ACTUARIAL SOCIETY OF SOUTH AFRICA

Quantifying Risk, Enabling Opportunity.

Exploring neural network predictions for insurance problems

Ronald Richman (FIA, FASSA, CERA)
Associate Director
QED Actuaries & Consultants

28/08/2020

QED Actuaries & Consultants

Agenda

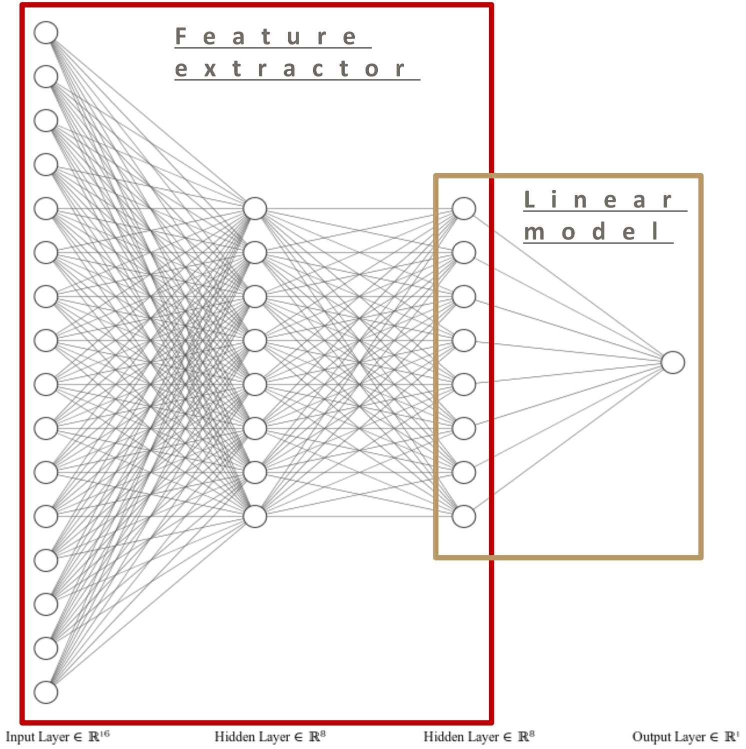
- Introduction
- Aggregating Predictors
- Networks and Aggregating
- Example: French Motor Third-Party Liability Insurance
- Conclusion

Rationale

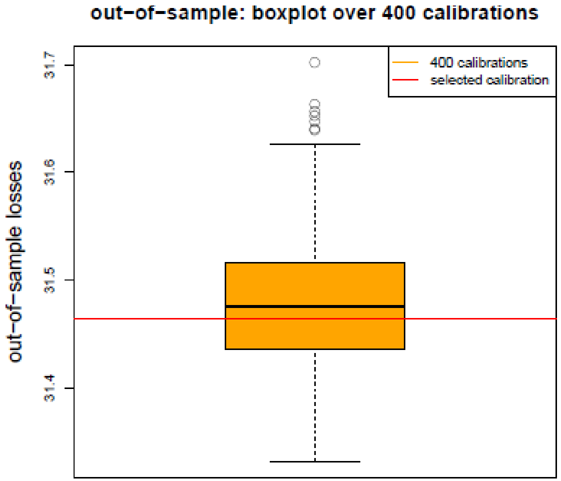
- Deep learning currently producing highly accurate models on diverse types of data:
  - *Within actuarial science – pricing, reserving, mortality forecasting, analysis of telematics data*
  - *In more general domains – computer vision, natural language processing, generative modelling, timeseries forecasting*
- On the other hand, training process leads to variable results:
  - *Aggregate level – performance varies depending on training run*
  - *Policy level – greater variability than aggregate level*

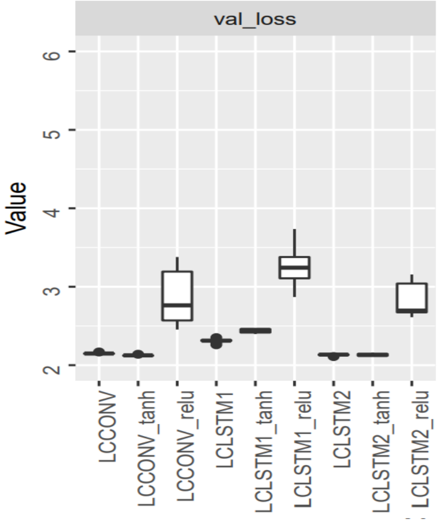
Fully Connected Neural Networks

- Intermediate layers = representation learning, guided by supervised objective
- Last layer = (generalized) linear model, where input variables = new representation of data
- No need to use GLM – strip off last layer and use learned features in, for example, XGBoost
- Or mix with traditional method of fitting GLM


Feature extractor
Linear model
Input Layer ∈ $\mathbb{R}^{16}$
Hidden Layer ∈ $\mathbb{R}^{8}$
Hidden Layer ∈ $\mathbb{R}^{8}$
Output Layer ∈ $\mathbb{R}^{1}$

Recent Examples





Neural networks fit to French MTPL dataset
Richman and Wüthrich (2020)

Neural networks fit to HMD dataset
Perla, Richman, Scognamiglio and Wüthrich (2020)

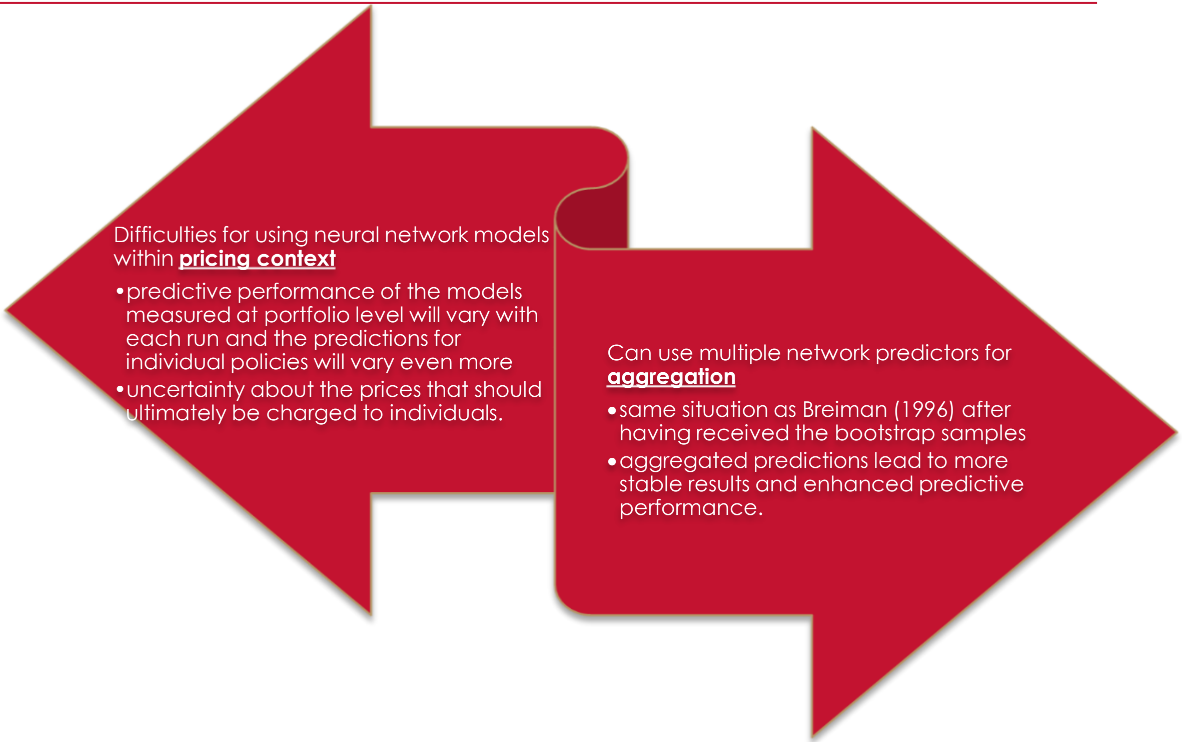
Nagging Predictors

Richman, Ronald; Wüthrich, Mario V. 2020. "Nagging Predictors." Risks 8, no. 3: 83.

- Aggregating is a statistical technique that helps to reduce noise and uncertainty in predictors and is justified theoretically using the law of large numbers.
- An i.i.d. sequence of predictors is not always available thus, Breiman (1996) combined **b**ootstrapping and **agg**regat**ing**, called **bagging**.
- This paper aims to combine **n**etworks and **agg**regat**ing** to receive the **nagging** predictor.
- Explore the statistical properties of the nagging predictors at a portfolio and at a policy level.

Consequences of using Neural Networks

- Neural network training produces infinitely many equally good predictors via gradient descent algorithms.
- Common neural network training techniques may even lead to more randomness in neural network results, for example:
  - Stochastic Gradient Descent; and
  - Dropout.



Difficulties for using neural network models within **pricing context**

- predictive performance of the models measured at portfolio level will vary with each run and the predictions for individual policies will vary even more
- uncertainty about the prices that should ultimately be charged to individuals.

Can use multiple network predictors for **aggregation**

- same situation as Breiman (1996) after having received the bootstrap samples
- aggregated predictions lead to more stable results and enhanced predictive performance.

Agenda

- Introduction
- Aggregating Predictors
- Networks and Aggregating
- Example: French Motor Third-Party Liability Insurance
- Conclusion

Notation and Definitions

- In context of non-life pricing for a policy i, define a regression model $\mu(.)$ for claims Y_i, based on covariates X_i and exposures v_i:

$$\mathcal{X} \to \mathbb{R}, \qquad \boldsymbol{x}_i \mapsto \mu(\boldsymbol{x}_i) = \mathbb{E}[Y_i].$$

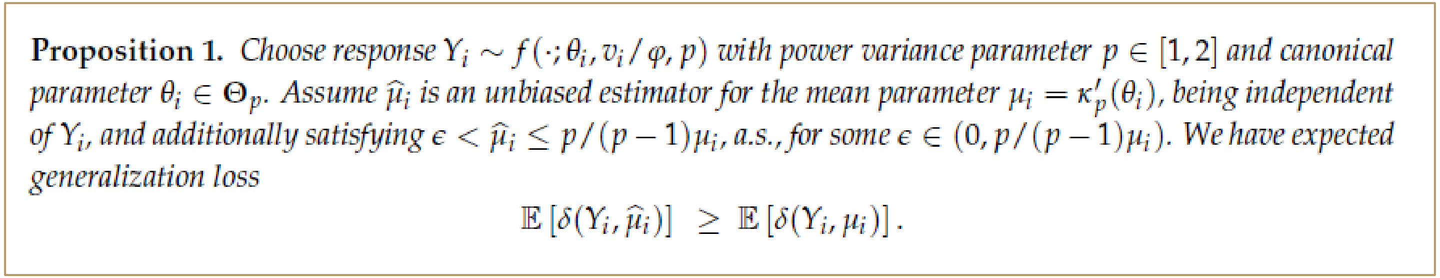
- Here, we approximate μ with a neural network:

$$\boldsymbol{x}_i \mapsto g\left(\mathbb{E}[Y_i]\right) = \left\langle \boldsymbol{\beta}^{(d+1)}, \left(\boldsymbol{z}^{(d)} \circ \cdots \circ \boldsymbol{z}^{(1)}\right)(\boldsymbol{x}_i)\right\rangle$$

- We fit the network by minimizing the deviance loss under a suitable distributional assumption:

$$\delta(Y_i, \mu_i)$$

Proposition 1



Proposition 1. *Choose response $Y_i \sim f(\cdot; \theta_i, v_i/\varphi, p)$ with power variance parameter $p \in [1,2]$ and canonical parameter $\theta_i \in \Theta_p$. Assume $\widehat{\mu}_i$ is an unbiased estimator for the mean parameter $\mu_i = \kappa_p'(\theta_i)$, being independent of Y_i, and additionally satisfying $\epsilon < \widehat{\mu}_i \leq p/(p-1)\mu_i$, a.s., for some $\epsilon \in (0, p/(p-1)\mu_i)$. We have expected generalization loss*

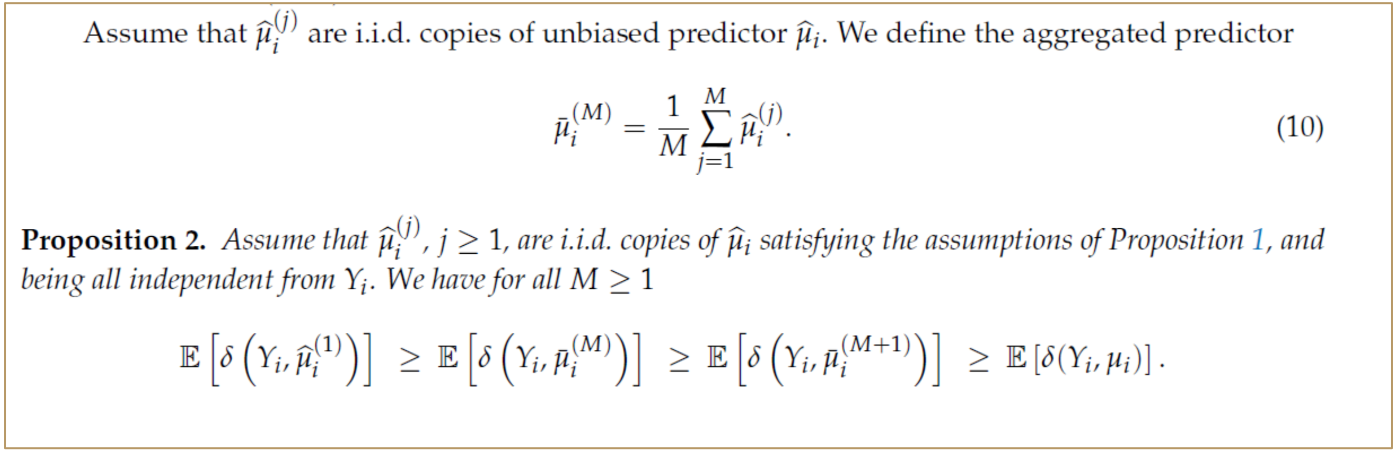
$$\mathbb{E}\left[\delta(Y_i, \widehat{\mu}_i)\right] \;\geq\; \mathbb{E}\left[\delta(Y_i, \mu_i)\right].$$

Proposition 1: model $\widehat{\mu}_i$ has an expected generalization loss which is bounded below by the one of the true model mean μ_i of Yi.

Aggregating = come as close as possible to this lower bound by combining predictors from multiple models.

Assumed that $\widehat{\mu}_i$ is unbiased.

Proposition 2



Assume that $\widehat{\mu}_i^{(j)}$ are i.i.d. copies of unbiased predictor $\widehat{\mu}_i$. We define the aggregated predictor

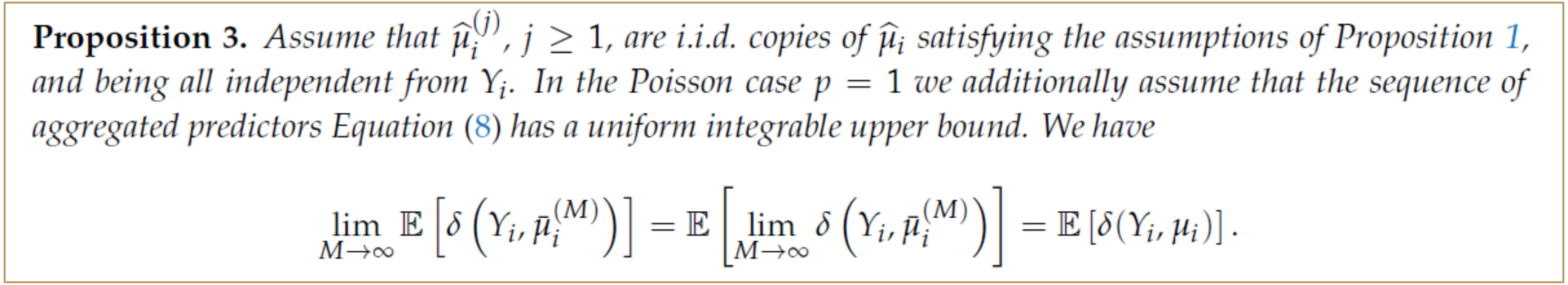
$$\bar{\mu}_i^{(M)} = \frac{1}{M} \sum_{j=1}^{M} \widehat{\mu}_i^{(j)}. \tag{10}$$

Proposition 2. *Assume that $\widehat{\mu}_i^{(j)}$, $j \geq 1$, are i.i.d. copies of $\widehat{\mu}_i$ satisfying the assumptions of Proposition 1, and being all independent from Y_i. We have for all $M \geq 1$*

$$\mathbb{E}\left[\delta\left(Y_i, \widehat{\mu}_i^{(1)}\right)\right] \;\geq\; \mathbb{E}\left[\delta\left(Y_i, \bar{\mu}_i^{(M)}\right)\right] \;\geq\; \mathbb{E}\left[\delta\left(Y_i, \bar{\mu}_i^{(M+1)}\right)\right] \;\geq\; \mathbb{E}\left[\delta(Y_i, \mu_i)\right].$$

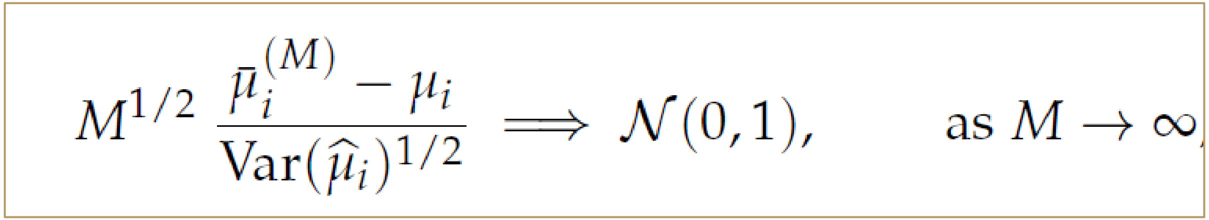
Proposition 2: Aggregation works, i.e., aggregating i.i.d. predictors Equation (10) leads to a monotonically decreasing expected generalization loss.

Proposition 3-4



Proposition 3. *Assume that $\widehat{\mu}_i^{(j)}$, $j \geq 1$, are i.i.d. copies of $\widehat{\mu}_i$ satisfying the assumptions of Proposition 1, and being all independent from Y_i. In the Poisson case $p = 1$ we additionally assume that the sequence of aggregated predictors Equation (8) has a uniform integrable upper bound. We have*

$$\lim_{M\to\infty} \mathbb{E}\left[\delta\left(Y_i, \bar{\mu}_i^{(M)}\right)\right] = \mathbb{E}\left[\lim_{M\to\infty} \delta\left(Y_i, \bar{\mu}_i^{(M)}\right)\right] = \mathbb{E}\left[\delta(Y_i, \mu_i)\right].$$



$$M^{1/2}\,\frac{\bar{\mu}_i^{(M)} - \mu_i}{\mathrm{Var}(\widehat{\mu}_i)^{1/2}} \Longrightarrow \mathcal{N}(0,1), \qquad \text{as } M \to \infty$$

Propositions 3-4: Convergence results

Agenda

- Introduction
- Aggregating Predictors
- <u>Networks and Aggregating</u>
- Example: French Motor Third-Party Liability Insurance
- Conclusion

Network Modelling & Bagging

- Propositions 1–4 based on assumption that we can generate a suitable i.i.d. sequence of unbiased predictors.

- In practical application this is not the case because data generating mechanism is unknown => rely empirical approximations to the true model.

- Strategy: use neural network regression models that are expected to generalize well to unseen data:

  - split data into training/validation/testing sets;
  - fit on training set and assess on validation set; and
  - final model accuracy assessed on test set

- **Bagging:** generate predictors $\hat{\mu}_i^{(j)}$ by bootstrapping training data (Breiman 1996)

The Nagging Predictor

- Neural networks initialized randomly, and parameters calibrated with gradient descent

- To prevent overfitting, training stopped early once network overfits to validation set

- Different parameter set received each time training is run

- **Nagging:** exploit random outcomes of neural network training to receive a sequence of predictors $\hat{\mu}_i^{(j)}$:

$$\bar{\bar{\mu}}_t^{(M)} = \frac{1}{M} \sum_{j=1}^{M} \widehat{\mu}_t^{(j)} = \frac{1}{M} \sum_{j=1}^{M} \mu(\boldsymbol{x}_t^{\dagger}, \widehat{\boldsymbol{\beta}}^{(j)})$$

Bagging vs Nagging

Bagging

- Performs re-sampling on observations => tries to create new observations from the data follow a similar law as this original data
- Re-sampling involves randomness and, therefore, bootstrapping is able to generate multiple random predictors $\hat{\mu}_i^{(j)}$
- Bootstrap predictors are i.i.d. by applying the same algorithm using i.i.d. seeds

Nagging

- Not based on re-sampling data, but works on the same data set
- Multiple predictors are obtained by exploring multiple parametrizations of the same model using gradient descent methods combined with early stopping
- => less randomness compared to bootstrapping because underlying data set always the same

Dependence on data

- Bagging and Nagging fully based on the observed data.
- Only extract information that is already contained in the data.
- If for some reason data atypical, reflected in bagging and nagging predictors and may exhibit poor out of sample performance.

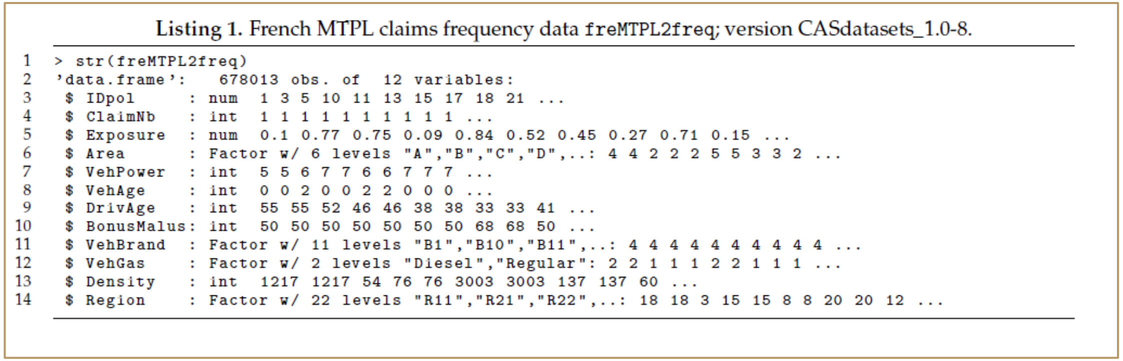
Agenda

- Introduction
- Aggregating Predictors
- Networks and Aggregating
- <u>Example: French Motor Third-Party Liability Insurance</u>
- Conclusion

French Motor Third-Party Liability Insurance

Data

- Explore nagging predictors on real data.
- French motor third-party liability (MTPL) claim counts data set of Dutang and Charpentier (2019).
- Well studied: Noll et al. (2018) and Wüthrich (2019).
- R package CASdatasets, see Dutang and Charpentier (2019).

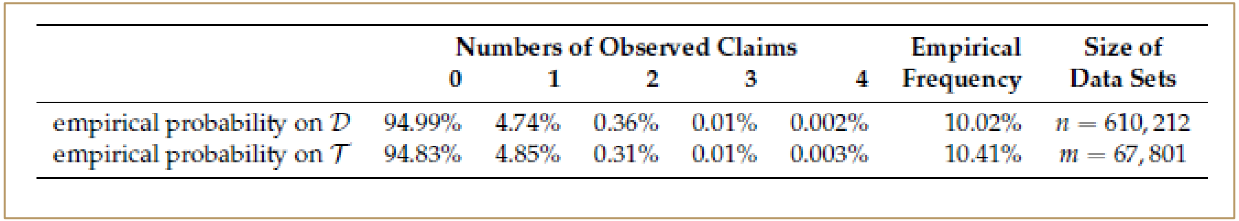


Listing 1. French MTPL claims frequency data freMTPL2freq; version CASdatasets_1.0-8.

```
> str(freMTPL2freq)
'data.frame':   678013 obs. of  12 variables:
 $ IDpol      : num  1 3 5 10 11 13 15 17 18 21 ...
 $ ClaimNb    : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Exposure   : num  0.1 0.77 0.75 0.09 0.84 0.52 0.45 0.27 0.71 0.15 ...
 $ Area       : Factor w/ 6 levels "A","B","C","D",..: 4 4 2 2 2 5 5 3 3 2 ...
 $ VehPower   : int  5 5 6 7 7 6 6 7 7 7 ...
 $ VehAge     : int  0 0 2 0 0 2 2 0 0 0 ...
 $ DrivAge    : int  55 55 52 46 46 38 38 33 33 41 ...
 $ BonusMalus: int  50 50 50 50 50 50 50 68 68 50 ...
 $ VehBrand   : Factor w/ 11 levels "B1","B10","B11",..: 4 4 4 4 4 4 4 4 4 4 ...
 $ VehGas     : Factor w/ 2 levels "Diesel","Regular": 2 2 1 1 1 2 2 1 1 1 ...
 $ Density    : int  1217 1217 54 76 76 3003 3003 137 137 60 ...
 $ Region     : Factor w/ 22 levels "R11","R21","R22",..: 18 18 3 15 15 8 8 20 20 12 ...
```

Learning and Test Data

- Features are pre-processed :
    - use MinMaxScaler for continuous explanatory variables; and
    - two-dimensional embedding layers for categorical covariates
- 90% of all policies allocated to training data D
- Remaining 10% are allocated to testing data T



	Numbers of Observed Claims					Empirical	Size of
	0	1	2	3	4	Frequency	Data Sets
empirical probability on $\mathcal{D}$	94.99%	4.74%	0.36%	0.01%	0.002%	10.02%	$n = 610,212$
empirical probability on $\mathcal{T}$	94.83%	4.85%	0.31%	0.01%	0.003%	10.41%	$m = 67,801$

Neural Network Architecture

We choose a network of depth d = 3 having (q_1, q_2, q_3) = (20, 15, 10) hidden neurons in the three hidden layers.

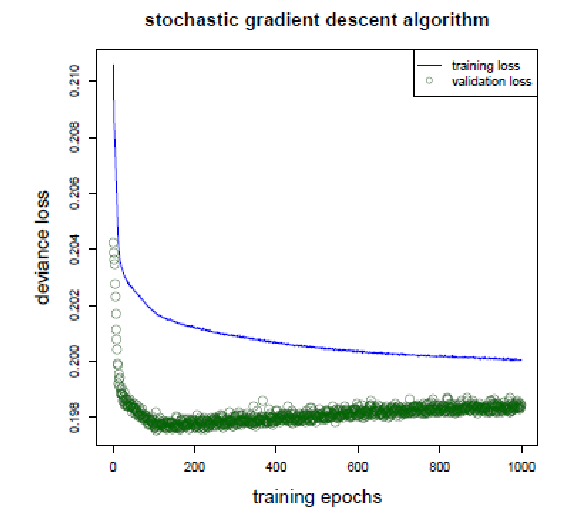
We have 7 continuous features components and two categorical ones having 11 and 22 labels, respectively. Using embedding dimensions 2 for the two categorical variables provides us with a network architecture having a network parameter of dimension *r* = 792; this includes the 66 embedding weights of the two categorical feature components.

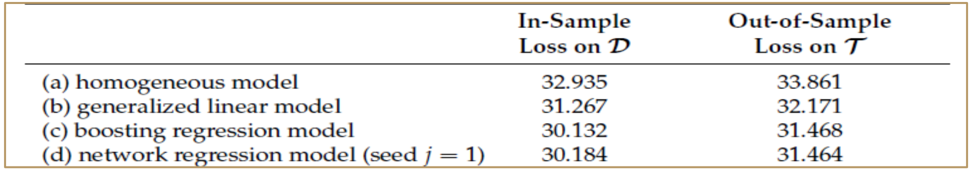
As activation function we choose the hyperbolic tangent. We implement this in R using the Keras library.

We choose the Poisson deviance loss function as objective function, and we use the nadam version of gradient descent.

Fitting the network

- Figure: one run of nadam gradient descent algorithm over 1000 epochs on random mini-batches of size 5000
- Retrieve the network parameter that has the smallest loss on V – stopping rule in place
- To prevent overfitting, training stopped early once network overfits to validation set
- Early stopping => that this network has a bias w.r.t. the learning data D.
- Applied bias regularization step proposed in Wüthrich (2019)

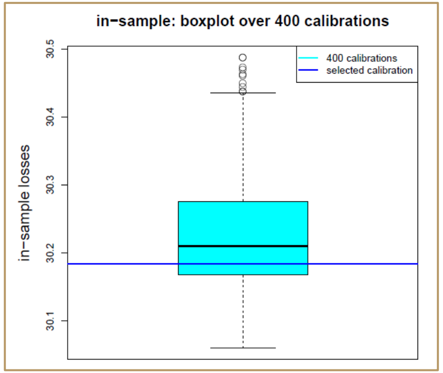


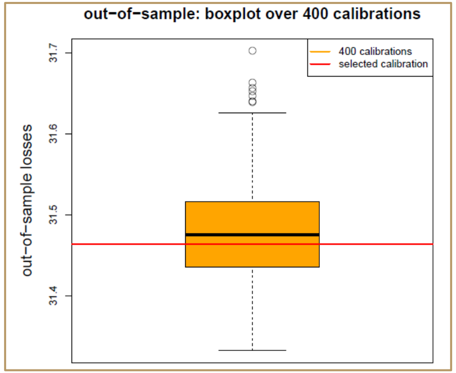


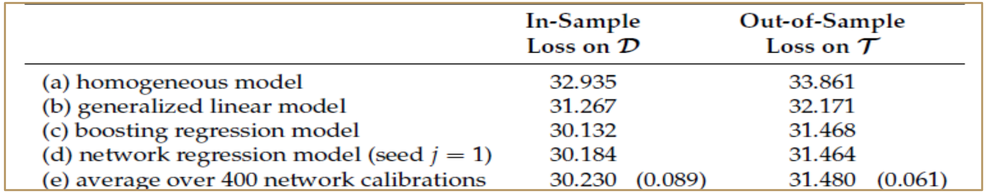
	In-Sample Loss on $\mathcal{D}$	Out-of-Sample Loss on $\mathcal{T}$
(a) homogeneous model	32.935	33.861
(b) generalized linear model	31.267	32.171
(c) boosting regression model	30.132	31.468
(d) network regression model (seed $j = 1$)	30.184	31.464

Variability of results

- Randomly split the learning data into training/validation
- Randomly split into mini-batches of size 5000
- Randomly choose the starting point of the gradient descent algorithm



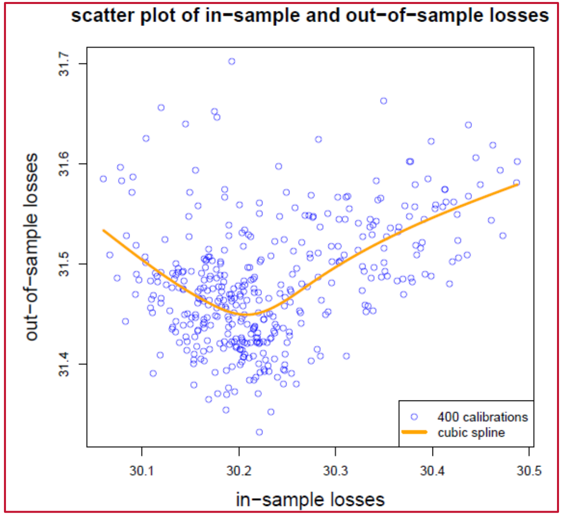




	In-Sample Loss on $\mathcal{D}$	Out-of-Sample Loss on $\mathcal{T}$
(a) homogeneous model	32.935	33.861
(b) generalized linear model	31.267	32.171
(c) boosting regression model	30.132	31.468
(d) network regression model (seed $j = 1$)	30.184	31.464
(e) average over 400 network calibrations	30.230 (0.089)	31.480 (0.061)

Can we predict out of sample performance?

- Scatter plot shows in-sample and out-of-sample losses over the 400 different runs of the gradient descent fitting (plus a natural cubic spline):
  - Small in-sample losses imply overfitting
  - Large in-sample losses imply calibrated model not optimal
  - Large variation even at most optimal in-sample loss



The Nagging Predictor

Calculate the nagging predictors $\bar{\bar{\mu}}_t^{(M)}$ over the test data set T .
Figure shows for M ≥ 1 the sequence of out-of-sample losses:

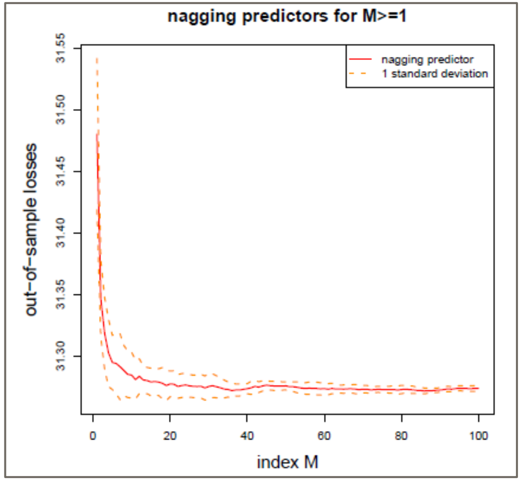
$$\mathcal{L}(\mathcal{T};\bar{\bar{\mu}}^{(M)}_{t=1,\ldots,m}) = \frac{1}{m}\sum_{t=1}^{m}\delta(Y_t^\dagger,\bar{\bar{\mu}}_t^{(M)}),$$

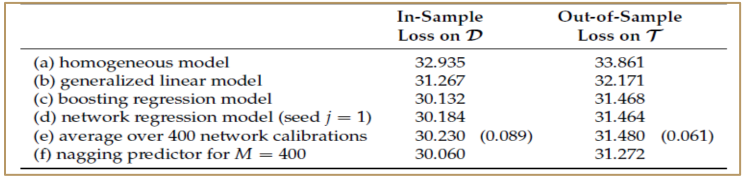
Nagging leads to substantial improvement in out-of-sample losses => nagging helps to improve the predictive model substantially.

Convergence takes place over first 20 aggregating steps in our example.

Dotted orange lines in give corresponding 1 standard deviation confidence bounds.

Sufficiently small confidence bounds after averaging over roughly 40 network calibrations.



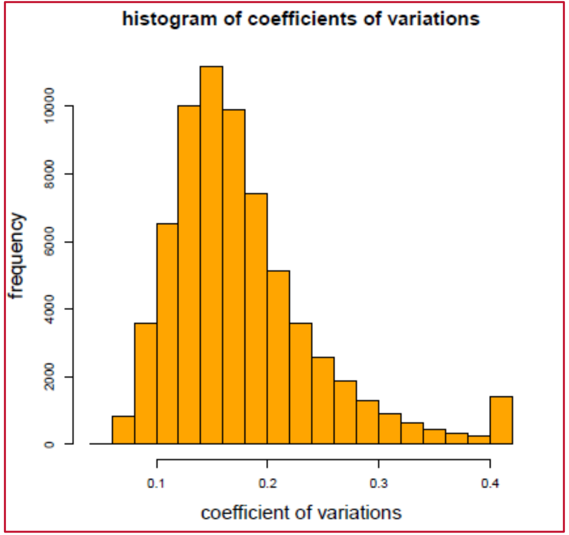


	In-Sample Loss on $\mathcal{D}$		Out-of-Sample Loss on $\mathcal{T}$	
(a) homogeneous model	32.935		33.861	
(b) generalized linear model	31.267		32.171	
(c) boosting regression model	30.132		31.468	
(d) network regression model (seed $j = 1$)	30.184		31.464	
(e) average over 400 network calibrations	30.230	(0.089)	31.480	(0.061)
(f) nagging predictor for $M = 400$	30.060		31.272	

Pricing of Individual Insurance Policies

- Must ensure robustness of prices on an individual insurance policy level.
- Expect need to average over more networks than portfolio level because the former statement includes an average over all policies
- We calculate for each policy t = 1, . . . ,M of the test data T, the nagging predictor $\bar{\bar{\mu}}_t^{(M)}$ based on over M = 400 different networks and we calculate the <u>empirical coefficients of variation</u> in the individual network predictors given by:

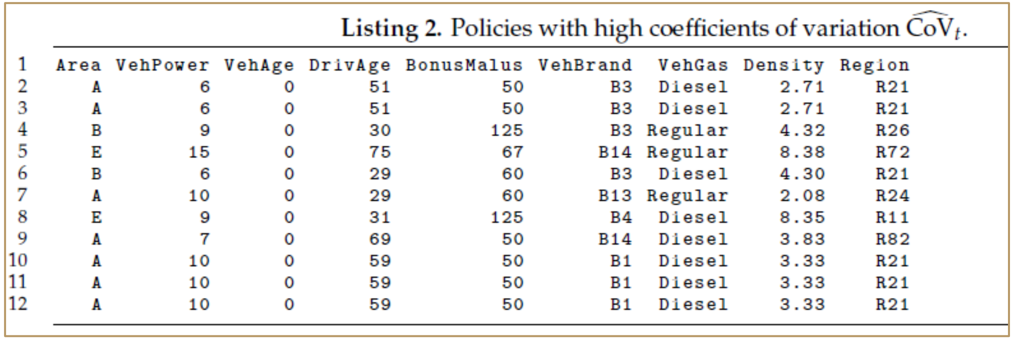
$$\widehat{\mathrm{CoV}}_t = \frac{\widehat{\sigma}_t}{\bar{\bar{\mu}}_t^{(M)}} = \frac{\sqrt{\frac{1}{M-1}\sum_{j=1}^{M}\left(\widehat{\mu}_t^{(j)} - \bar{\bar{\mu}}_t^{(M)}\right)^2}}{\bar{\bar{\mu}}_t^{(M)}},$$



- Most policies (73%) have a CoV of less than 0.2.
- 11 of the m = 67, 801policies have a CoV bigger than 1.
- For CoV ~- 1, and averaging over 400 different network calibrations we still have an uncertainty of $1/\sqrt{400}$ ~= 5% to 10%
- Need to aggregate over a considerable number of networks to receive stable network regression prices

Focus on Observations with CoV > 1

We list the 11 policies in the table below:



Listing 2. Policies with high coefficients of variation $\widehat{\text{CoV}}_t$.

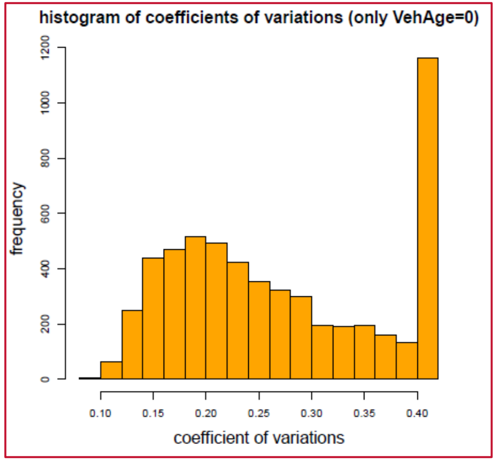
```
Area VehPower VehAge DrivAge BonusMalus VehBrand  VehGas Density Region
  A         6      0      51         50       B3  Diesel    2.71    R21
  A         6      0      51         50       B3  Diesel    2.71    R21
  B         9      0      30        125       B3 Regular    4.32    R26
  E        15      0      75         67      B14 Regular    8.38    R72
  B         6      0      29         60       B3  Diesel    4.30    R21
  A        10      0      29         60      B13 Regular    2.08    R24
  E         9      0      31        125       B4  Diesel    8.35    R11
  A         7      0      69         50      B14  Diesel    3.83    R82
  A        10      0      59         50       B1  Diesel    3.33    R21
  A        10      0      59         50       B1  Diesel    3.33    R21
  A        10      0      59         50       B1  Diesel    3.33    R21
```

All these policies have vehicle age **VehAge** = 0.

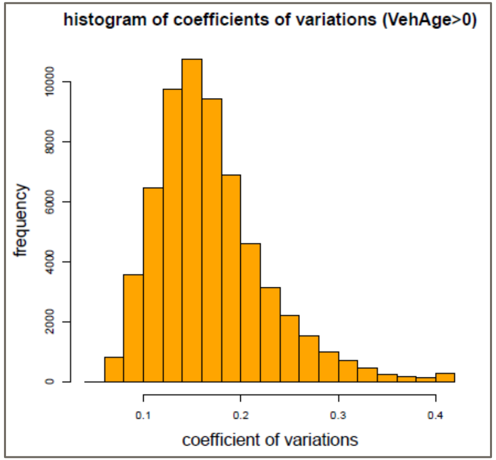
We will proceed to analyse policies with **VehAge** = 0 and **VehAge** > 0 separately.

Uncertainty in VehAge = 0

VehAge = 0 :



VehAge > 0 :



Confirm that mainly policies with **VehAge** = 0 are difficult to price. These could be rental cars (or some other special cases). Unfortunately, no further information is available for this data set that allows such analysis.

CoV of the nagging predictor is a useful data-driven tool for segmenting data and understanding network predictions

Meta Network Regression Model

- Nagging predictor substantially improves the predictive model

- Difficulty is that it involves aggregating over M = 400 predictors for each policy i

- Propose to build a meta model that fits a new network to the nagging predictors $\bar{\bar{\mu}}_i^{(M)}$, i = 1,…, M – "model distillation"

- Comparably simple to fit network to smooth surface described by nagging predictors $\bar{\bar{\mu}}_i^{(M)}$, i = 1,…, M , and over-fitting will not be an issue

Building the Meta Model

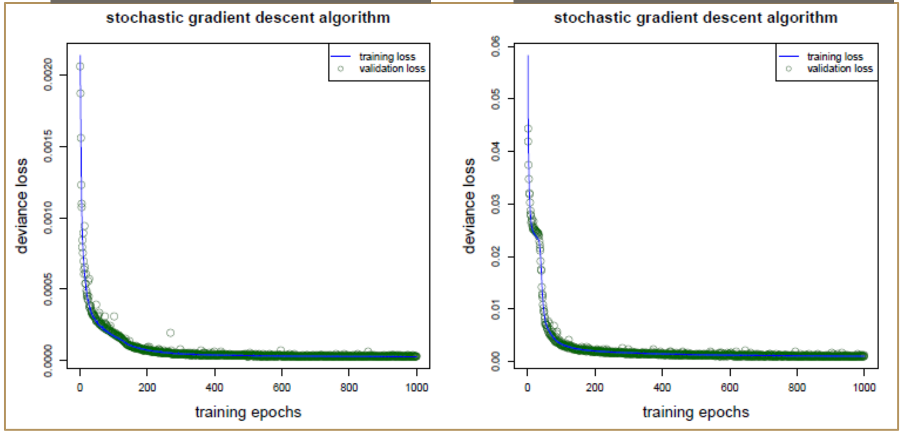
Use the same network architecture to build the meta model - change the loss function and the response variables.

Replace the original claim count responses Y_i by the $\bar{\bar{\mu}}_i^{(M)}$, and for the loss function we choose the square loss function – can choose an unweighted function or can weight the individual observations with $1/\hat{\sigma}_i$.

We conclude that the weighted version has better convergence properties in gradient descent fitting

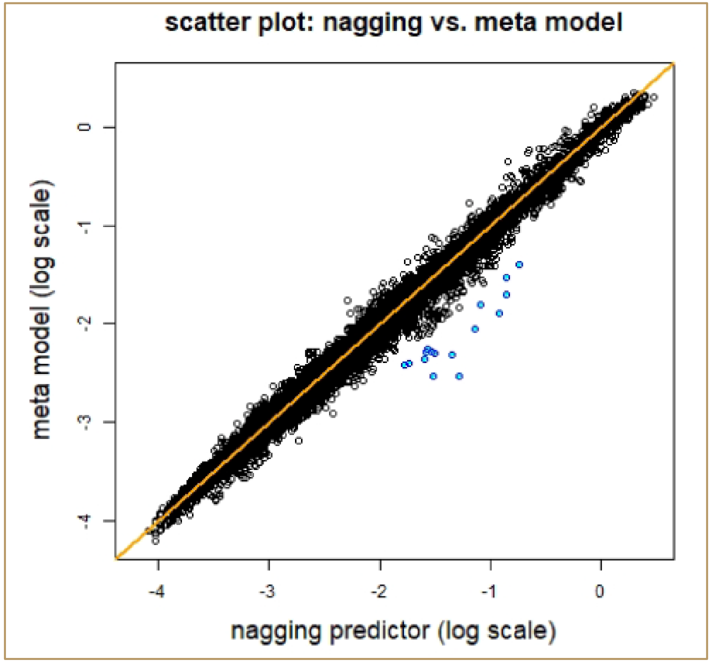
Unweighted square loss function

Weighted square loss function



Nagging Predictor vs Meta Model Predictor

The scatterplot below presents the two predictors:

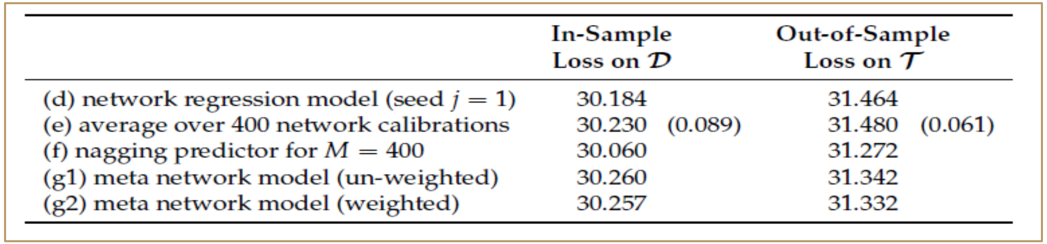


The models are reasonably equal with the biggest differences highlighted in blue.

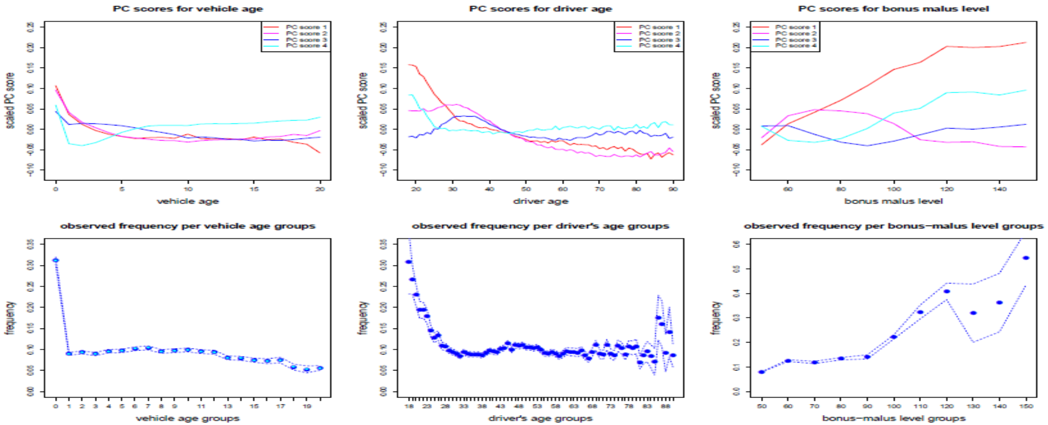
These refer to the policies with vehicle age 0 – the feature component within the data that is the most difficult to fit with the network model.

Optimal Model

The resulting in-sample and out-of-sample losses are in the table below:



	In-Sample Loss on $\mathcal{D}$		Out-of-Sample Loss on $\mathcal{T}$	
(d) network regression model (seed $j = 1$)	30.184		31.464	
(e) average over 400 network calibrations	30.230	(0.089)	31.480	(0.061)
(f) nagging predictor for $M = 400$	30.060		31.272	
(g1) meta network model (un-weighted)	30.260		31.342	
(g2) meta network model (weighted)	30.257		31.332	



- The weighted version (g2) has a better loss performance than the unweighted version.
- It is slightly worse than the nagging predictor model, however substantially better than the individual network models and easier in handling than the nagging predictor.
- Plotted PCA analysis of the learned representation in last layer of model, averaged for each covariate value.
- First PC follows empirical frequencies, other PCs reflect refinements and interaction effects.

Conclusions on Nagging Predictors

Produces accurate and stable portfolio predictions on the basis of random network calibrations, and has provided convergence results in the context of Tweedie's compound Poisson GLM's.

Shown that stable portfolio results are achieved after 20 network training runs. Achieved at the policy level by increasing the network training runs to 400 – important requirement for the use of networks for insurance pricing and more general actuarial tasks.

CoV of the nagging predictor is a useful data-driven metric for measuring the relative difficulty with which a network is able to fit to individual training examples – used to calibrate an accurate meta network which approximates the nagging predictor.

Another important aspect of consistency within insurance is stable pricing over time. Future work could consider methods for stabilizing network predictions as new information becomes available.

Questions

Thank you