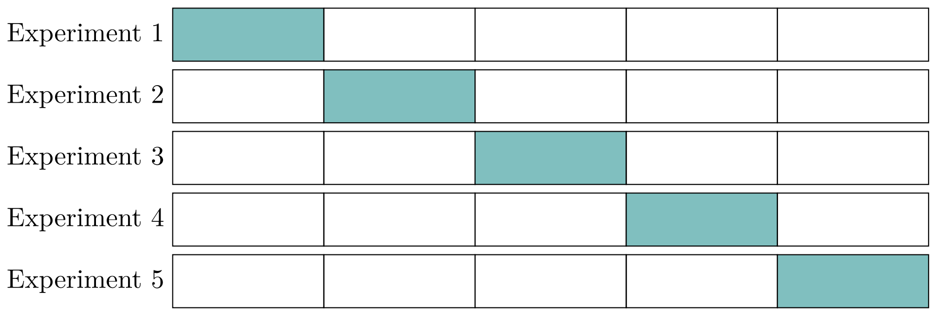


Figure 3.4 - An illustration of a K-fold cross validation process

Figure 3.2 illustrates the cross validation process where the dataset has been split into 5 subsets. Here we can see in a more illustrative manner how each iteration is allowed to act as testing data. This method might seem simplistic but it is widely used.

We define the K-Fold Cross Validation process in Algorithm 3.

```
Step 1. Split the data $\mathcal{D}$ into $K$ subsets.
Step 2.
foreach subset $S_i \in \mathcal{D}$ do
    1. Create a dataset $\mathcal{D}^*$ that holds out $S_i$.
    2. Train a temporary model $\mathcal{M}^*$ on $\mathcal{D}^*$.
    3. Predict the data in $S_i$ using model $\mathcal{M}^*$ and calculate the RMSE, save the results.
end
4. Calculate the average of all collected RMSE:s.
```

Algorithm 3: K-Fold Cross Validation

4 Data

4.1 Factor Levels & Data Summary

The data for this thesis is based on a French motor insurance portfolio which is publicly provided by *CASDatasets* [10], although it is sourced from an unknown insurer it is generally presumed that the data comes from a real insurance company. The chosen data material is often used as an introductory material for insurance pricing in various proof of concepts or training/educational sessions. Often when studying how models compare to each other it can be advantageous to simulate data from a proper distribution since simulated data will fulfil the theoretical assumptions of the chosen distribution, often much better than data that is collected from real-world observations since non-synthetic data tend to contain more noise. However, since this thesis compare model performances for practical purposes it seems more suitable to use a data source that stems from a real insurance portfolio, this way it likely contains many of the oddities that might be observed in reality and the process of fitting the models might be more true to how it happens in real model training scenarios.

The data is spread on 9 categories as shown in table 4.1.

Variable	Variable Explanation	No. Levels
VehPower	Vehicle horsepower.	12
VehAge	Age of the vehicle.	21
DrivAge	Age of the driver.	73
BonusMalus	Bonus Malus.	93
VehBrand	Vehicle brand.	11
VehGas	Type of fuel that the vehicle runs on.	2
Area	Regional inhabitant density.	6
Density	No. inhabitants per km^2 in the registered city.	567
Region	Policy Region.	22

Table 4.1 - Metadata

The Regional inhabitant density is categorically defined from A to F where F is defined as rural areas and A as Dense areas. Vehicle Brands are defined as the following.

- A: Renault, Nissan and Citroën.
- B: Volkswagen, Audi, Skoda and Seat.
- C: Opel, General Motors and Ford.
- D: Fiat.
- E: Mercedes Chrysler and BMW.
- F: Japanese (except Nissan) and Korean.
- G: Other.

It would have been interesting to model each brand for itself but unfortunately the data offers no such possibility. The Regions are defined according to the French regional classification from 1970-2015, While the documentation in [10] does not offer any clarifying labels to the regions it is a reasonable assumption that it follows the standard notation from the French *National Institute of Statistics and Economic studies*. However, since labelling the regions does not make any difference to the outcome of the analysis we will use the data as it comes without manually relabelling it.

4.2 Data Transformations

General Adjustments

Since the data is publicly available it is therefore also subject to scrutiny from whoever that uses it. Some key points that have been publicly pointed out are listed in the following list, some alterations in the data has been made since there is some validity in these points.

- Some observations with different claim numbers has a high likelihood of belonging to the same claim.
- Some observations have claims registered without a respective claim amount, also known as *null claims*.
- Some policies might have a large number of claims and are likely to be erroneous, see figure 4.1.

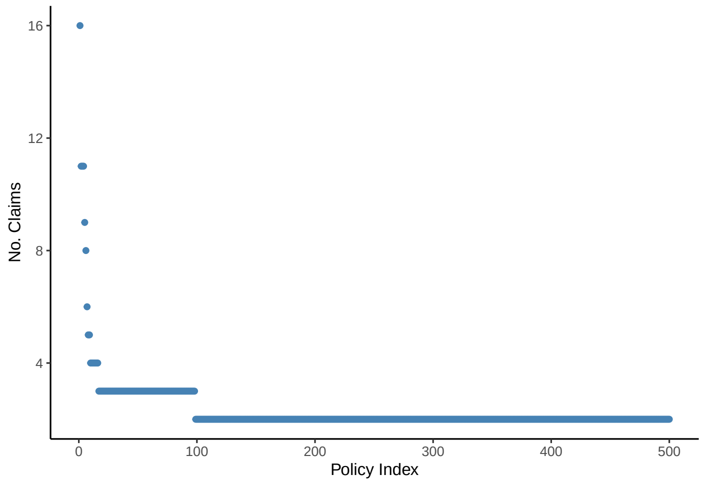


Figure 4.1 - 500 largest observations by the number of claims per policy number in descending order

Transformation 1 - No. Claims

Due to the previously made points all claims are truncated to a maximum of 5 claims, although nothing says that individual policies with 16 unique claims is a theoretical impossibility, data in general suggests that it is unlikely. For this reason such observations will be regarded as errors rather than actual observations. It should have little to no effect in the bigger picture since the size of the remaining data would likely water out whatever effect these errors add. We assign the number of claims as follows:

$$ClaimNb = \begin{cases} ClaimNb & \text{, if } ClaimNb \leq 5 \\ 5 & \text{, otherwise} \end{cases}$$

Transformation 2 - Drivers Age (DrivAge)

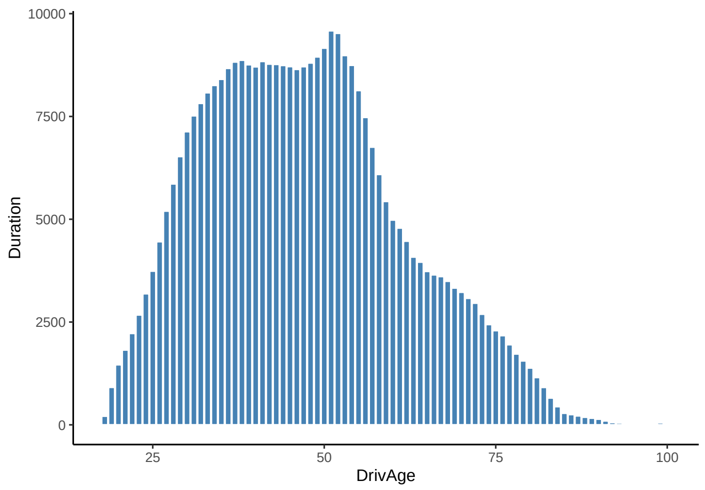


Figure 4.2 - Number of insurance years per Drivers Age

Data quality seems to be fairly satisfactory up until about the ages of about 80-90 due to natural reasons, to avoid high volatility for the higher ages the DrivAge factor will be truncated to maximum of the age 90. One could possibly group some levels together to make handling the model easier but there seems to be no clear reason to do so, transforming data by grouping can arguably lead distortions of small variations that might be of importance. We assign the drivers age as follows:

$$DrivAge = \begin{cases} DrivAge & \text{, if } DrivAge \leq 90 \\ 90 & \text{, otherwise} \end{cases}$$

Transformation 3 - Vehicle Age (VehAge)

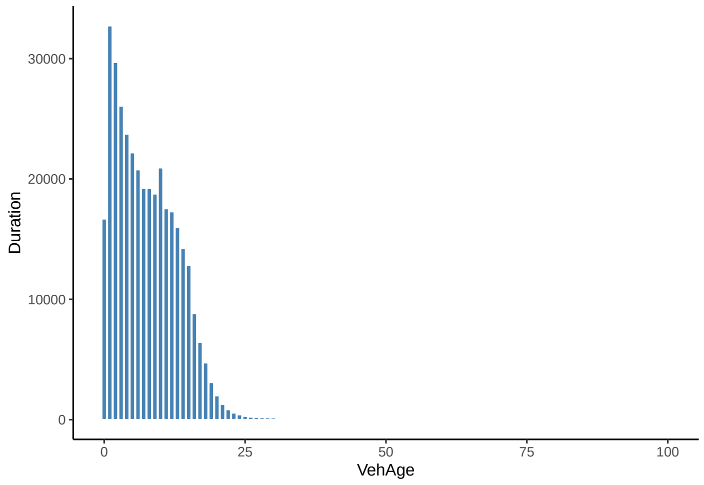


Figure 4.3 - Number of insurance years per Vehicle Age

Data quality seems to be satisfactory for all levels below 20-25 so similarly to the drivers age the vehicle age is truncated to a maximum age of 20. We assign the vehicle age as follows:

$$VehAge = \begin{cases} VehAge & \text{, if } VehAge \leq 20 \\ 20 & \text{, otherwise} \end{cases}$$

Transformation 4 - Bonus Malus

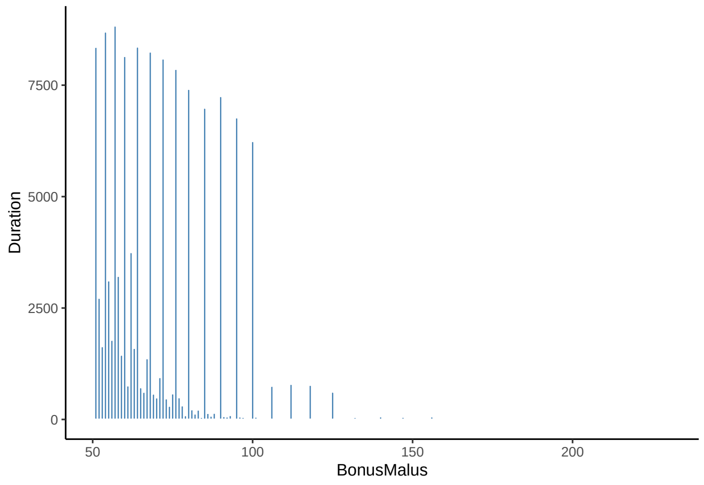


Figure 4.4 - Number of insurance years per Bonus Malus level

The Bonus Malus covariate spans between $50-350$ where values under 100 is considered to be bonus and values above 100 is malus, in other words observations where $BonusMalus > 100$ tend to contribute more to the portfolio risk than the observations with $BonusMalus <= 100$. The vast majority of observations are below 100 as it is often expected in a healthy insurance portfolio. We assign the Bonus Malus as follows:

$$BonusMalus = \begin{cases} BonusMalus & , \text{ if } BonusMalus \leq 150 \\ 150 & , \text{ otherwise} \end{cases}$$

4.3 Skewed Data

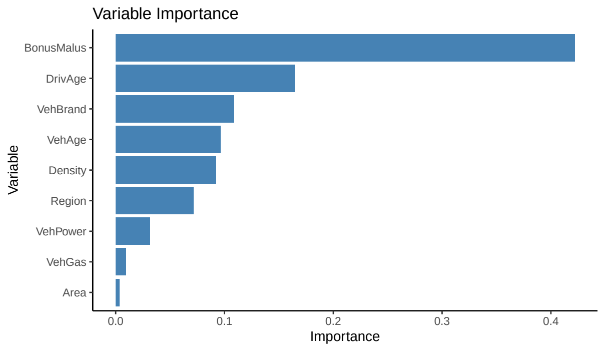
It has been mentioned in some of the previous sections that insurance data tends to be skewed towards a low number of claims, this meaning that the absolute majority of policies in the data will not have any associated claims, as shown in Figure 4.1. The ratio between the number of policies with claims vs without claims can vary depending on the type of asset that is insured, in this particular case we have that roughly 90% of the policies are without claims. This is not necessarily a problem but it is important to recognize that individual predictions are unlikely to be satisfyingly close to zero which can lead to somewhat confusing results when comparing prediction errors which is why this thesis makes use of the *Poisson Deviance* instead of the *MSE*. The quality of both the fit and the performance on test data will be judged with numerical quantities and also visually through graphs that illustrates the average claim frequency per covariate as the visual perspective is often a good complement to the deviances.

5 Model Fitting & Parameter specifications

5.1 Variable selection

As the goal is to compare three different models where two of the models are of the Machine-Learning kind one might notice that goodness-of-fit tests that are designed for GLM:s might not be a suitable option. One reason being that tests such as the LRT often makes use of the model parameters which does not translate well for Machine-Learning models due to their different parameter architecture. Since this thesis aims to answer which type of model that performs best in practical applications we find it appropriate to start the modelling process with training all models on the same covariates while simultaneously allowing for various transformations that would keep the model integrity, examples of such transformations could be *feature smoothing functions*. The idea behind this reasoning is that models with different covariates have a likelihood to perform differently because they are trained differently, one might observe that a model performs better in comparison to another and believe that it is due to the model architecture being more powerful, in reality it might instead be a matter of one model simply being trained on better features than the other one. It is simultaneously important to not restrict GLM:s with a set of too strict rules since they often need to transform covariates to better fit non-linear data. Machine-Learning models have a natural tendency to adapt to complex data-structures so restricting the GLM:s too much might lead to biased results.

The variable selection is done by fitting a general GBM model using all covariates and then excluding the ones that make small contribution to the model according to the importance factor. By observing the left graph in figure 5.1 one can see that practically all features except for the *Area* Variable has a noticeable contribution to the model, therefore the only covariate that will be excluded from the analysis is the Area classifier. The right graph in figure 5.1 illustrates the correlation effects between all covariates where each square will grow larger and have a less opaque colour as the correlation coefficient between two variables gets closer to 1 or -1. Two covariate pairs seems to be standing out with correlation coefficients being at about 40%, the pairs that are being referred to are *VehBrand:VehAge* and *BonusMalus:DrivAge* but through further investigation, the GLM model seems to perform better wihtout discarding any of the three variables while including interaction effects only for *VehBrand:DrivAge* in the model. More details regarding the interaction effects can be found in Section 5.2.



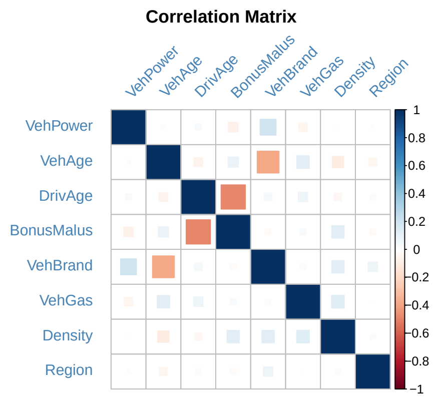


Figure 5.1 - Plots of variable importance and feature correlations.

5.2 GLM

The GLM is in some sense the trickiest model in the comparison since the predictive quality often depends on how the analyst chooses to transform the model variates and the underlying data. While the fit of Machine-Learning models usually differ by how the complexity is specified (depth, width etc.), a GLM has a wide variety tools that can be used to improve its fit, the main tool used in this thesis is *Smoothing Splines*. The *Smoothing Splines* are often used in *Generalized Additive Models* where the goal is to fit polynomials one piece at a time which should lead to the parameters for the covariate being generated from a continuous function of a desired degree (see Section 5 in [1]). Another option (that also is commonly used) is to group various levels that seem to have similar levels of risk into one level. The downside of grouping data however, is that it might be a tedious (and often impractical) task to identify levels with comparable risk when the number of unique levels are many. This option can also lead to sudden jumps in the premium if the data suggests any spikes in the claim frequency, a good example where this often happens is for age-related covariates. The details on how *Smoothing Splines* are fitted through the GAM framework and how they coincide with the GLM framework are referred to section 5 in [1].

Upon fitting an initial model without any transformations it was noticed that the model is somewhat struggling to fit the drivers age, see Figure 5.2. There could be many explanations to why a non-smoothed variable would have a hard time fitting the drivers age but it often boils down to a lack of data for all levels. Figure 5.2 illustrates how the fit changes when the degree of a smoothing function is changed and we can also observe that the initial model (model 1) has a somewhat good fit but is still missing the slopes that are observed in for all groups of ages, adding a polynomial transformation clearly improves the fit can be observed that the model makes better assessment as the degrees increase. Manual experimentation concludes that a 4th degree polynomial gives the best fit while simultaneously minimizing the risk for overfitting.

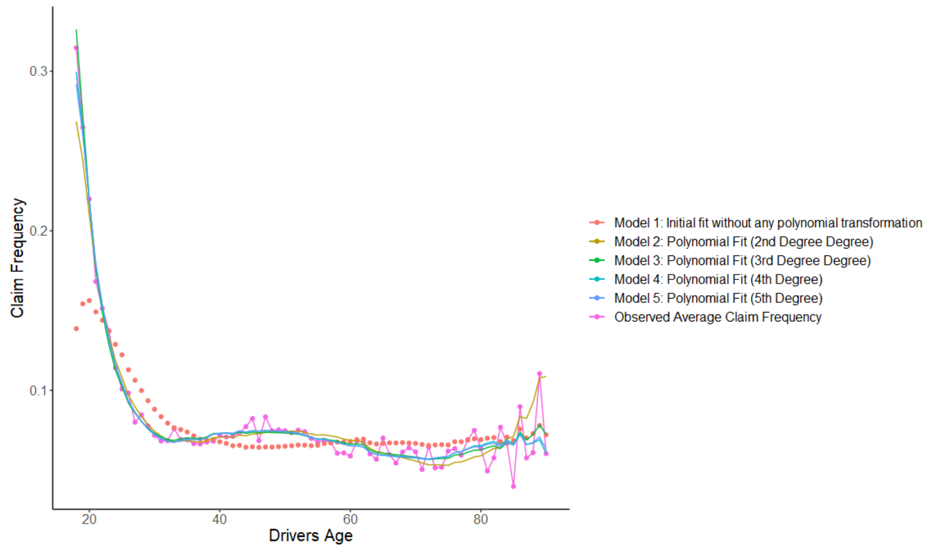


Figure 5.2 - Average fitted claim frequencies per divers age.

Further testing shows that there is some room for improvement by the help of interaction effects, this has likely to do with the correlations that was observed in section 5.1 but what is interesting is that although the improvements in deviance can be observed, both for the training data and testing data, the LRT does not find it necessary to keep both suggested interaction effects in the model despite a slight improvement in deviance. After weighing whether to ignore the results of the LRT or not it was concluded to only keep interactions for the Vehicle Ages and Vehicle Brands in the model. Although 5.1 shows that model 4, a model containing interaction effects for both covariate pairs, performs slightly better than Model 5 we can also observe that the AIC values are almost the same for both models, so in order to try and keep the number of parameters to a minimum it seems like the cost of loosing the interactions for Bonus Malus and the Drivers Age is very small.

The model structures are in Table 5.2 are defined in Table are defined in 5.1.

Model Version	Area	VehPower	VehAge	BonusMalus	VehBrand	VehGas	Density	Region	DrivAge	VehBrand:VehAge	BonusMalus:DrivAge
Model 1	X	X	X	X	X	X	X	X	X		
Model 2		X	X	X	X	X	X	X	X		
Model 3		X	X	X	X	X	X	X	X		
Model 4		X	X	X	X	X	X	X	X	X	X
Model 5		X	X	X	X	X	X	X	X		X
Model 6		X	X	X	X	X	X	X	X	X	

Table 5.1 - Model structures

Table 1: Cross Validation and AIC

Model	AIC	Poisson Deviance - Train Data	Poisson Deviance - Test Data
Model 1	171658.6	0.2415443	0.2423703
Model 2	171661.3	0.2415530	0.2423564
Model 3	171767.2	0.2408120	0.2416384
Model 4	171230.1	0.2406988	0.2415473
Model 5	171230.0	0.2407024	0.2415549
Model 6	171269.5	0.2408084	0.2416300

Table 2: LRT

Model Comparison	P-Value (Chi-Squared)
Model 1 Vs Model 2	0.02970869
Model 2 Vs Model 3	p<0.0000001
Model 3 Vs Model 4	p<0.0000001
Model 3 Vs Model 5	p<0.0000001
Model 3 Vs Model 6	0.03359628

Table 5.2 - Results from statistical testing.

The GLM is fitted using a linear Poisson model with the log-duration as an offset, some experiments were made with using $Tweedie(p)$ models with a variance power to be defined somewhere near 1 but no statistical advantage could be found to justify its usage. $Tweedie(p)$ models are often more properly used when modelling the risk-premium directly and one could even consider it to be quite rare that such a model would perform better in predicting claim frequencies than a model that assumes the Poisson distribution but depending on how the data is distributed it might be worth the while to just compare the two options. The $Tweedie(p)$ model will converge towards a Poisson model as p moves towards 1 as shown in section 3 so merely trying to train a model with a variance parameter close to 1 might not be a farfetched idea. To better visualize how the Tweedie distribution varies when p is altered one can observe Figure 5.2, it illustrates simulated densities from various $Tweedie(p)$ distributions in comparison to a Poisson distribution. It is noteworthy that the Poisson distribution is discrete and does not have a formally defined density, Figure 5.2 merely illustrates the results of a smoothed function of a histogram, this option is chosen because it is more visually forgiving than histograms when comparing multiple samples of this kind.

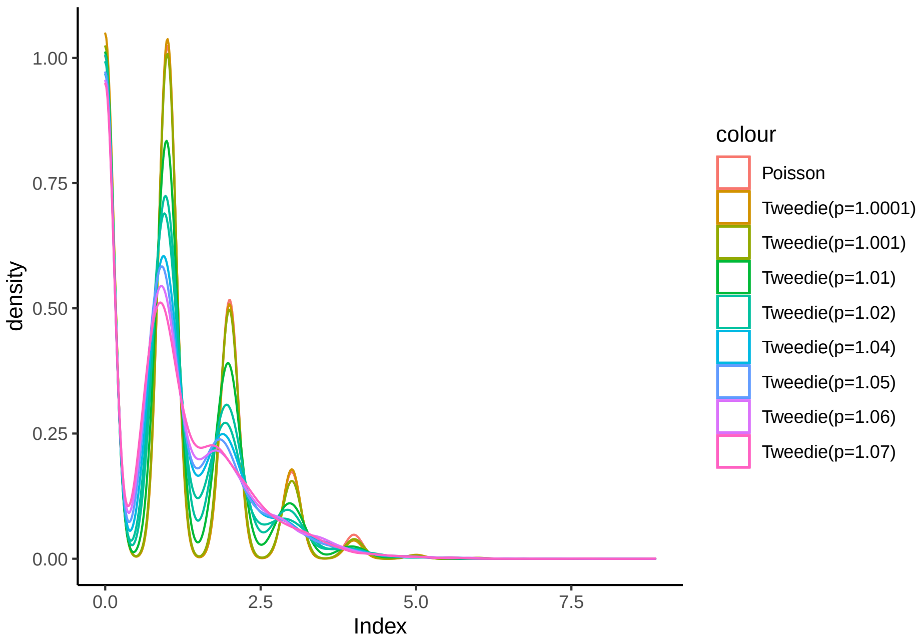


Figure 5.2 - Density plot of simulated Tweedie data for $1.0001 \leq p \leq 1.07$ *with expected values around 1.*

5.3 GBM

The GBM parameters are specified by iterating over all unique combinations of a predefined set of depths, rounds, maximum tree sizes and learning rates (η). It is important to make granular changes to the parameters for each iteration so that the results of as many perspectives as possible are gathered, roughly 9000 models has been trained during the tuning process. For each trained model a deviance is calculated and stored for later evaluation, once the process is finished it only remains to select the configuration which allows for the minimal deviance. Figure 5.3 illustrates the prediction error from each model and what we can see is that most configurations does not make a large difference to improve the performance, in other words it could be beneficial from both a computing and an interpretive perspective to select the model among the best ones that has the least level of complexity.

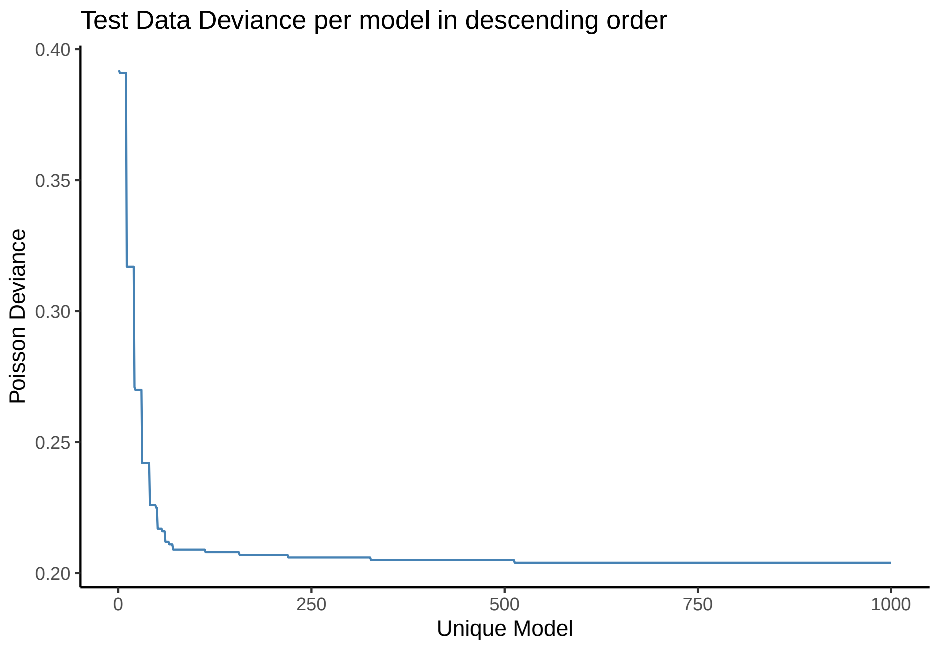


Figure 5.3 - Rounded prediction errors for testdata per unique model in a descending order (Truncated to 1000 first models)

Some models seem to perform identically to others but it is noteworthy that the deviance values are rounded to 6 decimals during the tuning process which leads to models that perform very close to each other will have a high probability to be rounded into the same value. In the end the following configuration produced the best fit at the lowest complexity cost.

- Rounds: 58.
- Depth: 6.
- Learning Rate (η): 0.3 (Default).

5.4 Neural Network

Similar to GBM:s, NN:s are also black-box models that leave the details on how they transform data hidden behind the curtains. Deciding the configuration of a neural network can often be a bit tricky since the line between overfitting and viable predictions can sometimes be a fine one. Similar to the model selection process of the GBM, a matrix containing all unique combinations of a predefined set of depths, widths and learning rates was generated in order to test which of these combinations would produce the best prediction in a cross-validation process. Figure 5.4 illustrates the *Poisson Deviance* per unique model in a descending order.

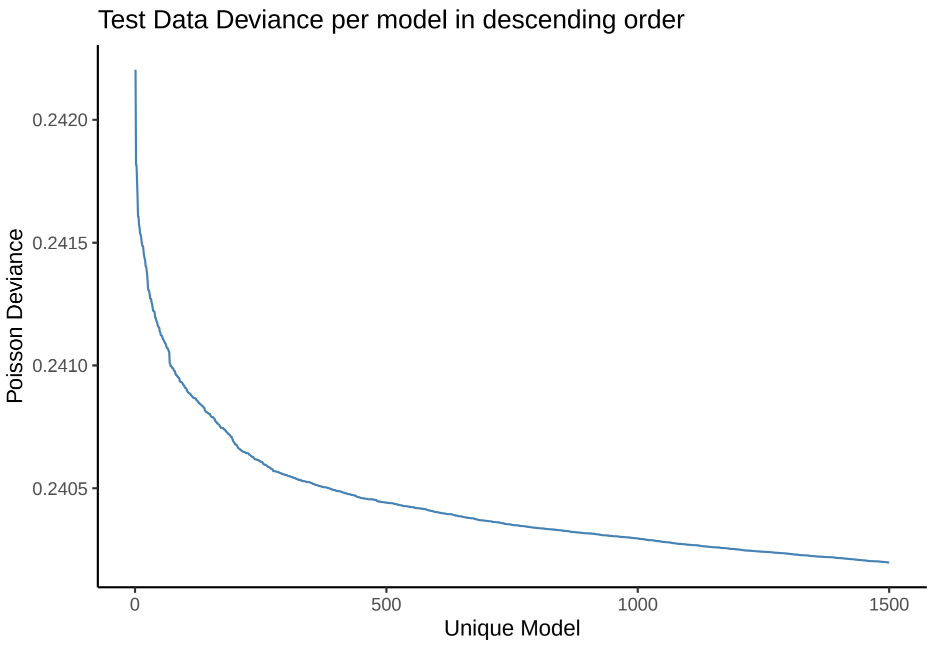


Figure 5.4 - Prediction error on test data per unique model in a descending order (Truncated to 1500 first models)

It might seem like the model can be improved a bit more due to the slope of the deviance curve but one should bear in mind that the models would improve less than 1/1000 per iteration once a level of complexity was reached. In the end it was decided to train the network on the following parameters.

- Input Layer Depth: 5.
- Number of hidden layers: 3.
- Hidden Layer Depth: [29,26,23].
- Output Layer Depth: 1.

Since NN:s makes use of activation functions we have to take account for two things.

1. Activation functions does not have the ability to process categorical variables.
2. Neural networks are scale-sensitive to the data that is input for the activation.

The first note has already been taken care of since all variables has been converted into numerical values in prior to modelling but when it comes to the magnitude of all feature levels the scale can actually pose a problem. If some features contain levels that are significantly larger in scale than the other levels then the feature with the larger scale will tend to be more dominant which in reality can cause the NN to predict less accurately than if the variable has been properly scaled. Another issue that can disturb the efficiency of the analysis is that larger numbers take up more space in a computers internal memory which in turn can have a significant effect on the processing speed, although this issue does not cause any deviations to the training outcome it can still be advantageous to minimize the computational effort when comparing a large amount of models, NN:s are often trained on large datasets so saving processing time whenever it is possible can be in the best interest of the analyst.

The scaling problem is solved by normalizing the features available in the model. When normalizing the features we are simply re-scaling all values so that the visual representation of each level is altered while still keeping the relevant uniqueness of each level. There are quite a few ways to scale features, in this thesis we do it as follows.

Each feature x is transformed through the following procedure.

$$x_i = \frac{x - x_{min}}{x_{max} - x_{min}}(u - l) + l, \tag{53}$$

where x_i is an observation in a given feature and u and l represents the upper and lower limit of the new transformation, the scale transformation are usually appropriately chosen to match the requirements of the activation function, since we are fitting the neural network with a tanh-activation we specify that $l = -1$ and $u = 1$. Once scaling of this sort has been applied all levels in each feature will have a unique definition so that $x_i \in [-1, 1]$.

5.5 Hardware & Software specifications

Several programming languages and packages has been used to fit the mentioned models where R has been the main software that ties the bag. Initially the NN:s were programmed in a separate Python environment but since this thesis is written in RMarkdown it became clear quite fast that jumping between environments would be highly impractical. The NN:s were instead designed and trained in R with the help of the Keras API which acts as a bridge between the R environment and the Google-developed Tensorflow framework. Some R-native packages that support NN:s were found but they seemed somewhat impractical to use in this thesis since the CANN framework requires some level of freedom when it comes to designing a model, this is where Tensorflow could supply the demand. The GLM and GBM frameworks have rich sets of packages available in R that require no third party application so they were trained using standard R libraries that are well-documented. Table 5.3 lists all packages used in the training process. Although R is not the fastest programming language available it is likely that a significant amount of time would be spent on formatting and migrating data between environments during the testing phase if the NN:s were chosen to be trained in Python. The key advantage in running everything in R is that R provides built-in prediction functions which will allow for a smoother testing experience regardless of what model we are evaluating.

During the tuning process a total of about 9000 GBM:s and about 18000 neural networks has been trained and evaluated using an Intel Core i7-6700K CPU, 32 gb high-speed RAM and a NVIDIA GeForce 960 GPU with 4GB RAM. The computer-interested reader might notice that the equipment is somewhat old in comparison to modern standards, surely some processing time would have been saved with more powerful hardware but this setup seemed to do the job just fine.

Packages	Software Environment	Usage
dplyr	R	Data manipulation.
splitTools	R	Data partitioning.
stat	R	Fitting the GLM (Default Package).
XGBoost	R	Fitting the GBM (Default Package).
Keras	Python	Training the NN via Tensorflow.
Python	Python	Main language used by Keras.

Table 5.3 - Summary of softwares used in the training process

6 Results

6.1 Summarized results for all models

It is worth mentioning, before the results are presented, that the iterations in Table 6.1 corresponds to the number of iterations made in the gradient descent process which is not used by the GLM. The parameters corresponds to the number of unique regression parameters for the GLM:s, the number of trained neurons for the NN and the number terminal nodes for the GBM:s final tree.

It is noticeable in Table 6.1 that both Machine-Learning models makes quite a small impact in comparison to the GLM framework when judging by the train- and test deviance. A fair question would be why anyone should bother training a model with the complexity and time consumption of a Machine-Learning model if the yield is not more visually impressive than this. Although the question is fair, it is important to acknowledge that the GLM framework has proven itself to be a good framework for linear data and this particular case is no exception. Perhaps the question needed to be asked instead is how much of an improvement there is room for. The CANN will, by design, try to fill in the GLM:s residuals and when the GLM performs strongly the neural network will naturally have a lesser impact and perhaps this should be what is expected of it. Rather than hoping for the NN architecture to make GLM:s obsolete one should maybe see it as a mechanism for boosting an already reasonable fit.

Model	Iterations	Parameters	Train Deviance	Test Deviance
Model 1: GLM - Without Variable Smoothing	NA	42	0.2408	0.2416
Model 2: GLM2 - With Variable Smoothing	NA	46	0.2408	0.2416
Model 3: Standard Neural Network	300	60	0.2387	0.2406
Model 4: CANN	300	78	0.2372	0.2402
Model 5: GBM	58	63	0.2336	0.2402

Table 6.1 - Summary statistics of all models in action.

The effect of the CANN can be observed more clearly by intentionally worsening the fit of its underlying GLM. We do this by removing one or more explanatory variables from the model and then re-training the CANN using the parameters from the new GLM. Judging by the results in 6.2 it is quite clear that the CANN can be a great tool to use hand-in-hand with the GLM. In this particular case the underlying GLM is trained using only the feature for the vehicle fuel type and despite the lack of all other variables, the CANN is able to produce results that are similar to Model 3 in Table 6.1, even slightly better. It is as if the well-fitted GLM constrains the NN:s training process and and is thus hindering it from making finer improvements. This is not completely illogical since the more fixed parameters you add to a CANN the less freedom it is given to learn from the data. However, by looking at the results from the cross-validation process in Table 6.1 we can state that the plain NN is able to further explain the residuals of the GLM to the point where it is a viable option.

Model	Parameters	Train Deviance	Test Deviance
Model 1: GLM - Without VehGas	2	0.2519	0.2532
Model 2: CANN2 - Without VehGas	78	0.2378	0.2401

Table 6.2 - Summary statistics of an intentionally worsened GLM and a CANN trained with it.

6.2 GLM Results

The GLM framework has proved itself to be a good predictor for this particular dataset as seen in Table 6.1 and when plotting the model in the style of tariff-summary, a common practice in the insurance industry, we get a visual representation for how the GLM model is able to perform over its covariates. When putting the results into view (Figure 6.1) we can see that there is some room for improvement as some levels for the vehicle brands and vehicle powers are a bit off from the observed average, although perhaps not by a great factor. The model seems to be having a hard time with predicting feature levels where data becomes a scarce resource but even given the scarcity situation we observe that the GLM still is able to produce results that is not entirely off the rails. As previously mentioned the fit can be improved by using continuous functions in the fitting process but we see no clear need to add such transformations besides from the drivers age.

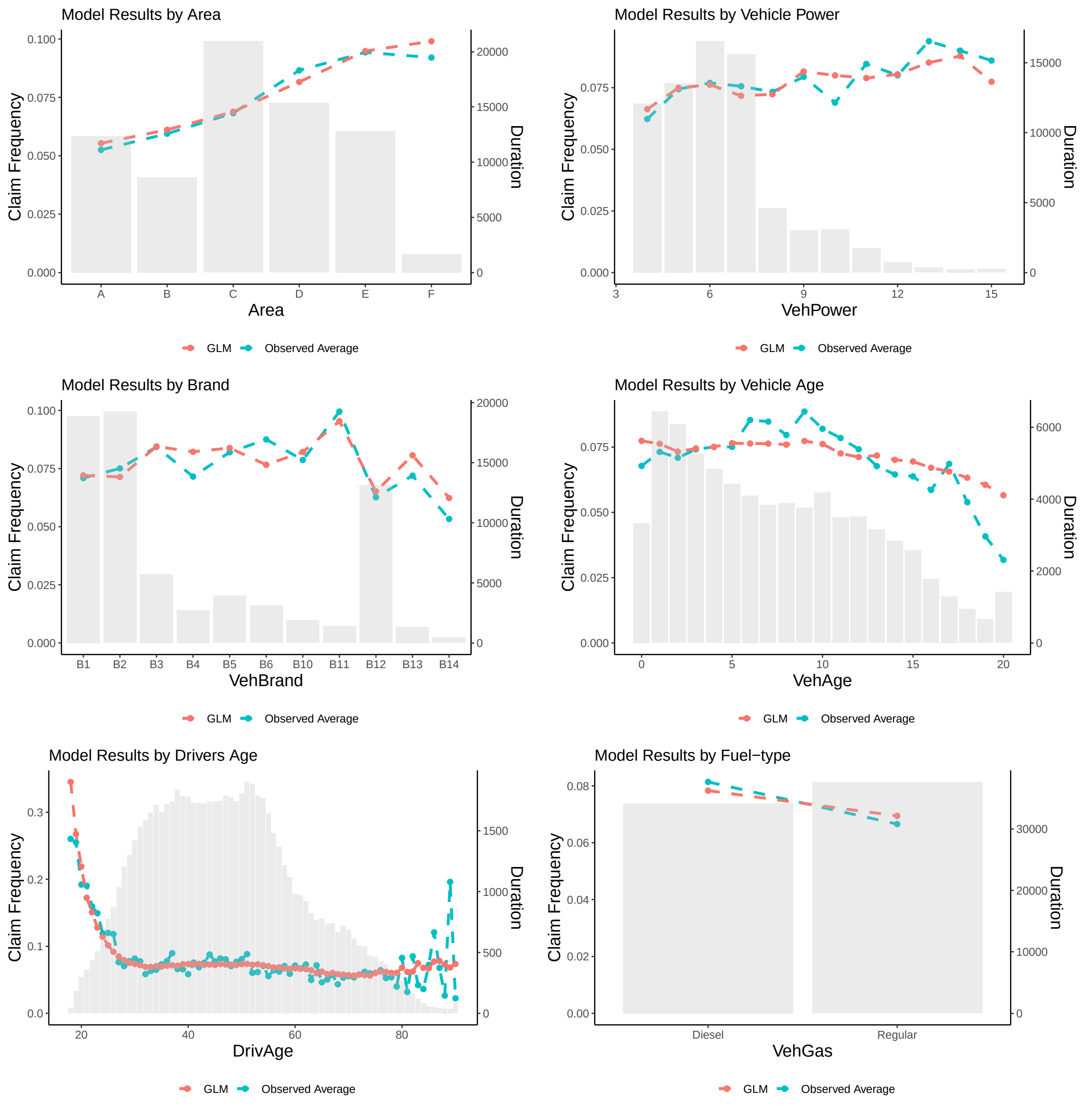


Figure 6.1 - Grouped plots for GLM results.

6.3 GBM Results

Figure 6.2 illustrates how the deviance evolve as the number of iterations in the *Gradient Descent* increases and it can be clearly seen that more iterations stagnates the improvement on the test deviance. Eventually, adding more iterations to the fitting process will only improve the fit on training data while doing little for the predictions on test data, and since the GBM is an additive process it can continue iterating in all perpetuity unless it is told to stop, this leaves the GBM in a position where it is incredibly easy to overtfit if one is not careful.

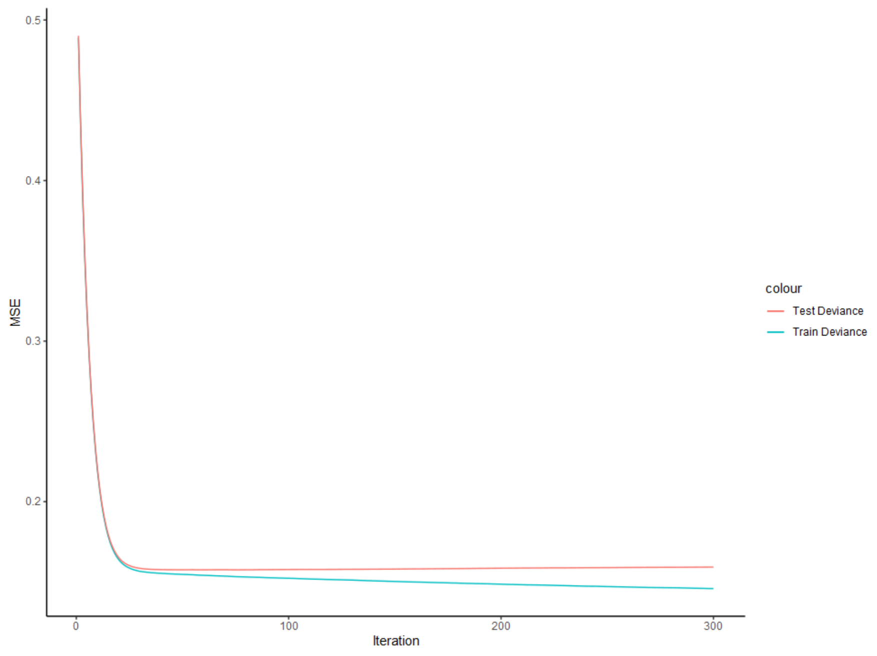


Figure 6.2 - Evaluation log from the XGBoost process.

In table 6.1 we saw that the GBM produced a slightly better model than the GLM with respect to the *Poisson Deviance* for both training data and test data. However, by observing Figure 6.3 it is not quite clear where this small improvement have been applied since the GBM seem to perform worse than the GLM for some of the feature levels. Observing graph 6 in Figure 6.3 we can see that diesel-powered vehicles performs almost perfectly while gasoline-powered vehicles performs worse for the GBM than for the GLM. It is as if the GBM simply has raised the intercept of the GLM which can leave a person wondering if there is a deeper reason to why the diesel-powered vehicles perform better in GBM:s.

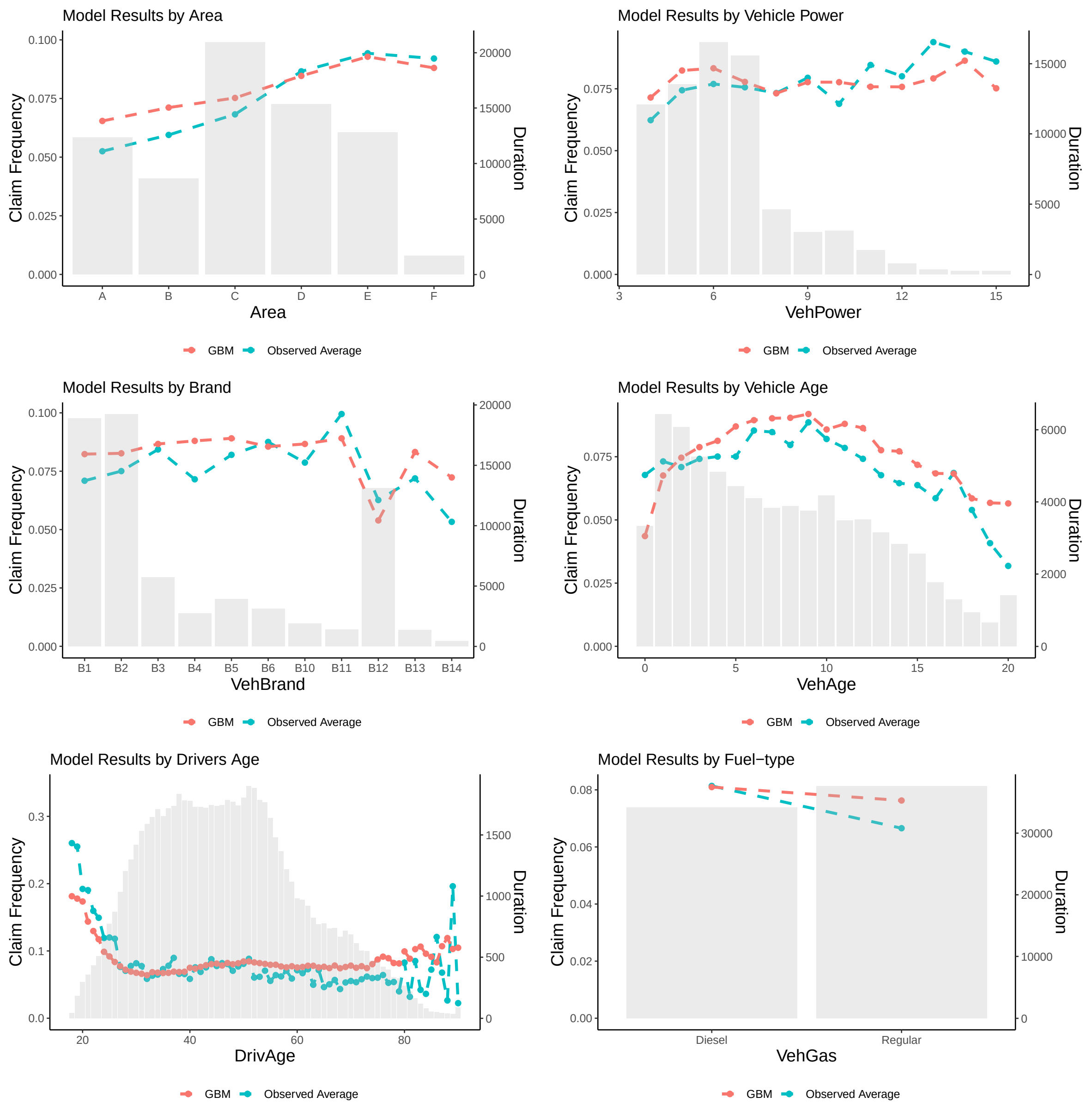


Figure 6.3 - Grouped plots for GBM results.

6.4 CANN Results

As the CANN is tuned we observe in Figure 6.4 that the *Back-Propagation* process decreases the loss as the epochs increase where it starts to converge after descending to a deviance of about 0.1615. However, it might be noticed that the road from the maximum to the minimum is not a very dramatic descent which is largely explained by the fact that the embedded model has already shown to be a viable predictor in Figure 6.1.

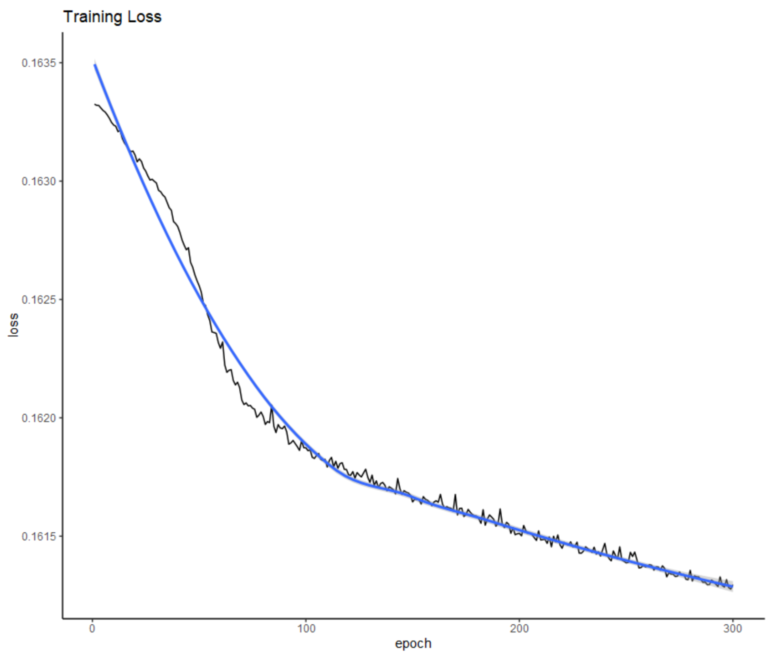


Figure 6.4 - The deviance of the neural network as the number of training iterations increase.

As expected in Figure 6.5, we do observe that the CANN performs similarly to the GLM but also slightly better in some areas such as the vehicle age but we also observe that not all cases did actually improve the fit but instead slightly worsened it. It is natural given the S-shaped nature of the tanh-activation function that some observations will get a worse outcome than observed, especially when we force the NN to train around static parameters, since there is a chance that certain parts of the feature space might be forced into the extremes of the activation space. an illustration of this effect can be seen in Figure 6.6.

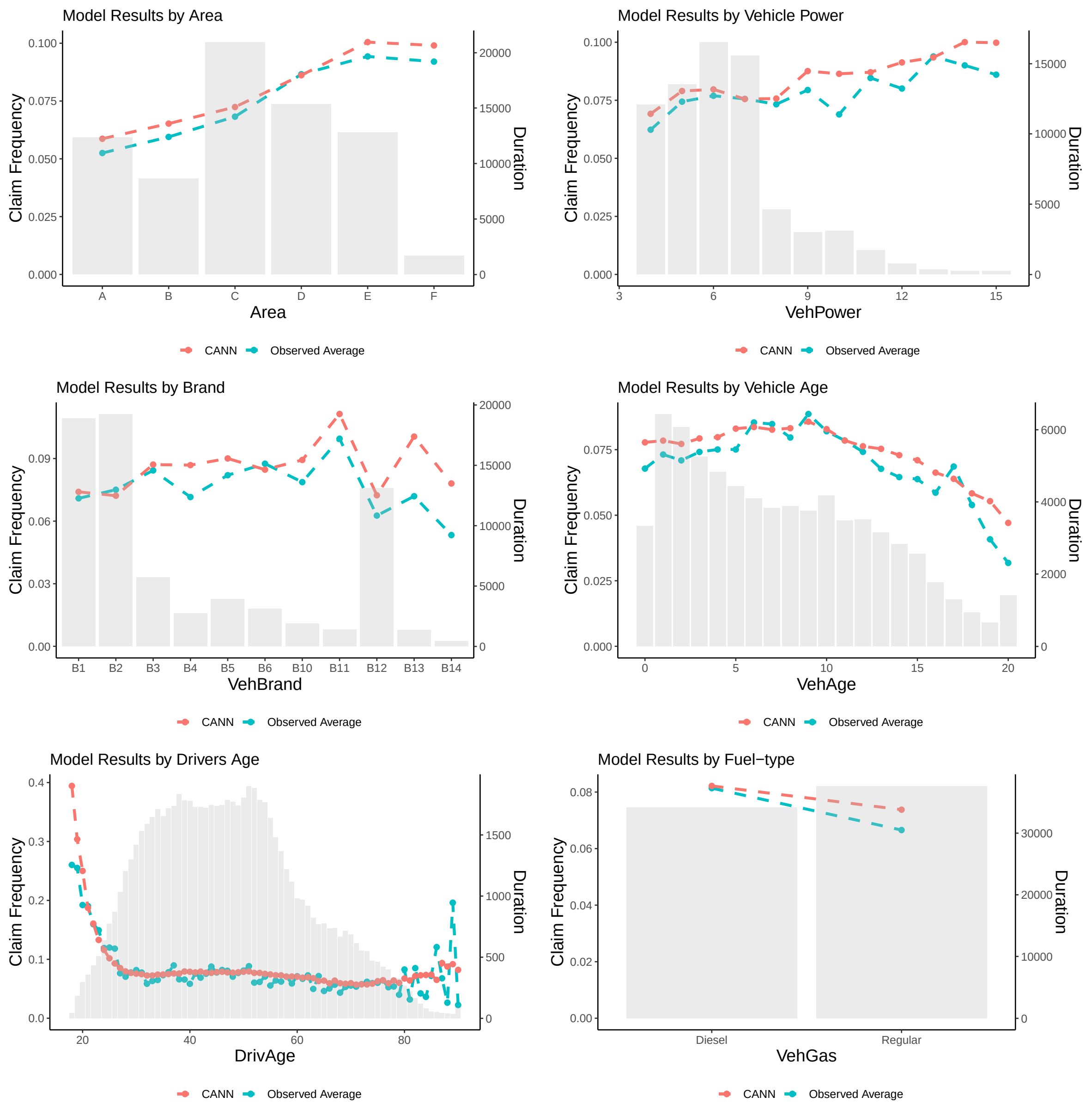


Figure 6.5 - Grouped plots for CANN results.

An interesting point of observation is the effect that the NN adds to the GLM which we can extract from the CANN outcome. In section 3.2.3.4 we showed how the CANN can be split into an "NN"-part and a "GLM"-part by splitting up the results of the linked output such that $\hat{y}_i^{(CANN)} = \hat{y}_i^{(NN)}\hat{y}_i^{(GLM)}$. Since the neural network is trained with a static GLM layer where the activation function is the en exponential function we can simply solve for y_i^{NN} from the outcome and retrieve how much of the prediction that the neural network amounts for. In Figure 6.6 we observe the weight of the neural network amounts to an average average improvement to the GLM by 0.0018 which is a fairly small effect, this is one likely reason to why the observed difference in deviance between the GLM and the CANN is small.

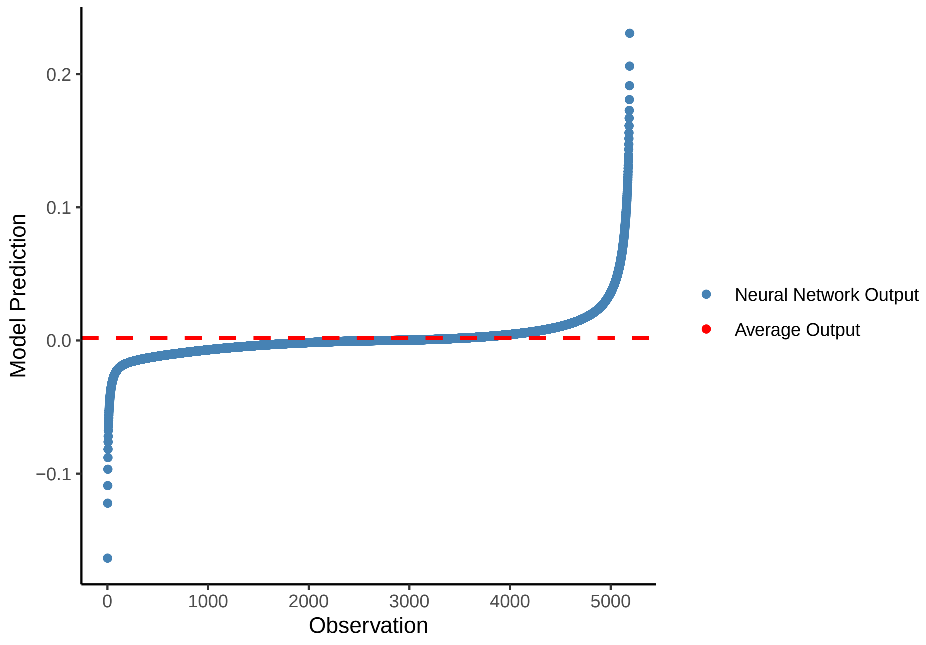


Figure 6.6 - plot of $\hat{y}_i^{NN}$ - Only every 100th observation is included for efficiency.

7 Discussion

7.1 Model fit & Performance

We initially expected the Machine-Learning models to perform better than the GLM due to their flexible ability to fit to data. However, the level of improvement was expected to be greater. It can be quite natural to expect Machine-Learning model (especially a Neural Network) to outperform the GLM since there are many cases where data might be complex enough to break (for lack of better words) the GLM:s internal structure. An example of such cases could be data for image- or voice-recognition where you essentially need to fit millions of complex data points in order for the model to be viable. However, the recent up-tick in AI-popularity might also be a factor that in one way or another is a source of bias to the analyst, in the sense that it sets a level of expectation prior to the analysis itself. When initiating a modelling process with biased expectations it can often lead to the point where one starts to doubt reasonable results, all because it does not match what the analyst wants to see. Machine-Learning models, especially NN:s, are often talked about as models that will render GLM:s obsolete but one should always bear in mind what the appropriate model is for the data at hand. In the case of this thesis we observed that the GLM performed almost as well as the Machine-Learning models so if there is not a lot to gain from using the more complex structure one should really ask if it is worth the while.

In Section 6 we observed that the predictive factor for diesel-powered vehicles received a substantial improvement with the CANN while gasoline-powered vehicles actually worsened, even to the point where they were largely overestimated. Simply looking at the deviance to determine how well the model fits data is valuable but it will not explain how adverse selection, as a result to the chosen price strategy, effects the portfolio. In this particular example, Figure 6.5 suggests that we are over appreciating the risk by about 2% if we chose to use the CANN as the primary price strategy. While 2% might not seem like a dramatic overestimation one must bear in mind that it is 2% on average. This essentially means that the predicted risk can vary and on top of that we are talking about overestimating the risk for more than half the portfolio. Overpricing is a common issue in insurance which can cause problems with adverse selection and seeing and one could conclude that it is a problem that needs to be addressed if the CANN is chosen as a primary price strategy. Splitting data between two or more models can be advantageous however, instead of choosing between strengths and weaknesses between two model architectures one can instead use both models for different data partitions to allow them to complete their weaknesses instead of competing against each other. In Figure 7.1 the data has been split up such that all observations where the diesel-powered vehicles are fitted by the CANN and the gas-powered vehicles by the GLM, and by the looks of it, it seems that this combined model predicts the test-data better than both architectures separate. While data partitioning might not be the go-to method it might still prove to be a worthy tool since most modern IT-tools does support model-splitting in one way or another, regardless if its done by a dedicated functionality or by regarding the split-dataset as separate tariffs.

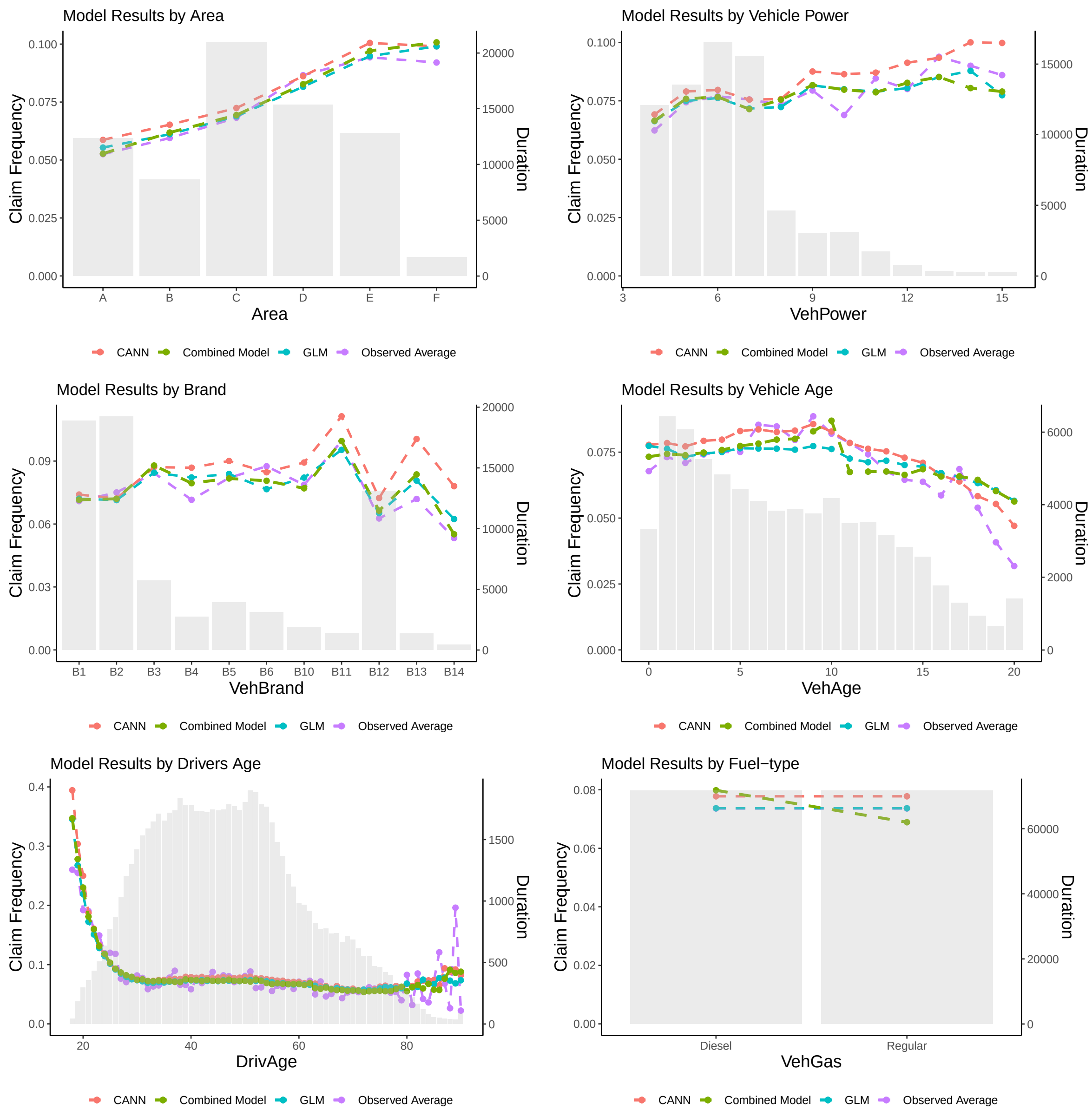


Figure 7.1 - Grouped plots for combined models results

It can be observed in Table 7.1 that combining the GLM and the CANN frameworks as previously described we get a performance boost that none of the models can compare to alone. It seems that there are some advantages to be gained when allowing the models to discard data observations that weakens them.

Model	Parameters	Train Deviance	Test Deviance
Model 1: GLM - Without Polynomial transformations	46	0.2408	0.2416
Model 2: GLM2 - With Polynomial transformations	46	0.2408	0.2416
Model 3: Standard Neural Network	686	0.2391	0.2406
Model 4: CANN	1715	0.2370	0.2404
Model 5: GBM	348	0.2336	0.2402
Model 6: GLM & CANN Combined	78	0.2397	0.2386

Table 7.1 - Summary of all models

The GBM architecture is a rather interesting one since it quite a powerful tool while simultaneously being the easiest one to overfit, while the CANN is somewhat restrained by the underlying and non-trainable model, the GBM is literally free to descend along the gradient and generate an endless amount of models unless it's stopped. Since the weak learner in this case is a Regression Tree the GBM requires some data transformations in the sense that the architecture requires does not allow categorical variables, despite this, the GBM is a surprisingly effortless tool to work with. Besides from having a rich theoretical framework, GBM:s are also well equipped in the software section where most modern programming languages supports it in some way or another. Most tools such as R, Python or even SAS offer reliable solutions to assist the analyst in easily setting up and tweaking a model in an almost plug-and-play manner, with some minor learning bumps of course. The drawbacks however are quite clear, the outcome of the GBM can be rather messy due to the visually incoherent and nestled nature of additive regression trees, Figure 10.4 in the appendix shows a visualization of a more aggressively pruned tree from the GBM process.

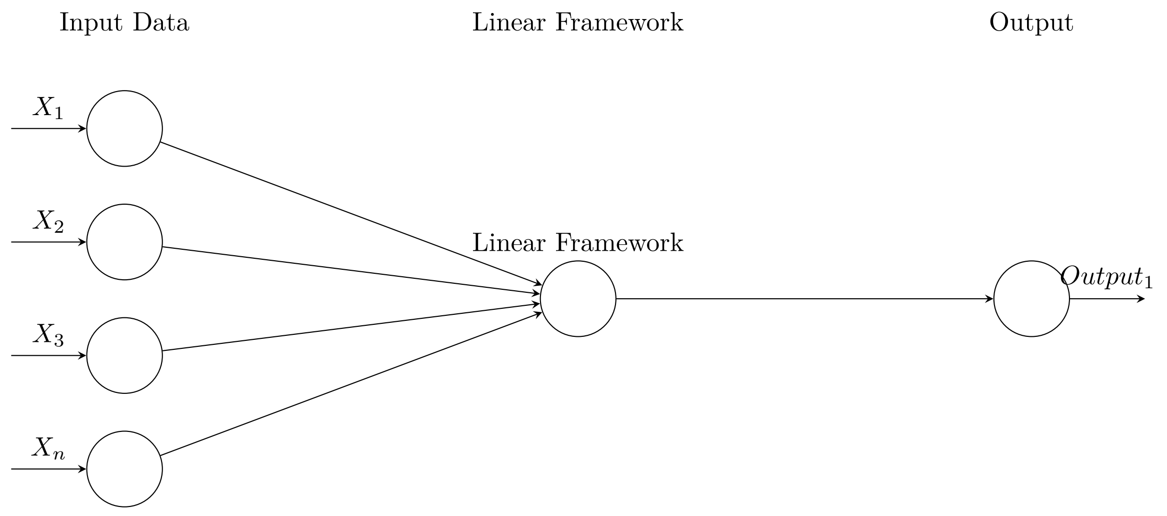
Its hard to objectively say which model is the best one since that question not always answered by the lowest prediction error. If one only looks at it from the statistical perspective then the machine-learning models might often outperform GLM:s on linear as well as non-linear but as mentioned earlier there are cases where the GLM is a more appropriate tool. Although machine-learning models are more advanced one should still regard the necessity of advanced analytics for the data at hand; a carpenter could hammer a screw into a piece of wood and it will probably do the job for the most part, however, it cannot be denied that a screwdriver would be a more appropriate tool for that specific task. In a similar example the analyst might find that the need for advanced analytics is not always there due to various reasons such as low data quantity, deviating data distributions, low data complexity etc.

7.2 Interpretability & Practical usage

A great issue with Machine-Learning models is that they are regarded as black-box models in the sense that a most of the path that the data takes is happening behind the curtains whereas the analyst rarely studies the dataflow on a detailed level, while the analyst might be certain that his model does the job it is likely that some individuals might discard the mathematical backbone of the model as jargon. The GLM architecture has been around for a long time and has proven itself to be a worthy tool for modelling linear data so it might not come to a surprise Machine-Learning models does in fact encounter resistance when talked about. It is hard to say where you should draw the line for when a model should be regarded as a black-box, when it comes to the interpreting the model output this thesis has showed that it can be done in a simple manner that is similar to how it is commonly done with GLM:s in real insurance companies. While GLM:s are regarded as highly understandable, even to the point that practically anyone in a corporate environment could grasp the outcome fairly well one cannot say the same about its underlying architecture as it often requires deeper understanding to fully grasp. If one stops to think about it for a second, the Maximum-Likelihood process could be regarded as a black-box for anyone without a mathematical background. We do argue that that the black-box argument that is commonly made for Machine-Learning models perhaps is not a valid one since clearly these models are not designed to value mid-level analysis such as ANOVA or similar frameworks, at least not to the same extent as GLM:s do. While the internal mechanics of any model can be somewhat confusing the output can often be made understandable for any framework.

Perhaps the CANN could be a bridge between GLM:s and Machine-Learning frameworks, since it is trained around a GLM then we will always have the interpretability what comes with as a foundation to the model. In a general sense the CANN does not disturb the normal process of training a GLM greatly since it essentially acts as an added step at the very end. Loosely speaking it merely trims the hedges of the GLM residuals and makes slight improvements to it, perhaps this behaviour would be somewhat easier to explain than pure Machine-Learning models. While the black-box argument is still present it might be an acceptable counter-argument that its presence is small and it only nudges the GLM towards better results. From an objective perspective the CANN is showing better results that does not differ greatly from the GLM for the particular data in used in this thesis, at the very least it could be used as a tool to further validate the underlying GLM.

There is an essential part of the NN that is hard to explain, namely why the optimal model structure performs the way it does, an analyst could perhaps study the relation between all weights in the model but even so it will not result in an intuitive explanation to why a certain set of neurons performs better than a slightly different set, say for example a a model with 29 layers in comparison to a model with 28 layers. GLM:s have grown into the industry standard partly because they bridge the gap between analysts and non-analysts by being light-weight from an interpretation perspective. Varying performances in GLM:s often has a natural explanation such as the way certain transformations alters the fit, how smoothing functions makes covariate less "jumpy" or how interactions cancel out the negative effects of correlated features. The NN, although being a powerful tool, is often a great source of confusion for individuals who lack knowledge of how it works, how is is trained or how it passes data. The GLM:s in comparison, can often be broken down in simpler terms that are user-friendly to almost anyone. While there are plenty of mathematical reasons to why a specific NN performs as it does it is rarely an acceptable solution to say "Just trust the math!" when pitching a new price strategy, for this reason it is often convenient to just stick to GLM:s. Figure 7.2 illustrates how GLM:s and NN:s pass data through their frameworks and it is quite obvious that the NN is less intuitive than the GLM, perhaps it might even be intimidating for some observers. Bear in mind that the network illustrated in figure 7.2 is considered to be a relatively small one with a low level of complexity.



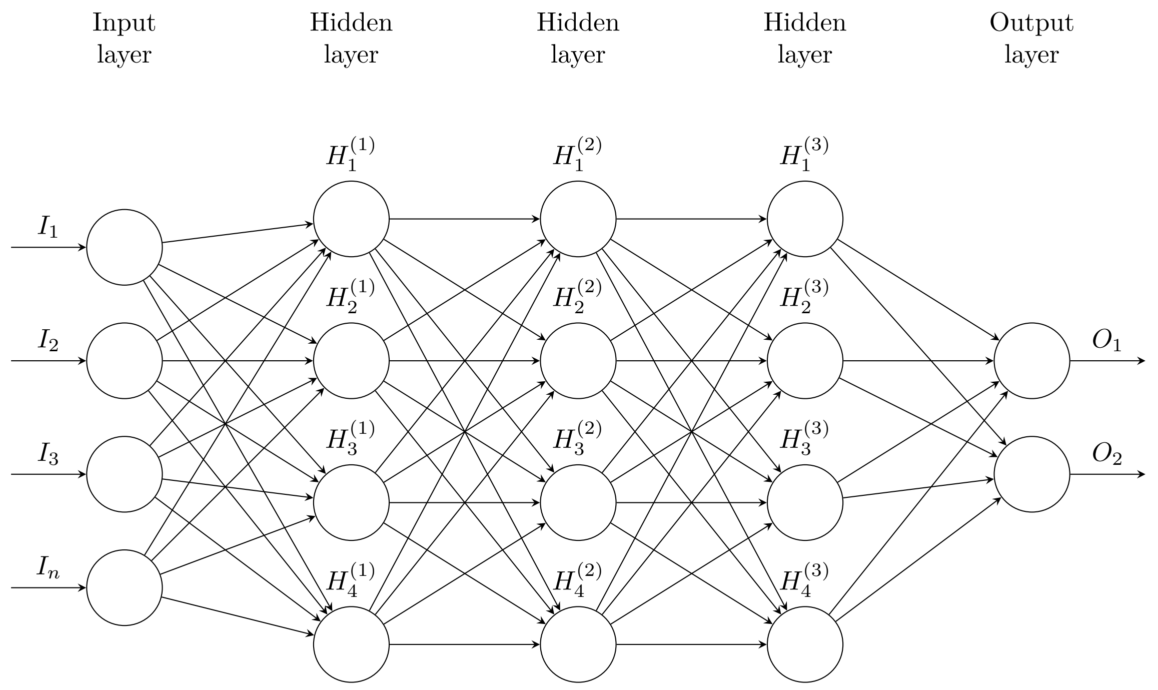


Figure 7.2 - Visual representations of Linear Models and Neural Networks architectures

As the author of this report I do believe that the future of actuarial analytics will at the very least have a minor inclusion of the Machine-Learning frameworks. While NN:s has been theoretically available already since 1960:s the general public has had little to no interest of it. Since it offered little application possibilities given the computing standards of the day, and with the fact that change often requires time, it often leads to the development of analytical standards being a slow race to win. However, as previously mentioned the computing standards of the day are ridiculously high compared to the standards of just a few of decades ago, arguably, the technological roof that slows down development is yet to be reached and one thing that is sure about humans is that we are a curious race who always seek to improve what we have.

Insurance companies (in Sweden) are required to set a reasonable premium, that is in line with the observed risk and without unreasonable discrimination and this could be one of many, if not one of the major reasons to why interpretability is a highly demanded factor when launching new price models. Should it happen that a policyholder feels unjustly discriminated then the insurance company have to supply a good enough explanation to why this individuals premium differs to that of his/her expectation, it does not take much to see that it can be somewhat tricky to explain the price if the primary price engine consists of a neural network with 15+ layers, each with a depth of 30 neurons and variety of skip-layers that improve the fit for some unknown reason. Perhaps the market just needs some time to mature into trusting the process, surely there had to be some black-box-alike arguments against GLM:s when they were first introduced as a tool for pricing, and given some time it became the industry standard for pricing.

We are to some extent generalizing the complexity of the machine-learning models with the notion of them being just as easy to interpret or explain as GLM:s since the argument is heavily relying on the explanation being similar as done in this thesis, and we have essentially largely ignored large chunks of the data path. While this could possibly be restricting analytical possibilities the purpose of this thesis is not to argue for a revolutionizing change in actuarial pricing standards but to show that methods that are regarded as complex and incomprehensive can be used in a comprehensive way that actually helps to improve the risk management, all while simultaneously keeping the essence of what makes the standardized methods of the day so widely used.

A significant drawback of the NN, that was briefly mentioned in Section 7.1, is the availability of predefined tools. Due to the lack of standardization it can be rather time consuming to just have the model up and running, this effect can often be amplified when the person who is developing new models is currently in a digital learning process. The actuarial world is a rather small one that most non-insurance people never really hear of which often leads to the development of digital tools to lag behind the rest of the tech-world. To develop CANN models for this thesis a combination of R, Python and Tensorflow has been utilized where almost every step has been a time consuming custom-job that is specifically made to answer the questions of this thesis, all while satisfactory model versions for the GLM and GBM frameworks was trained and tuned in a matter of days. Perhaps the future will provide easier ways to work as the demand for modelling tools increases but as of today one can expect having to customize code with little reproductive abilities and somewhat lacking documentation.

7.3 General Comments

Most large-scale businesses of the day are old institutions that are heavily invested in outdated software solutions and database architectures might find it hard to transition from one way of analysing data to another, this might be one of the leading causes to why the rate of change is slow. It is often hard to motivate costly changes when the financial gains are marginal, as we have seen our example to be. One thing is for sure however, if one actor on the insurance market finds financial sentiments in using Machine-Learning models and if it causes an influx of unwanted risks to the rest of the market, then the pressure will increase on everyone the get back ahead in the race. This is a natural way of reasoning and often also a phenomenon that can be observed in any arbitrary insurance market, since theory suggests that better risk assessments tend to give high-risk customers the incentive to swap insurers in favour of whoever offers the cheapest premium.

A scenario where machine-learning models might be less adequate to use might be segments where the insurer is mandated by law or by internal corporate policy to price a segment in a certain way. There are few to none regulations in Sweden that meddle with an insurers tariffs, besides from regulations that are anti-discriminatory, but internal policies is a common occurrence. An example of internal policies could be if the company has decided to set a fixed rate for some group of policies that they wish to attract. While GLM:s offer a neat way to offset variables that allow the model to be effortlessly trained around some offset factor, the same cannot be said about machine-learning models. Even passing offset values in a CANN will not stop the “NN”-part of the model from trying improve on the residuals. This is not to say that it is impossible to solve the issue, if a Neural Network is used as the main price strategy in such a case then one could possibly use skip-connections in a cleaver way to help train the model around an offset. It might be tempting to just add manual rules after the model has finished training but this should be avoided since it is important that the model is “aware” of the offset somehow, adding manual rules to finished model might lead it to have an inappropriate baselevel (intercept) which in turn might cause over- or underfitting. Similar issues can be stated for the GBM. While it does have the possibility to be trained on GLM:s the gradient descent process will still try to improve the fit for all inputs. On top of that most programming languages offer little to no freedom in micro-managing the GBM:s weak learner prior to training. This probably means that the analyst will need to customize tools of his own to achieve what is intended, most issues are solvable but there is also a workload-to-reward ratio and a question of sustainability that often has to be accounted for since custom codes, manual rules and Ad-Hoc work tend to stay in production longer than the analyst is planning to stay around which is a scenario that comes with problems of its own.

There is an interesting side-effect that Machine-Learning frameworks brings to the table and that is that they are quite better at analysing granular data than GLM:s. Due to their architectures they tend to learn non-linear relationships that GLM:s might miss out on. GLM:s are designed to work for linear data and it does the job very well in terms of regression. However, they are not as flexible as NN:s and may not be able to accurately model highly non-linear or complex relationships in the data. Should the use of extremely granular data become a common practice in in the insurance market then it would be fair to ask where the future development will focus. Since granularity often implies that fewer policies are targeted per group it should logically lead to a point where making finer adjustments simply will not be of interest since the financial gains might be too small.

8 Conclusion

This thesis studies if Machine-Learning models such as Combined Actuarial Neural Networks and Gradient Boosting Machines can perform better than Generalized Linear models for an insurance portfolio in a real-world application. The initial expectation (as it often might be) was that the Machine-Learning models would significantly outperform the GLM due to their more advanced architecture but to some surprise they only performed slightly better. This is largely explained due to the GLM being a good estimator to begin with. While the slight improvement of the Machine-Learning models is impressing despite the small magnitude it is somewhat ironic the best performance of all models came from combining the CANN and GLM for different partitions of the data. Perhaps it should not come as a surprise since combining models essentially allows the analyst to tune the final prediction so that the weaknesses from each architectures are toned down while simultaneously highlighting the strengths.

The question of which model is the best is not a question that always has a clear answer. The reason being that there is a wide variety of model architectures that are designed for various purposes and data assumptions, this implies that different model architectures can show varying levels of performance for different kinds of data. Even the predictive capabilites of a GLM can vary depending on on what distribution assumption is made when configuring the model. Using Machine-Learning models is often advantageous for data with high levels of complexity since they are not limited to the linearity of GLM:s, they are therefore more versatile and can often find patterns in the data that might even be hard for a human eye to see. There is a famous tool made by the engineers at Tensorflow which allows the user to create NN:s for hypothetical data where it can be seen how the fit changes as the model complexity rises and its a good visual example on how Machine-Learning algorithms can adapt to unusual data patterns (An illustration can be found in Figure 10.5 in the appendix). While Machine-Learning models often tend to perform better for complex data we observe that, for our particular dataset, the statistical advantages are not dramatic. We return once more to the point made earlier in this thesis, it is wise to consider which tool that is appropriate to use for any given dataset; the data should decide what model that fits it the best, not the other way around!

The keyword that concludes this thesis is improvement. The results show that the CANN as well as the GBM and the NN does have an improving effect compared to the GLM. Regardless of the improvement being small or not the Machine-Learning models seem to perform better, since insurance portfolios are often large and carry the risk of a significant amount of individuals in society we know that small improvements to the fit often can lead to large improvements in monetary value while simultaneously benefiting the policies that subsidize most of the risk. All in all Machine-Learning seems to be a positive addition to the actuaries tool-box and hopefully the industry will find ways to incorporate them into the risk management standard given enough time.

9 References

[1] Esbjörn Ohlsson, Björn Johansson. (2010) *Non-Life Insurance Pricing With Generalized Lined Models.* Berlin, Heidelberg: Springer.

[2] Ian Goodfellow, Yoshua Benigo, Aaron Courville. (2016) *Deep Learning.* Cambridge, Massachusetts: The MIT Press.

[3] John Hertz, Richarg G. Palmer, Anders Krogh. (1991) Redwood City, Calif: Addison Wesley, cop.

[4] Mario V. Wüthrich, Christoph Buser. (2023) *Data Analytics for Non-Life Insurance Pricing.* Swiss Finance Institute Research, 16-68.

[5] Michel Denuit, Donatien Hainaut, Julien Trufin. (2019) *Effective Statistical Learning Methods For Actuaries III - Neural Networks and Extensions.* Switzerland: Springer Actuarial - Springer Nature.

[6] Mario V. Wüthrich, Andrea Ferrario, Alexander Noll. (2020) *Insights from Inside Neural Networks.* SSRN, Manuscript ID: 3226852

[7] Steven Boyd, Lieven Vandenberghe. (2004) *Unconstrained Minimization. Convex Optimization.* New York: Cambridge University Press.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. (2015) *Deep Residual Learning for Image Recognition.* New York: Cornell University, arXiv:1512.03385.

[9] Jerome H. Friedman. (2001) *Greedy function approximation: a gradient boosting machine.* Annals of Statistics, 5 1189–1232.

[10] CASdatasets Package Vignette. Published 2020-12-11. Version 1.0.11

[11] Trevor Hastie, Robert Tibshirani, Jerome H. Friedman. (2009) *The Elements of Statistical Learning.* New York: Springer.

[12] McElreath, Richard. (2016) *Statistical Rethinking: A Bayesian Course with Examples in R and Stan.* Florida, Boca Raton: CRC Press.

[13] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, Charles J. Stone. (1984) *Classification And Regression Trees.* New York: International Biometric Society.

[14] Daniel Graupe. (2013) *Principles Of Artificial Neural Networks (3rd Edition).* Singapore: World Scientific Publishing Company.

[15] Mario V. Wüthrich. (2017) *Non-Life Insurance: Mathematics & Statistics.* SSRN, Manuscript ID: 2319328.

[16] Harold James, Peter Borscheid, David Gugerli, Tobias Strauman. (2013) *The Value of Risk: Swiss Re and the History of Reinsurance.* United Kingdom: Oxford University Press.

10 Appendix

10.1 Comparative model graphs

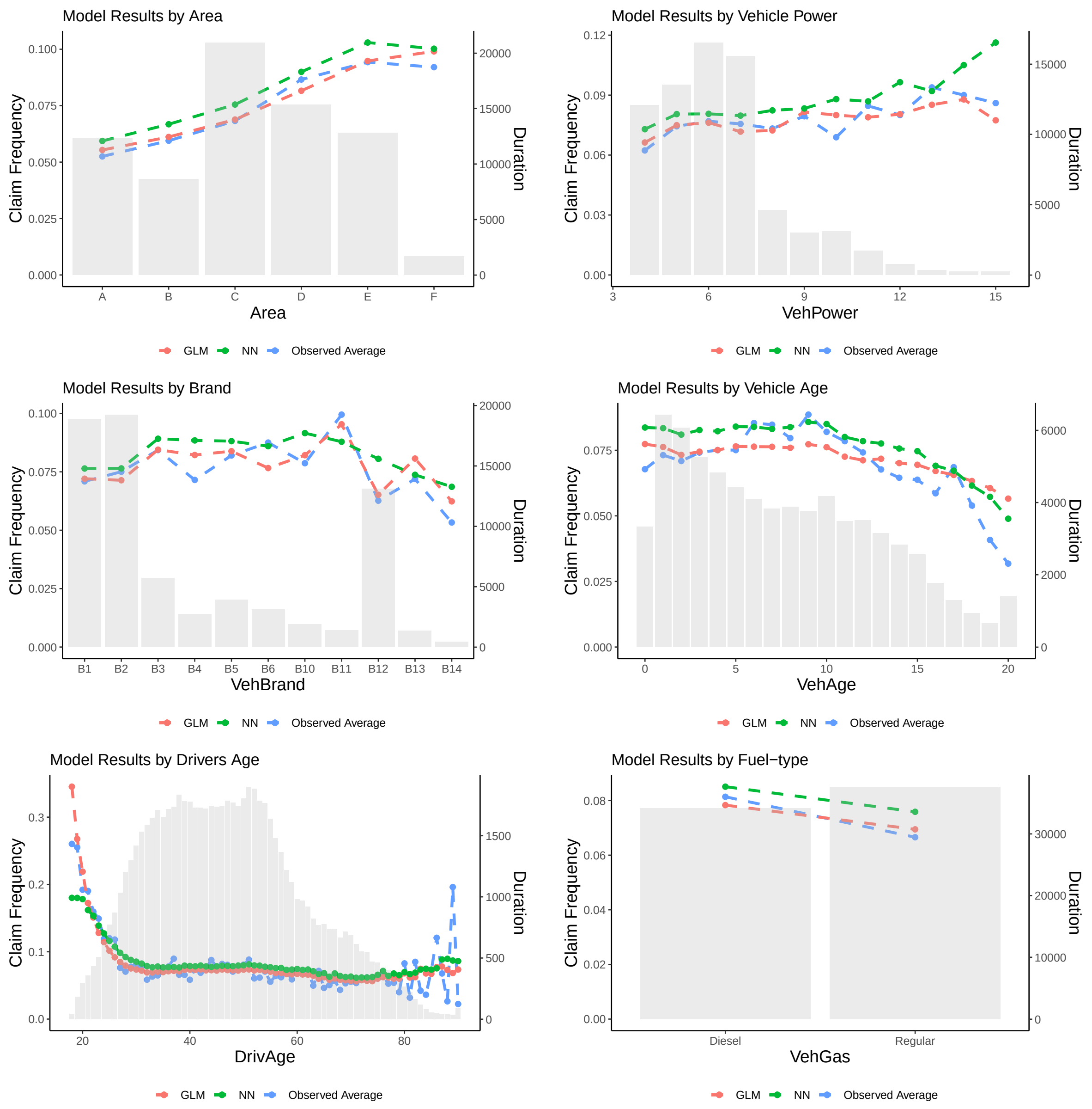


Figure 10.1 - Plain Neural Network without embedded models vs GLM

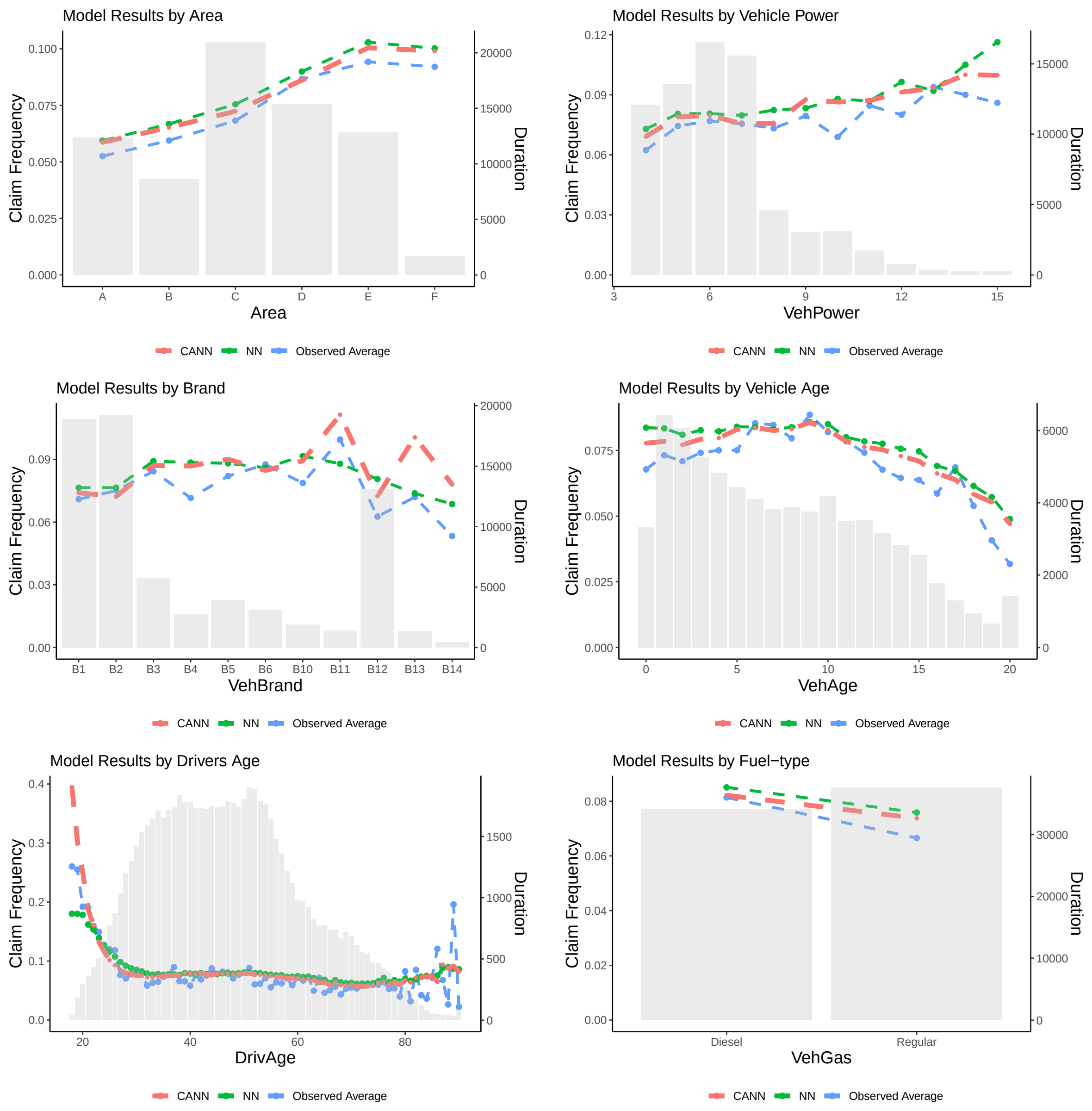


Figure 10.2 - Plain and un-weighted Neural Network vs CANN

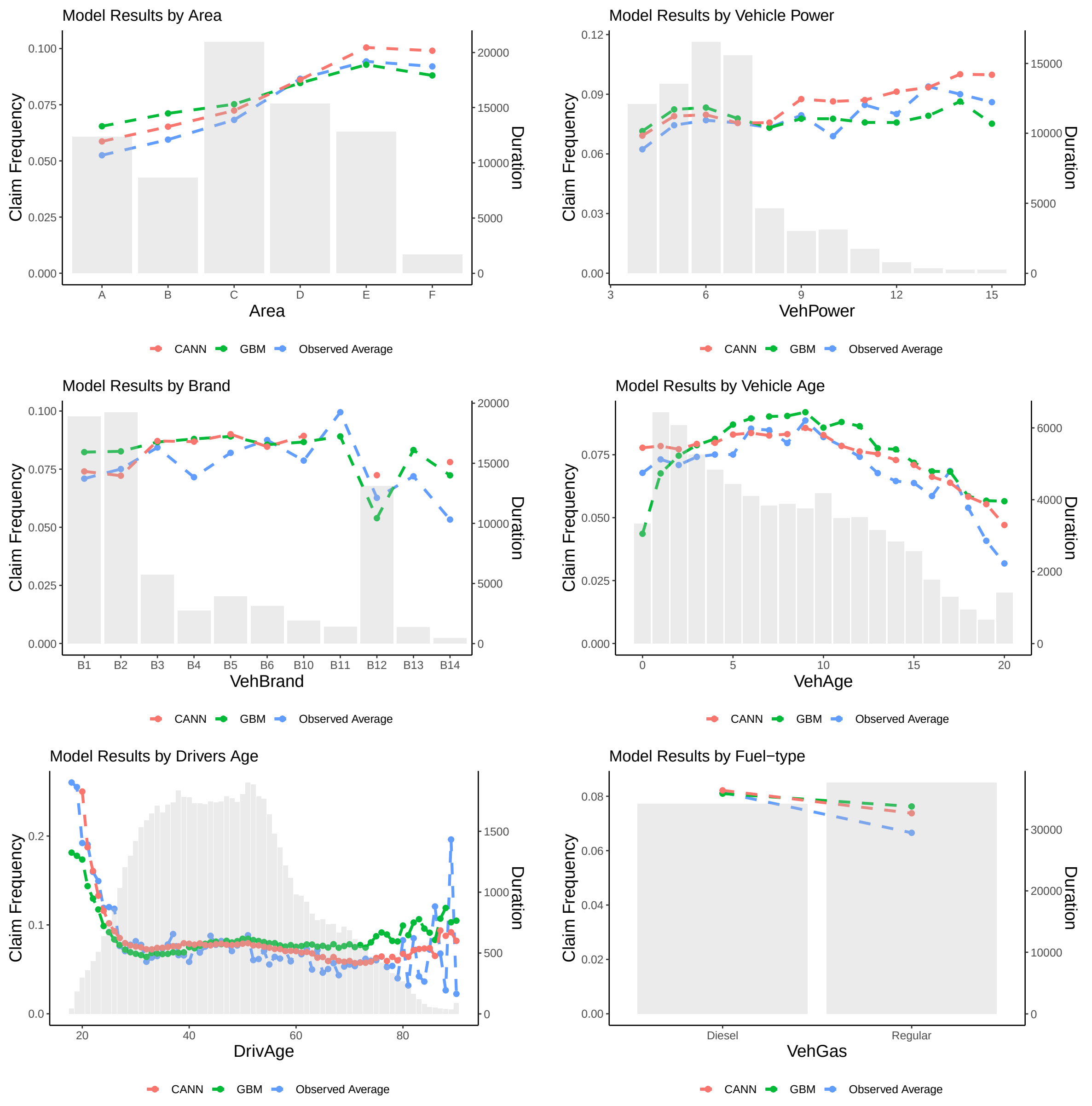


Figure 10.3 - CANN vs GBM

10.2 Miscellaneous

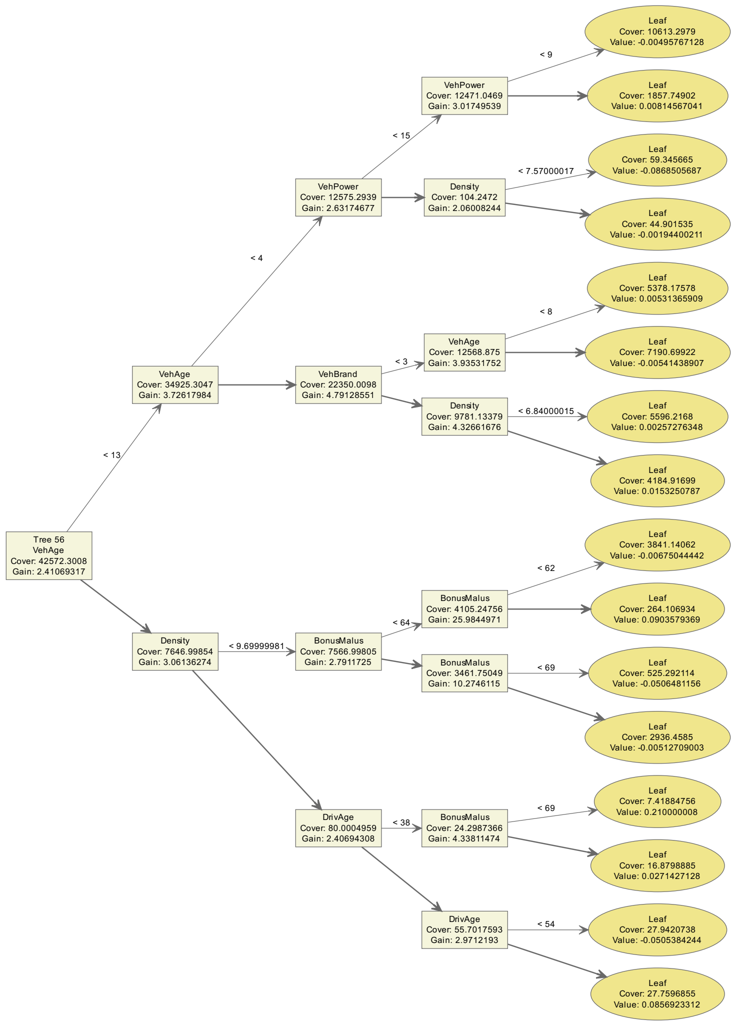


Figure 10.4 - An illustration of a heavily pruned tree which is trained under the same conditions as the optimal GBM model.

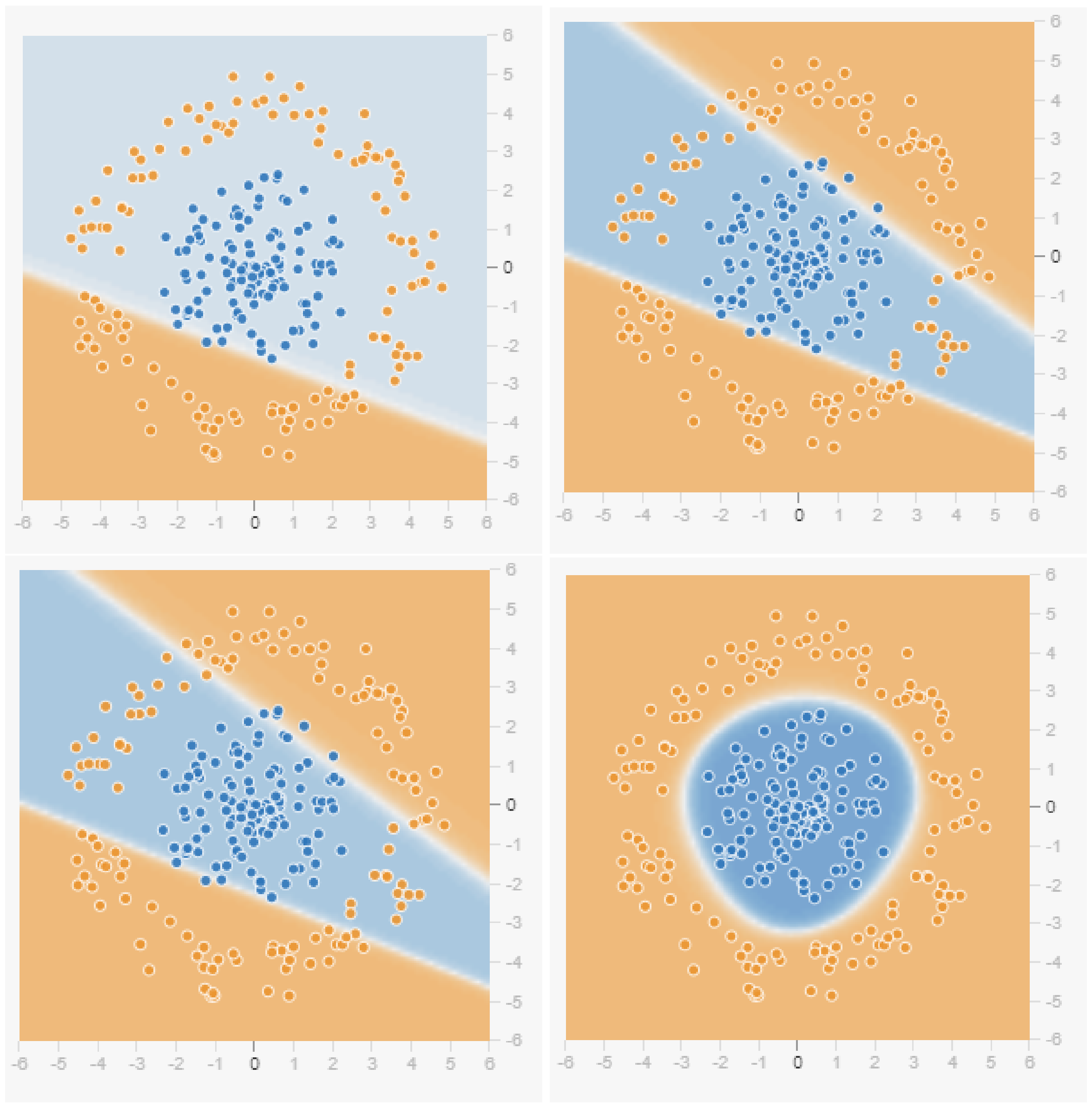


Figure 10.5 - A Tensorflow Playground illustration of a 3-layer Neural Network where the depth is increased by 1 unit per graph starting from the top-left image.