Credibility in network regression with σ-hot encoding and weight balancing

Anne van der Scheer

Perunum
ACTUARIEEL ADVIES

28-02-2024

Outline

- New chances for credibility
- 1-hot encoding
- σ-hot encoding
- Weight balancing
- Initialization and weight regularization
- Case I: claim severity (simulation), Avanzi et al. 2023
- Case II: longterm disability (simulation)
- Conclusions

New chances for credibility

Observation: classical GLM needed the extension to GLMM for handling credibility of sparse[1] categories. In network regression we also find the distinction between fixed and randomized categoricals, the latter with specific extensions and under restrictive assumptions.

Idea: general networks are already adequate as long as we create an equal level playing field for all categorical and continuous variables. Then they can fairly compete for available complexity and credibility is naturally achieved.

Therefore we reconsider the encoding and handling of categorical variables.

[1] we prefer this term above high-cardinality

1-hot encoding

The standard transformation procedure of getting from unordered categories to numbers is 1-hot encoding. We repeat the definition of 1-hot encoding of one categorical variable with k levels to k binary variables:

> *for every* $v \in \{v_1, v_2, \ldots, v_k\}$ *the corresponding 1-hot encoded values* $w_1, w_2, \ldots, w_k \in \{0, 1\}$ *are defined by* $w_i = \mathbb{1}_{(v=v_i)}$.

Following the electrical metaphor, we will call 0 the cold-value and 1 the hot-value.

We will here assume that we do not use a reference variable and therefore use k instead of $k-1$ variables.

1-hot encoding

Issues with 1-hot encoding:

- sparse categories have their own "full" weights which can lead to overfitting
- in regularization (like L1 and L2 penalties) all weights are of equal importance

Therefore we conclude:

- hot-values and their weights seem to be interchangeable, but they are not
- hot-values should not be arbitarily chosen (i.e. always equal 1)

1-hot encoding

We propose some necessary requirements for a good hot-encoding:

1. the encoding only depends on the observed hot-fraction p, i.e. the number of hot-observations divided by the total number of observations
2. $\text{hot}(p) \to 0$ if $p \downarrow 0$
3. $\text{hot}(p) \to 0$ if $p \uparrow 1$

Remarks:

- Scaling the number of observations should not make a difference for hot-encoding, in analogy to standardization of continuous variables. Therefore the absolute number of observations should be irrelevant
- For very small and very large values of p the hot-values become insignificant, which lead to the limit-value of zero

1-hot encoding

We repeat standardizing continuous variables before they entry the network:

$$v \rightarrow \frac{v - \mu}{\sigma}$$

where μ is the mean and σ is the standard deviation[1] of the variable observations . Standardized variables become centered around zero with a standard deviation of 1. It is well known that standardization plays an important role in the learning capacity of the network.

Average values are close to zero, more extreme values are further from zero. All variables are on the same scale. We try to adopt these properties for categorical variables, so that they have optimal interaction with the continuous variables during training.

[1] population or sample standard deviation

1-hot encoding

In an attempt to equalize categorical and continuous variables, we start naively by standardizing the 1-hot encoding, using the exact formula of a continuous variable:

$$0 \to \frac{-p}{\sqrt{p(1-p)}}$$

$$1 \to \frac{1-p}{\sqrt{p(1-p)}}$$

where p is again the fraction of hot-values in the observations.

Remark: the resulting values are independent of the specific choice of 0 and 1, as long as cold-value $<$ hot-value.

1-hot encoding

We get the following results:

- $p \approx 0$: standardized hot-value goes to infinity, standardized cold-value goes to 0.
- $p \approx 1$: standardized hot-value goes to 0, standardized cold-value goes to -infinity.

This does not yet meet the properties for good hot-encoding. To proceed we multiply the standardized outcome by the relative frequency of the cold- en hot-values ($1-p$ resp. p) to get

$$\text{cold-value} \rightarrow (1-p) \cdot \frac{-p}{\sqrt{p(1-p)}} = -\sqrt{p(1-p)} = -\sigma$$

$$\text{hot-value} \rightarrow p \cdot \frac{1-p}{\sqrt{p(1-p)}} = \sqrt{p(1-p)} = \sigma$$

1-hot encoding

Instead of the 1-hot encoding this leads to a signed σ-coding, where σ is the standard deviation of the observed values, independent of the original cold- and hot-values.

Remarks:

- Notice that the new encoding satisfies our earlier requirement for sparse ($p \approx 0$) and dominant ($p \approx 1$) categories. In both cases σ gets close to zero
- σ gets a maximal value of 0.5 when $p = 0.5$
- For categorical variables with only two categories we get for both categories the same encoding
- The last step in the encoding destroyed the balance of the hot-encoded variables. We have to repair this

σ-hot encoding

To make the following easier, we set the cold-value on zero (instead of -σ) and leave the hot-value unchanged:

$$\text{cold-value} \rightarrow 0$$
$$\text{hot-value} \rightarrow \sigma$$

We will call this end result the σ-hot encoding.

Just like the continuous variables we want a balanced input around zero for the categorical variable. To be precise, we want to ensure that the sum of the weighted inputs equals zero for every categorical variable. At initialization and during the whole training.

σ-hot encoding

Balancing the sum of the weighted inputs for a categorical variable starts with a proper initialization. The standard in literature is a random initialisation of all weights. Biases are often initialised as zero. We will do it the other way around:

- all weights are initialised as zero
- all biases in a layer are deterministically initialised around zero, thereby avoiding symmetry

With zero weights we start with a balanced sum of each categorical variable at each entry of neurons in the first hidden layer. After each update of the weights we compute for each neuron j in the first hidden layer the inbalance for every categorical variable.

σ-hot encoding

Let $W_{ji}^{(1)}$ for $i = 1, \ldots, k$ be the weights of the σ-hot encoded variables of some categorical variable, let n_i be the corresponding numbers of hot-encodings and let N be the total number of observations. Then we get

$$\text{inbalance}_j = \sum_{i=1}^{k} W_{ji}^{(1)} \cdot \sigma_i \cdot n_i$$

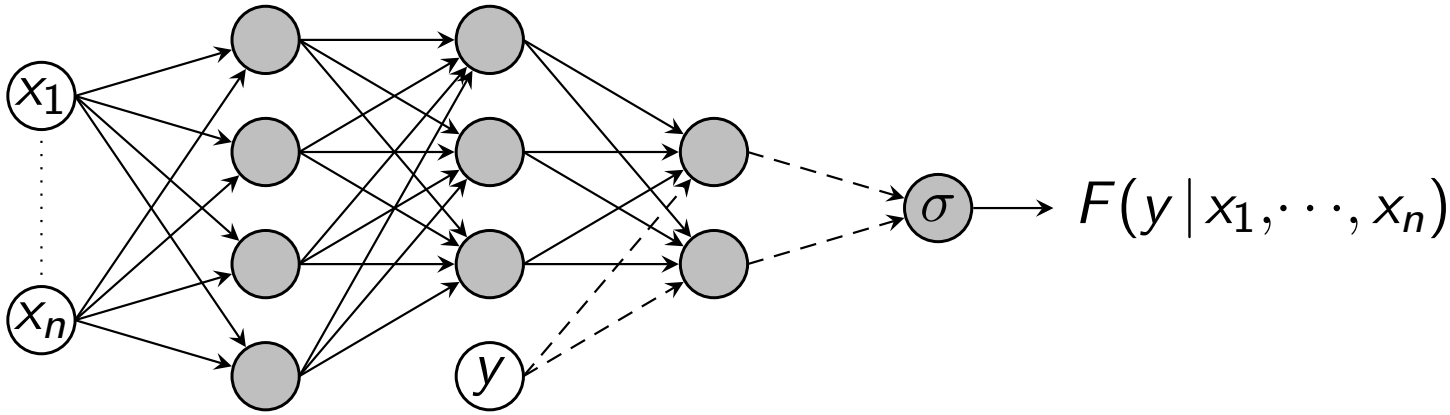
To restore balance again, we update the bias b_j and the weights of the receiving neuron j:

$$b_j^* = b_j + \frac{\text{inbalance}_j}{N}$$

$$W_{ji}^{(1)^*} = W_{ji}^{(1)} - \frac{\text{inbalance}_j}{\sigma_i \cdot N}$$

σ-hot encoding

The weight balancing ensures that we have a balanced input, the σ-hot values ensures a proper magnitude
And weight regularization can be stricter to help us. We choose a uniform bound on all individual weights in the network.
With this conclusion we can go further and implement this.



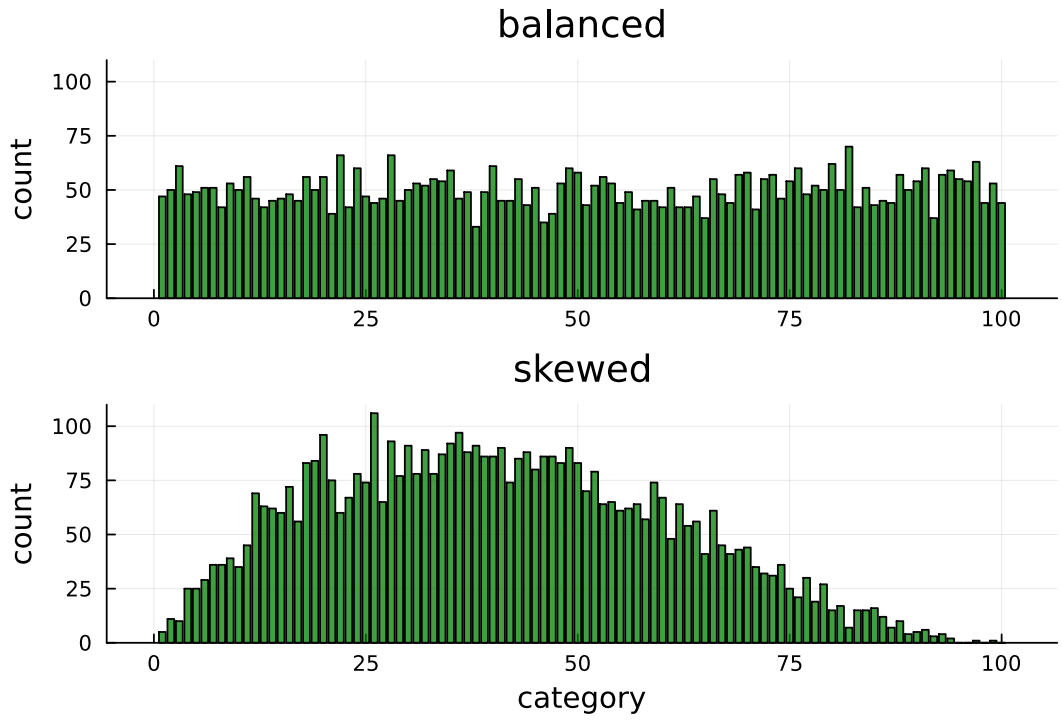
But we even get more: For very small categories we will get a weighted input that becomes almost zero. Knowing that the network itself responds continuously, we could set the input zero when we don't know the categorie, or if we want an indifferent response of the network. For unseen categories in the network there

And is the σ-hot the only right encoding? No. We could replace with another function, for example the squared version of variance or 2 times σ. The network should be capable of transitioning different functions to the right one.
We have to set the hyperparameter c and the size of the network. These hyperparameters somehow include the influence of N in the network. When the σ-value is low for a categorie, it may still be the case that the number of observations are large and there may be statistical significance for an own estimation or even an seperate model: The Netherlands population is a small part of the world, but large enough to have own mortality tables.
It is good to reflect on this for a good understanding.
And there is work to do: The σ-hot encoding is easy. In deployment of the network we could even switch back to 1-hot by multiplying the input weights with the σ-hot values.
The balancing on the level of categorical variable is not a standard implementation, neither is the uniform bound on weights. And there is another issue: weight initialization. We will come to that later.

Weight initialization is generally done with randomization. Several algorithms exist. The biases are generally set on zero.
We do it the other way around: weights are intialized by zero. This implies that we have balanced input for categories from the beginning!
How to do the biases?
We use a uniform bound on weights, which guarantees that the network-function will never blow up. This can easily lead to overfitted likelihood for small number (or even one) observations, comparable with the classical case for mixed normal distributions. Therefore, we don't need early stopping. We don't choose big networks, always start small. We don't use embedding of categoricals, we think that σ-hot solves it. Especially for small networks (and therefore a small first layer of neurons).
It feels like the σ-hot encoding leads to less randomization, even for initialization. It encourages small networks, perhaps even using full batches. It feels like going back to the plain and simple network and focus on the magical and bad understood power. Finally, it feels like shrinking the computations and increasing the hope of not getting stuck in local optima.

σ-hot encoding

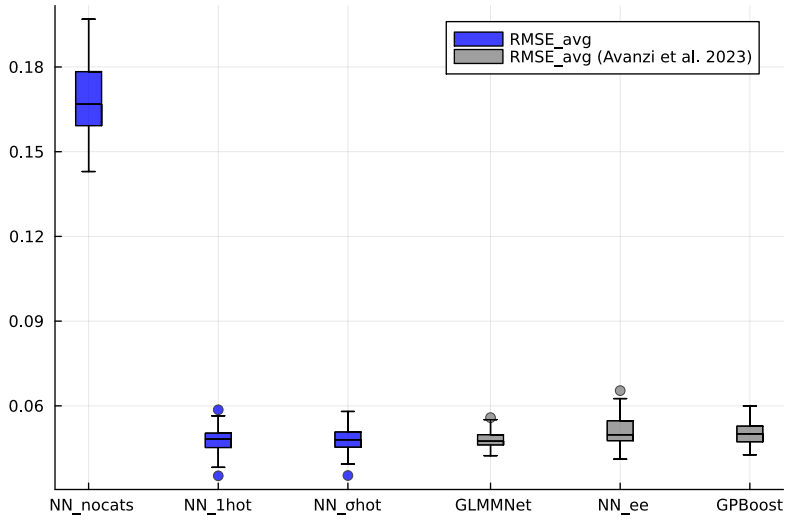
number of observations by category



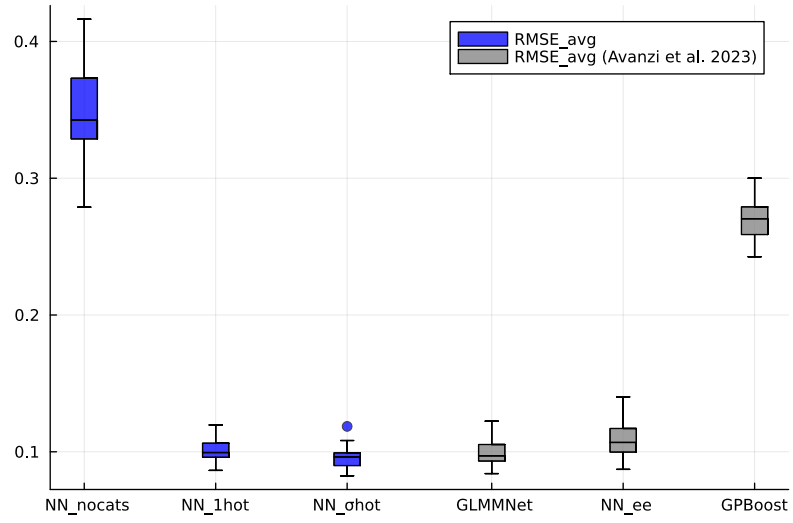
simulations Friedman function

We start with the two most important experiments

statistics experiment 1



statistics experiment 2

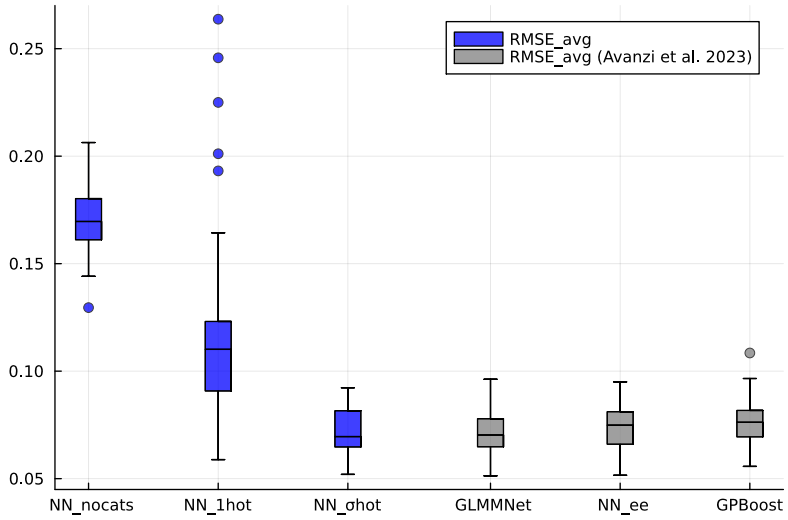


We see that these are indeed the most important experiments.

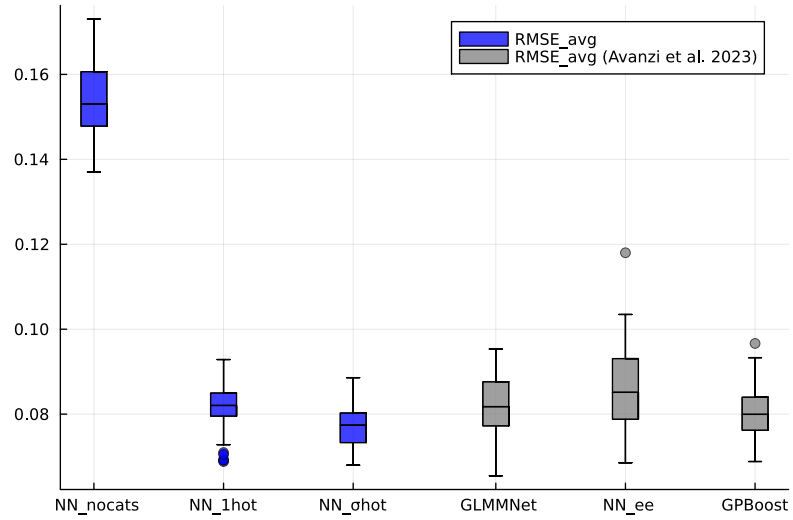
simulations Friedman function

We start with the two most important experiments

statistics experiment 3



statistics experiment 4

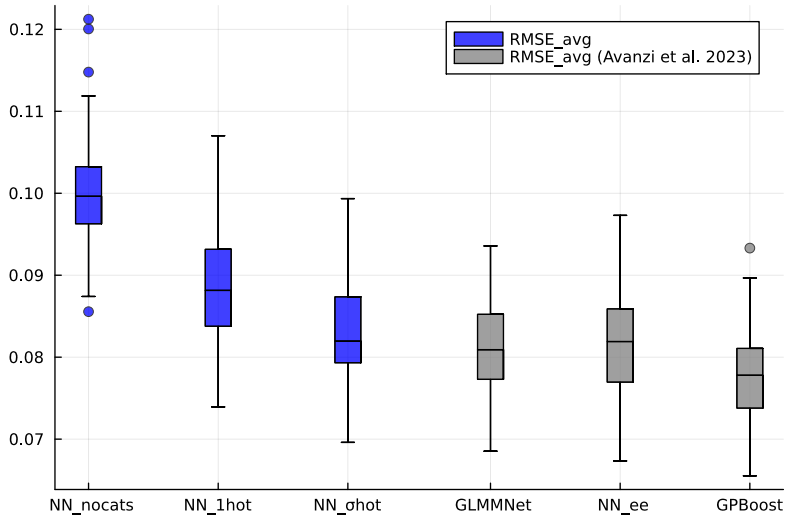


We see that these are indeed the most important experiments.

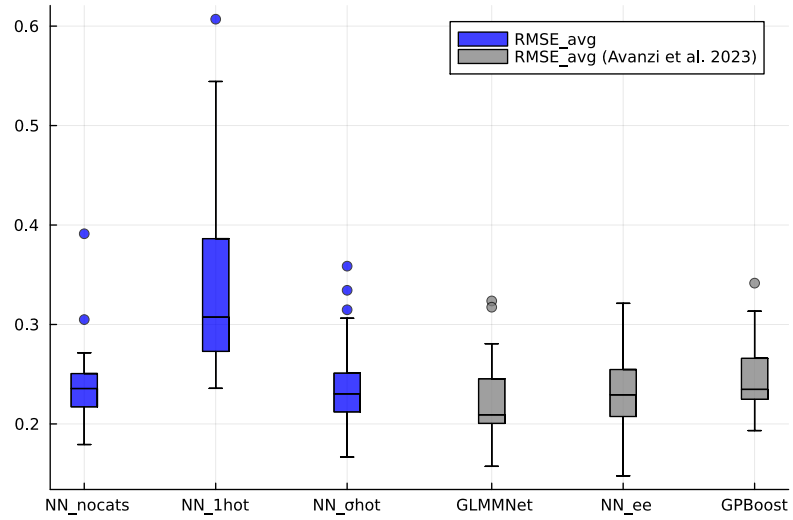
simulations Friedman function

We start with the two most important experiments

statistics experiment 5



statistics experiment 6

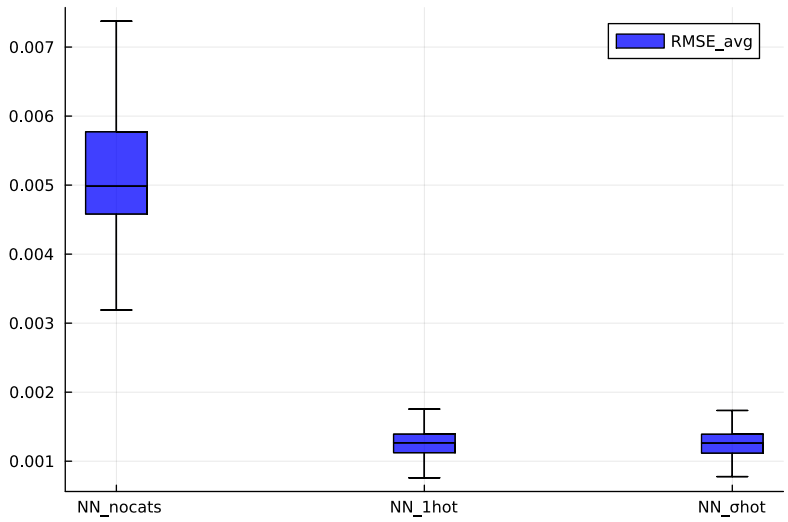


We see that these are indeed the most important experiments.

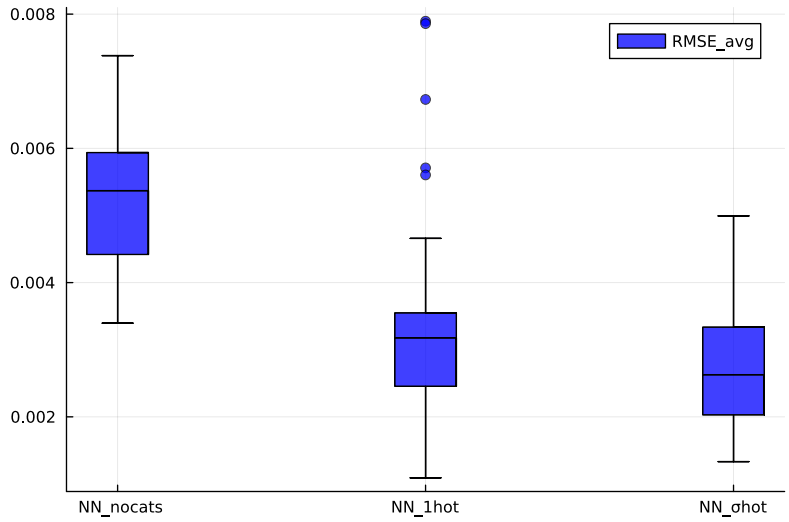
simulations disability insurance

We start with the two most important experiments

statistics experiment 1



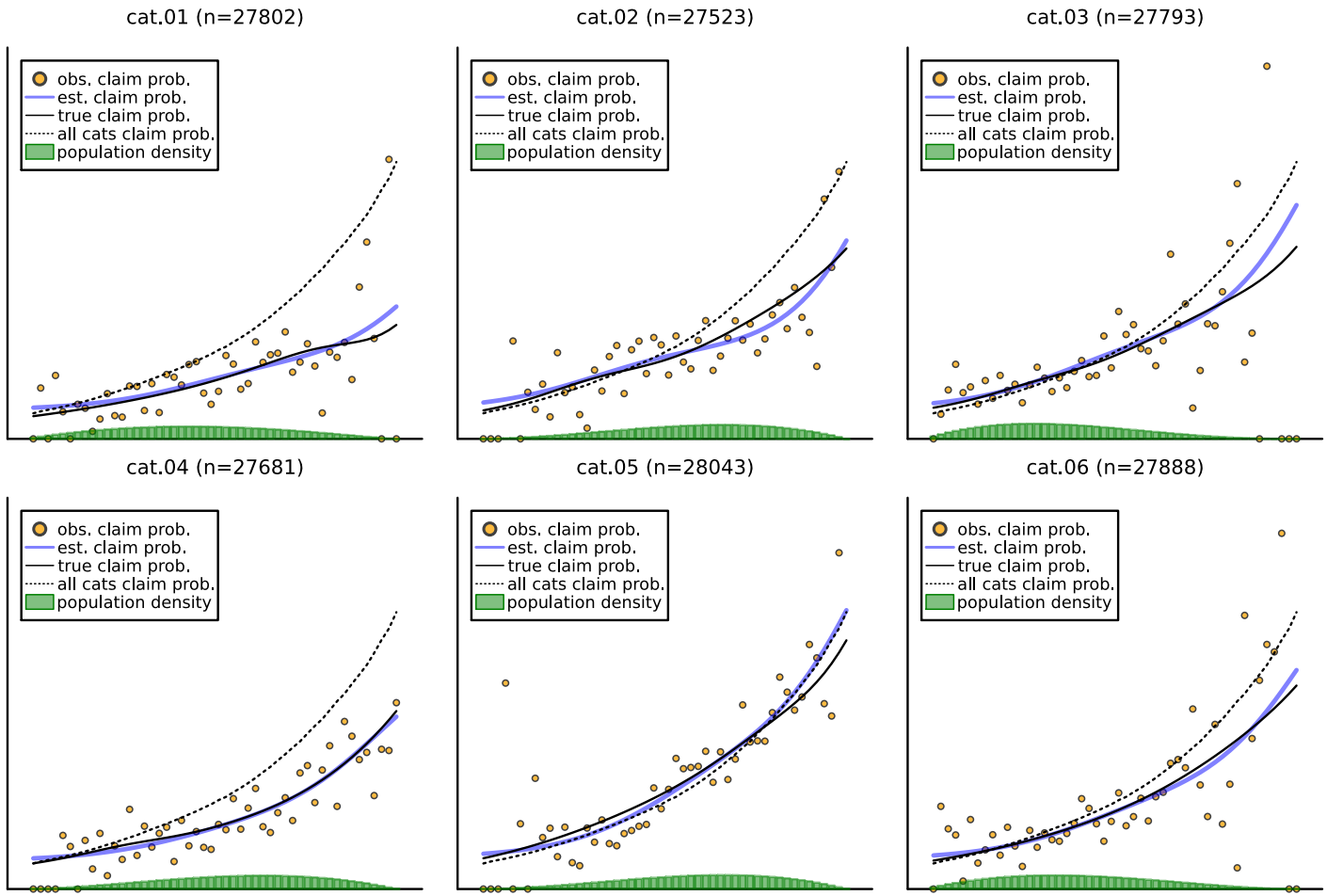
statistics experiment 2



We see that these are indeed the most important experiments.

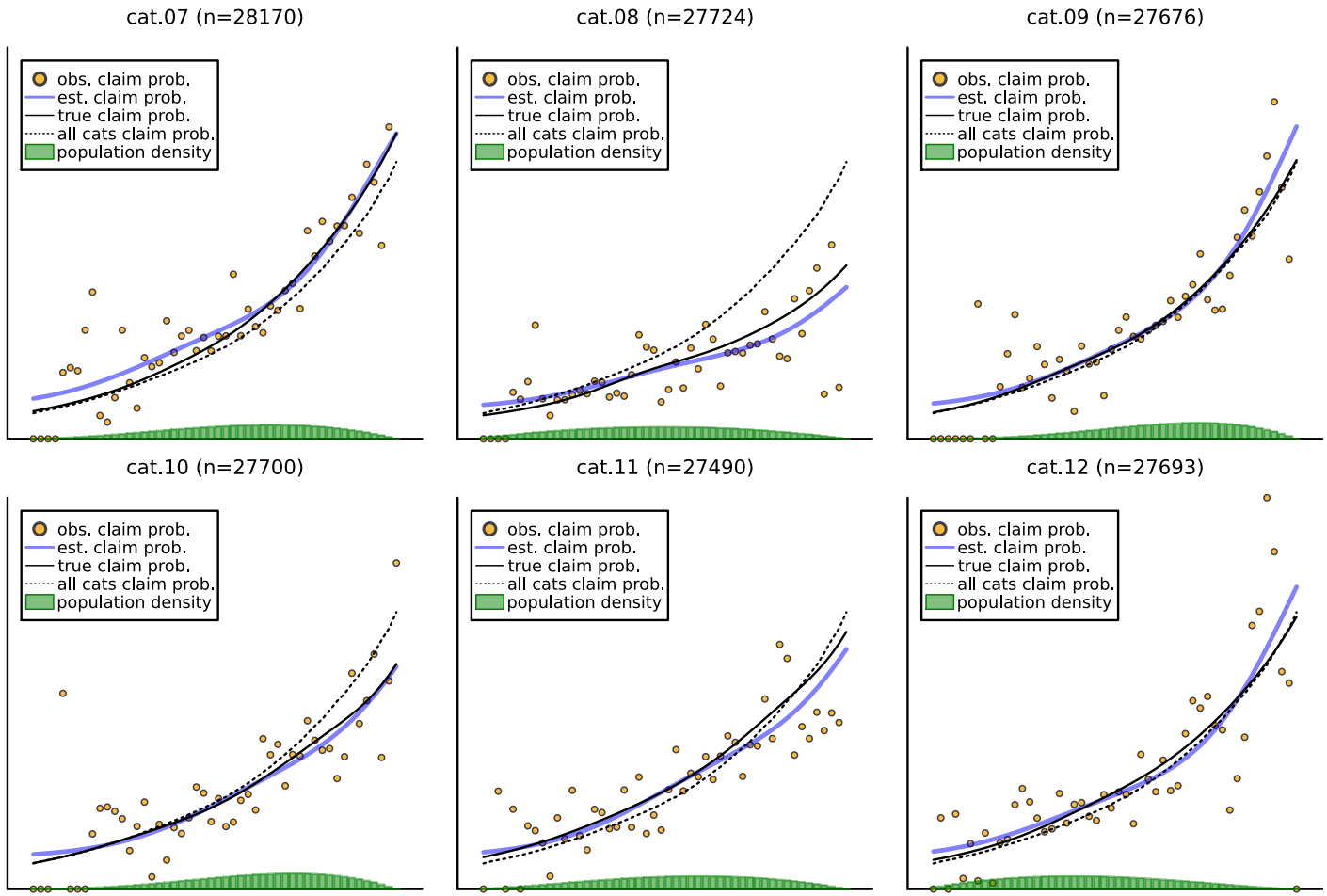
simulations disability insurance

results experiment 1, cats 01 t/m 06



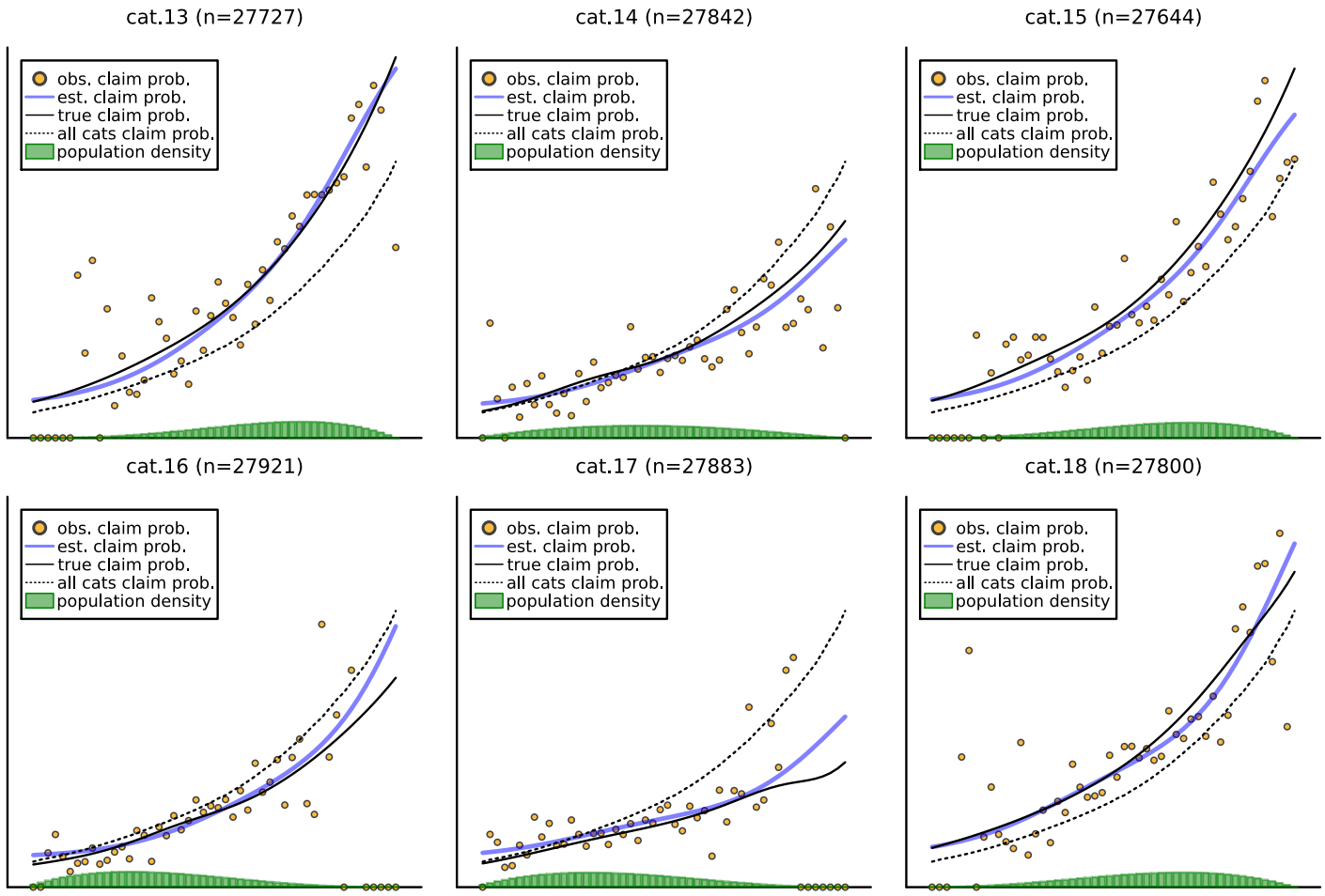
simulations disability insurance

results experiment 1, cats 07 t/m 12



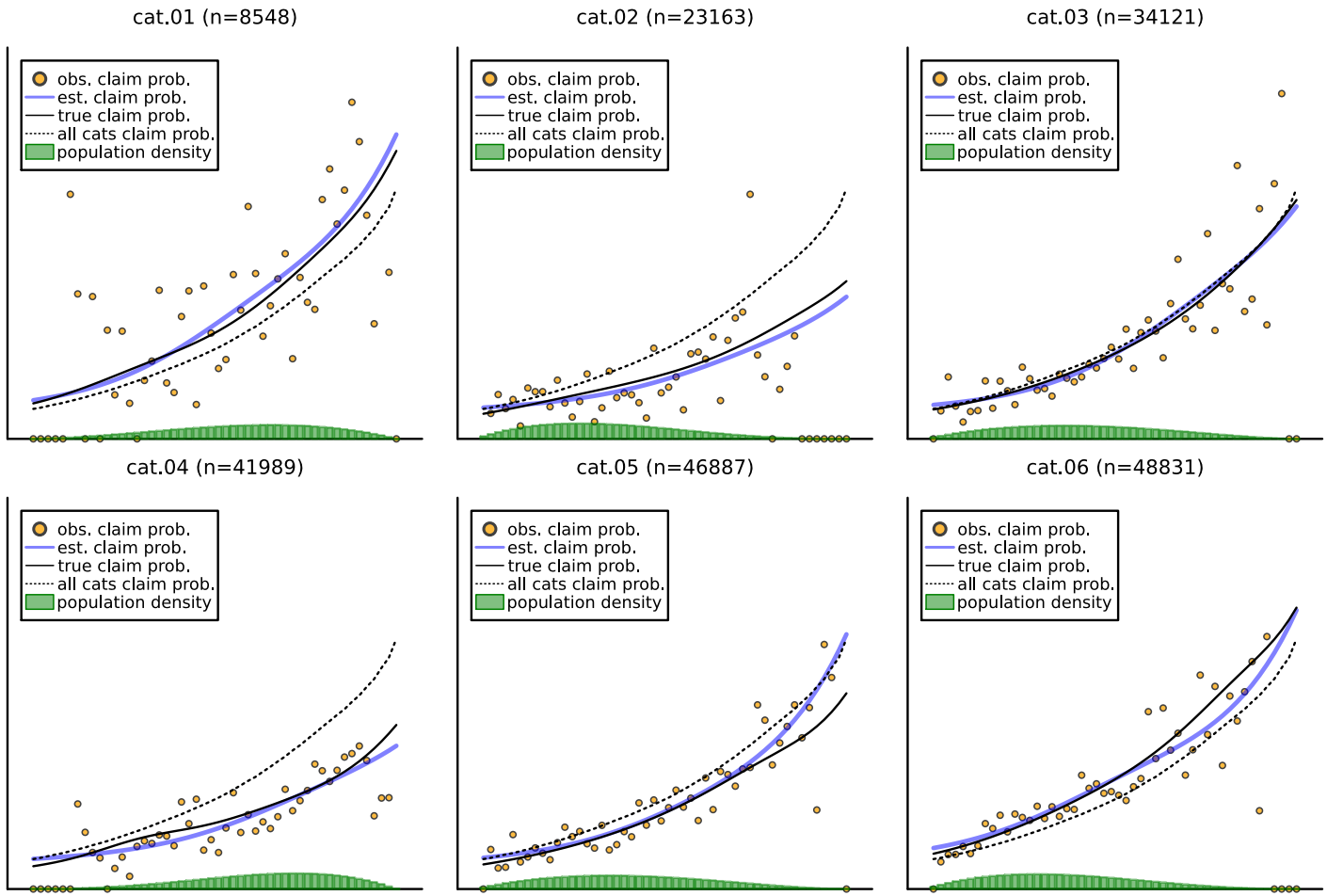
simulations disability insurance

results experiment 1, cats 13 t/m 18



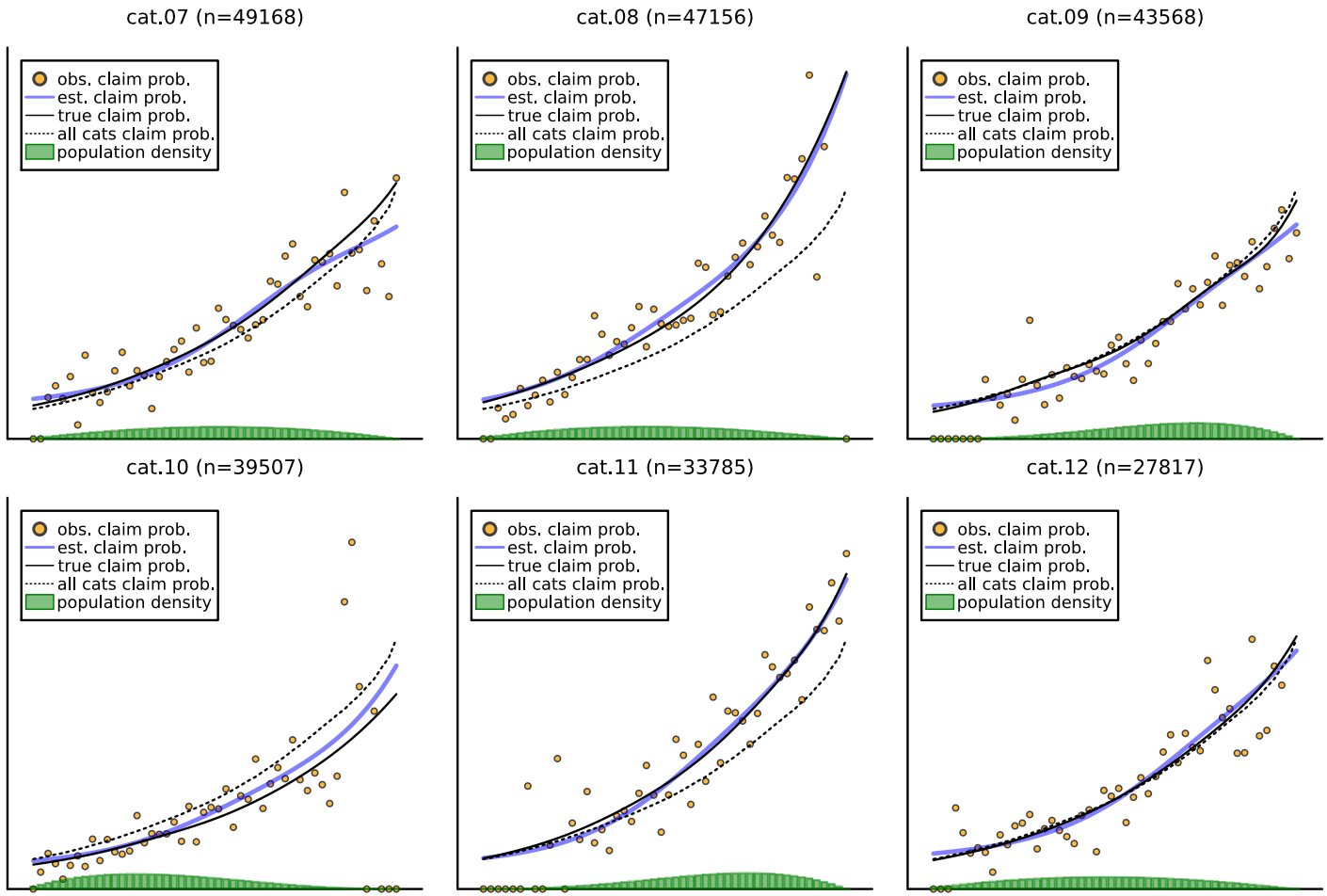
simulations disability insurance

results experiment 2, cats 01 t/m 06



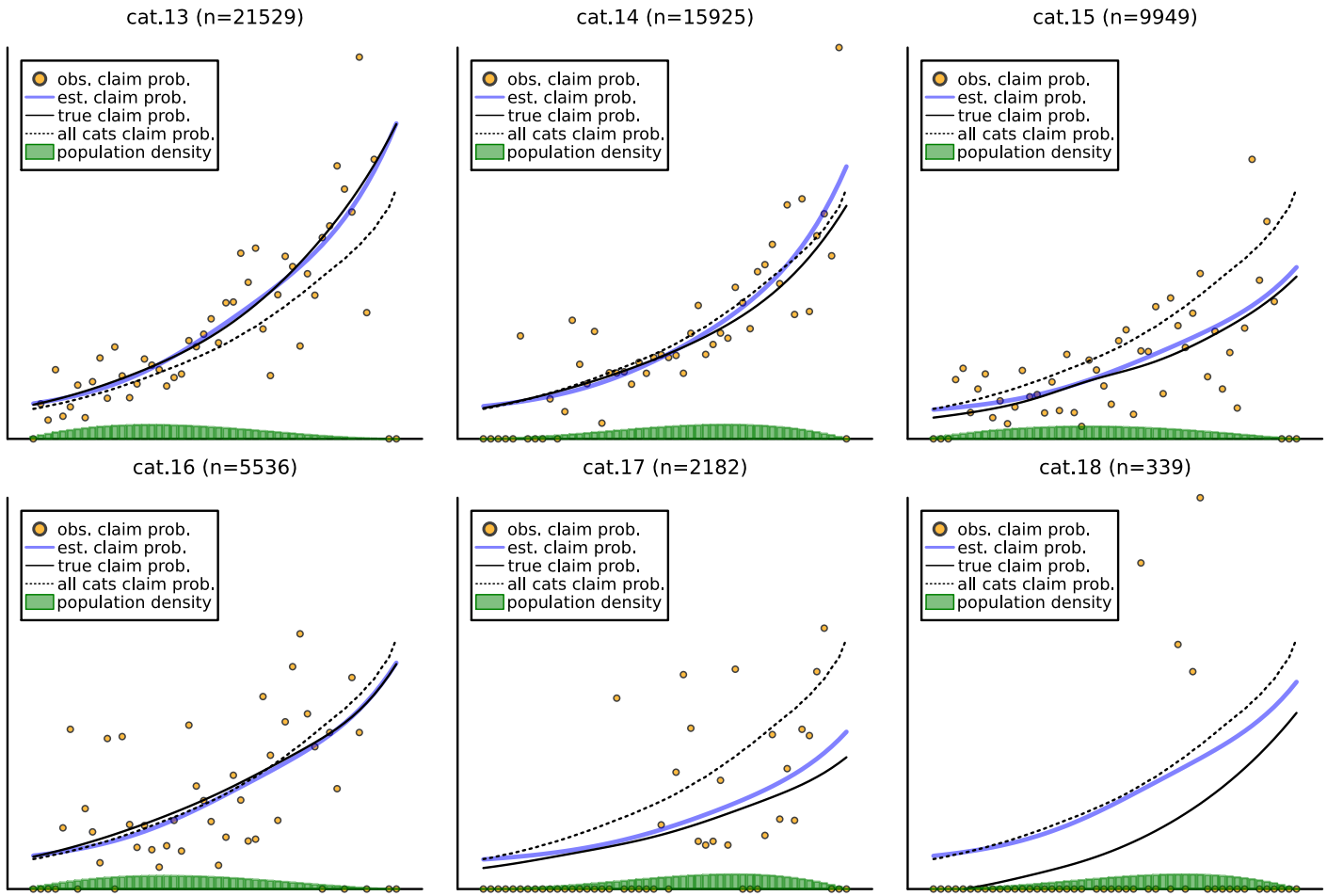
simulations disability insurance

results experiment 2, cats 07 t/m 12



simulations disability insurance

results experiment 2, cats 13 t/m 18



simulations disability insurance

Motivatie (2)

3. Geen hypothesevorming en -toetsing
   - tijdbesparing in modellering
   - generieke kansverdelingen en correlaties
   - modellen ook inzetbaar als benchmark
4. Van formules naar data
   - nadruk op genereren data in plaats van formularium
   - inzichtelijk voor brede groep analisten
   - bruikbaar met bestaande analyse tools

Cumulatieve verdeling F schatten is uitgangspunt

Een univariate continue cumulatieve verdeling F heeft een eenvoudige karakterisering en bruikbare eigenschappen:

- F is niet-dalend op het gehele bereik en heeft linkerlimiet 0 en rechterlimiet 1
- Als waarden van F expliciet berekend kunnen worden, dan zijn vanwege het niet-dalen ook inverse waarden F^{-1} te berekenen via een zoekalgoritme
- Sampling uit de verdeling kan dan eenvoudig via $F^{-1}(\text{rand})$, waarbij rand een random getal is uit het interval (0,1)

We kiezen bij het generiek samenstellen van continue verdelingen daarom voor de cumulatieve verdeling in plaats van de dichtheid.

Berekenen van verwachting uit F^{-1}

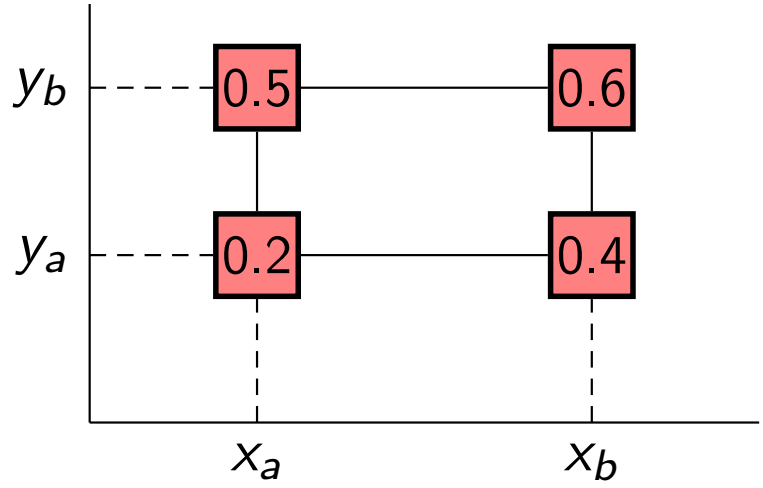
- Met randomisatie kan de verwachting worden geschat met behulp van $F^{-1}(\text{rand})$ door het gemiddelde te berekenen van een niet te kleine sample
- Dit kan echter ook zonder random getallen, door een fijnmazige vaste set van getallen tussen 0 en 1 te gebruiken
- De juistheid van deze benadering blijkt uit

$$\int_{-\infty}^{\infty} x \cdot f(x)\, dx = \int_{0}^{1} F^{-1}(y)\, dy = \lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} F^{-1}\left(\frac{k-\frac{1}{2}}{n}\right)$$

- Het berekenen van andere momenten voor bijvoorbeeld variantie en scheefheid gaat volgens dezelfde benadering

Uitbreiding naar multivariate verdeling (1)

Het generiek samenstellen van multivariate verdelingen is lastiger. Behalve dat $F(x, y)$ niet-dalend is in zowel x als y moet namelijk ook gelden dat $\frac{\partial^2 F}{\partial xy} \geq 0$. Voorbeeld waarbij aan de laatste eis niet wordt voldaan:



$P(x_a < x \leq x_b, y \leq y_b) = F(x_b, y_b) - F(x_a, y_b) = 0.6 - 0.5 = 0.1$
$P(x_a < x \leq x_b, y \leq y_a) = F(x_b, y_a) - F(x_a, y_a) = 0.4 - 0.2 = 0.2$
Dan $P(x_a < x \leq x_b, y_a < y \leq y_b) = 0.1 - 0.2 = -0.1 < 0$. Fout!

Uitbreiding naar multivariate verdeling (2)

Oplossing voor multivariate verdelingen met k variabelen is om deze te schrijven als het product van k onafhankelijke voorwaardelijke univariate verdelingen:

$$f(x_1, x_2, \cdots, x_k) =$$
$$f_1(x_1) \cdot f_2(x_2 \mid x_1) \cdot f_3(x_3 \mid x_1, x_2) \cdot \cdots \cdot f_k(x_k \mid x_1, x_2, \cdots, x_{k-1})$$

Vervolgens:

- schatten we k univariate voorwaardelijke cumulatieve verdelingen (zonder 'lastige' eisen multivariate verdeling)
- kunnen we random samples maken door forward sampling uit achtereenvolgens k univariate verdelingen in de volgorde $x_1, x_2, \cdots, x_k$

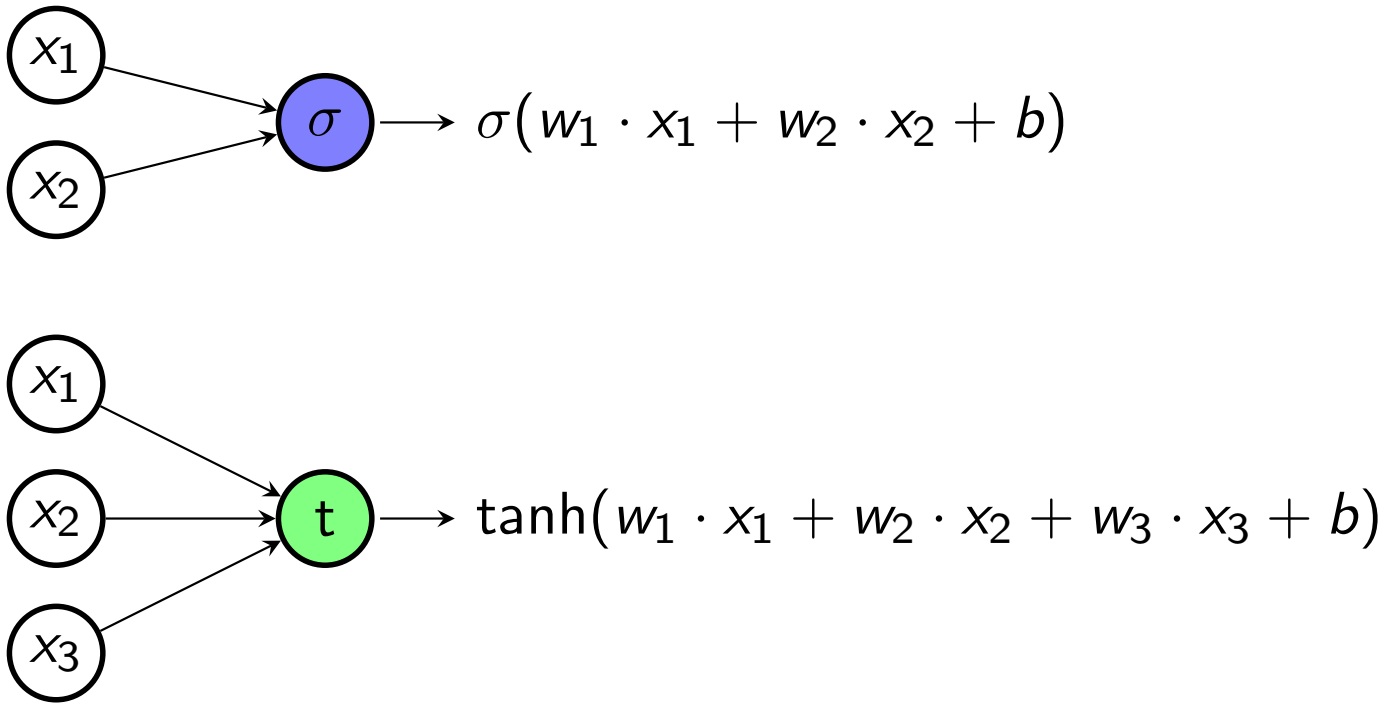
Algemene functie benadering

Voor het algemeen met een functie benaderen van een univariate voorwaardelijke cumulatieve verdeling heb ik twee richtingen overwogen:

1. Met een genetisch algoritme kunnen functies als een soort DNA worden samengesteld. In een 'populatie' vindt evolutie plaats via kruising en selectie. Het zoekgebied is echter enorm en het binnen een tijdsduur ontstaan van goede resultaten is onzeker. Deze outsider van Machine Learning viel uiteindelijk af
2. Met een neuraal netwerk ontstaat een gestructueerde functie met een enorm bereik. Een optimale oplossing is niet gegarandeerd, maar in de praktijk kunnen zeer goede resultaten worden behaald. Op deze underdog van statistical learning viel uiteindelijk de keuze

Het basiselement neuron

Een neuron transformeert input-waarden naar een output-waarde met behulp van gewichten (w_i), een bias (b) en een activatie functie. Als activatie functie gebruiken we de logistische (sigmoid, σ) functie en de tanh functie. Voorbeelden met twee en drie input-waarden:



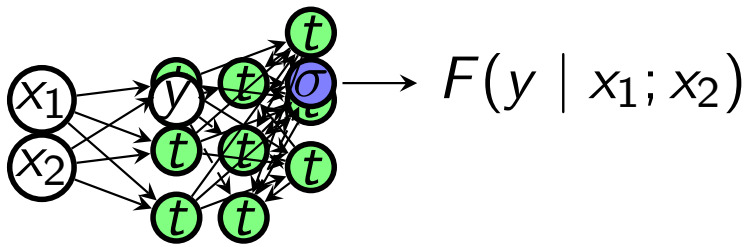
Activatie functies

Eigenschappen van de functies σ en tanh:

- De logistische functie σ wordt gedefinieerd door $\sigma(x) = \frac{1}{1+e^{-x}}$. Deze functie is strikt stijgend, puntsymmetrisch in $(0, \frac{1}{2})$ en heeft linkerlimiet 0 en rechterlimiet 1
- $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. De functies tanh en σ zijn lineaire transformaties van elkaar, waarbij $\tanh(x) = 2\sigma(2x) - 1$

Netwerk voor continue verdeling (1)

Een continue verdeling heeft de volgende (voorbeeld) structuur [1]:



De neuronen zijn gegroepeerd in lagen. Opeenvolgende lagen zijn volledig verbonden. Ze vormen een neuraal netwerk.

[1] zie 'Chilinski, Pawel and Silva, Ricardo, Neural Likelihoods via Cumulative Distribution Functions, arXiv preprint arXiv:1811.00974 (2020).'

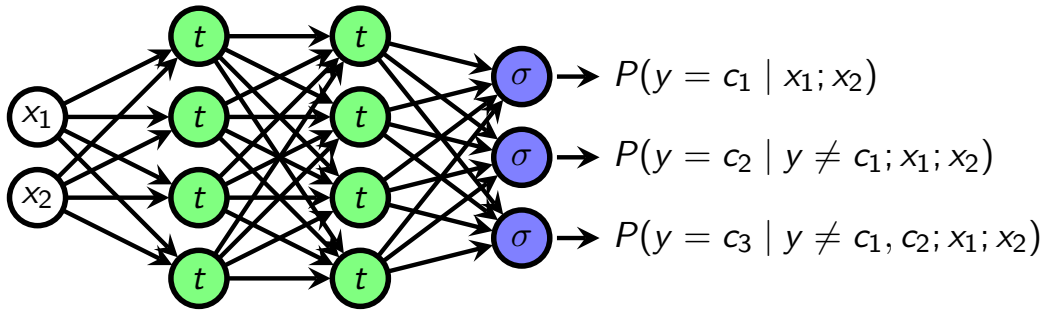
Netwerk voor continue verdeling (2)

Eigenschappen van het (voorbeeld) netwerk:

- De laatste laag bevat een σ-neuron, die waarden tussen 0 en 1 aanneemt. Voorgaande lagen bevatten uitsluitend tanh-neuronen, die symmetrisch zijn in hun waardebereik
- De linker twee lagen bevatten alleen relaties tussen de covariaten (x_1 en x_2), daarna wordt variabele y toegevoegd
- De gestreepte pijlen (ná y) hebben verplicht een gewicht ≥ 0, waardoor F niet-dalend is in y
- De geschatte F heeft geen linkerlimiet 0 en rechterlimiet 1. Bij optimalisatie ontstaat dit automatisch bij benadering
- Er zijn twee lagen met drie neuronen vóór y en één laag met drie neuronen ná y. We noteren deze structuur als [3,3],[3]

Netwerk voor categorische verdeling (1)

Een categorische verdeling heeft de volgende (voorbeeld) structuur:



De getoonde verdeling kan vier mogelijke waarden aannemen: c_1, c_2, c_3 en c_4. De σ-neuronen zijn als sequential logit geordend en kunnen hierdoor afzonderlijk van elkaar elke waarde tussen 0 en 1 aannemen. Merk op dat voor de laatste categorie c_4 geen σ-neuron nodig is.

Netwerk voor categorische verdeling (2)

Opmerkingen bij het (voorbeeld) netwerk:

- De standaard benadering in de literatuur maakt gebruik van een softmax functie bij meer dan twee categorieën, die waarborgt dat individueel gemodelleerde kansen per categorie sommeren tot een totaal van 1
- De hier gekozen opzet is een natuurlijke uitbreiding van een model met twee categorieën en één σ-neuron. Verrassend genoeg kwam ik deze opzet niet tegen in de neurale netwerk literatuur
- Vanwege het gebruik van vier neuronen in beide tanh-lagen noteren we deze structuur als [4,4]
- Behalve de netwerk structuur moeten ook de categorieën in volgorde worden aangegeven, notatie $[c_1, c_2, c_3, c_4]$

Preparatie van data

De data bestaat uit onafhankelijke observaties uit eenzelfde kansverdeling en heeft een tabulaire vorm zonder ontbrekende waarden. Extra stappen in de preparatie zijn:

- Standaardiseren van variabelen door lineaire transformaties, zodat een gemiddelde 0 en een standaardafwijking 1 ontstaan. Dit geeft geschikte grootheden om aan het netwerk aan te bieden
- Scherpe grenzen en overgangen in data vermijden door extra data splitsingen of transformaties (bijvoorbeeld met een log-functie)
- Categorische variabelen vertalen in numerieke waarden. Verklarende variabelen met meer dan twee categorieën worden gesplitst, zodat tweewaardige categorische input ontstaat

Likelihood voor $f_\theta(x)$

Veronderstel n onafhankelijke univariate of multivariate observaties $o_1, o_2, \cdots, o_n$ uit een verdeling met dichtheid f_θ, waarbij elke o_i een getal of vector is en θ een parameter-vector. Definieer:

- likelihood is $f_\theta(o_1) \cdot f_\theta(o_2) \cdot \cdots \cdot f_\theta(o_n)$
- log-likelihood is $h_\Sigma(\theta) = \log(f_\theta(o_1) \cdot f_\theta(o_2) \cdot \cdots \cdot f_\theta(o_n)) = \log(f_\theta(o_1)) + \log(f_\theta(o_2)) + \cdots + \log(f_\theta(o_n))$
- de maximum likelihood schatter is de waarde van de vector θ waarvoor de (log-)likelihood wordt gemaximeerd

De log-likelihood heeft als voordeel ten opzichte van de likelihood dat de analytische uitwerking gemakkelijker is vanwege de sommatie in de formule.

Uitbreiding likelihood naar $f_\theta(y \mid x)$

Het principe van maximum likelihood wordt ook toegepast op univariate voorwaardelijke verdelingen. Veronderstel in dit geval n waargenomen onafhankelijke paren $(y_1, o_1), (y_2, o_2), \cdots, (y_n, o_n)$ uit een voorwaardelijke verdeling met dichtheid $f_\theta(y \mid x)$. Elke o_i is een getal of vector met covariaten en θ is een parameter-vector. Voor de likelihood krijgen we de uitdrukking

$$f_\theta(y_1 \mid o_1) \cdot f_\theta(y_2 \mid o_2) \cdot \cdots \cdot f_\theta(y_n \mid o_n)$$

De log-likelihood is gelijk aan

$$h_\Sigma(\theta) = \log(f_\theta(y_1 \mid o_1) \cdot f_\theta(y_2 \mid o_2) \cdot \cdots \cdot f_\theta(y_n \mid o_n)) =$$
$$\log(f_\theta(y_1 \mid o_1)) + \log(f_\theta(y_2 \mid o_2)) + \cdots + \log(f_\theta(y_n \mid o_n))$$

Waardenbereik van θ

Het waardenbereik van θ (gewichten en biases) moet zo worden gekozen dat een maximum likelihood bestaat. In het algemeen zien we in de netwerk-modellen echter dit:

- Voor θ is een reeks waarden instelbaar waarbij één of enkele waarnemingen een willekeurig hoge dichtheid krijgen (smalle en steile piek)
- De likelihood van alle waarnemingen stijgt daarbij echter ook tot willekeurig hoge waarden

Remedie: de absolute waarde van alle afzonderlijke gewichten (in θ vector) tot een maximum c begrenzen. Hoge waarden van c laten 'grillige' kansverdelingen toe, lage waarden geven een 'vlakke' kansverdeling. In het voorbeeld verderop is $c = 10$ gebruikt.

Gradiënt methoden

Voor het bepalen van de maximum likelihood schatter moeten we uitwijken naar een benaderingsmethode:

- Bekend en gebruikelijk zijn gradiënt-zoekmethoden
- Er zijn vele varianten waaronder de populaire Adam [2] methode

We kiezen een niet-stochastisch algoritme zonder mini-batches als uitgangspunt. De belangrijkste algoritmen kiezen voor een smooth benadering van het optimum, waardoor convergentie soms langzaam kan verlopen. Dit gaf aanleiding tot het ontwikkelen van een nieuw algoritme.

[2] Kingma, Diederik P and Ba, Jimmy Lei, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).

Opzet van GradBounce

De werking van het algoritme is als volgt:

- Voor elke parameter worden naast de waarde de richting, vertraging en snelheid apart bijgehouden
- Gestart wordt op maximale snelheid in de stijgende richting (positieve gradiënt) van elke parameter. De snelheid blijft maximaal tot het moment dat de gradiënt negatief wordt. Dan verandert de richting, waarbij een extra en blijvende vertraging wordt ingevoerd. Die vertraging bevordert convergentie en voorkomt een cyclus
- Zolang de gradiënt stijgend blijft wordt de snelheid tot een maximum stapsgewijs verhoogd. Bij omslaan van de gradiënt wordt de snelheid steeds een stap verlaagd
- Het stuiterende en dempende gedrag van de gradiënt is aanleiding voor de naam GradBounce

Algoritme GradBounce (1)

Algorithm *GradBounce* maximeert een doelfunctie (log-likelihood) gebruikmakend van de gradiënt als functie van de parameters (gewichten en biases). In het voorbeeld zijn de volgende waarden gebruikt: $\alpha = 0.03$, $\beta = 0.8$ en $\gamma = 0.999$.

Require: $\alpha > 0$: stapgrootte
Require: $\beta \in (0, 1)$: eerste verval parameter
Require: $\gamma \in (0, 1)$: tweede verval parameter
Require: $h_{\Sigma}(\theta)$: doelfunctie met parameter vector θ
Require: θ: initiële parameter vector
  $d \leftarrow 0$: vector
  $k \leftarrow 0$: vector

Algoritme GradBounce (2)

while {stop criterium = onwaar} **do**
  $g \leftarrow \nabla_\theta h_\Sigma(\theta)$
  for all $i \in$ {vector indices} **do**
    if $d_i = 0$ **then**
      $d_i \leftarrow \text{sgn}(g_i)$
    else if $\text{sgn}(d_i) \neq \text{sgn}(g_i)$ **then**
      $d_i \leftarrow (-\gamma) \cdot d_i$
      $k_i \leftarrow k_i + 1$
    else
      $k_i \leftarrow \max(k_i - 1, 0)$
    end if
    $\theta_i \leftarrow \theta_i + \alpha \cdot d_i \cdot \beta^{k_i}$
  end for
end while
return θ: resultaat parameter vector

Startwaarden van gewichten en biases

Het initialiseren van gewichten en biases wijkt af van de in de literatuur gangbare werkwijze:

- Startwaarden worden niet-stochastisch vastgesteld. Vanwege het eenmalige karakter van de vaststelling lijkt er geen voordeel aanwezig om die aan het toeval over te laten
- Elke verbinding tussen twee opeenvolgende lagen krijgt de startwaarde $\frac{2}{I+J}$, waarbij J en I het aantal beginpunten en eindpunten zijn van alle verbindingen tussen die lagen
- Biases krijgen geen nul als startwaarde, maar startwaarden die binnen een laag verschillend zijn en symmetrisch rond nul liggen met een gekozen maximale afstand (spread). In het voorbeeld verderop wordt spread=5 gebruikt. De biases van de σ-neuronen krijgen altijd startwaarde nul

Stop criterium

Voor de hand liggende stop criteria zijn:

- Het bereiken van een bepaalde mate van convergentie van de log-likelihood
- Het werken met een apart training- en validatiebestand, waarbij wordt gestopt (early stopping) als de log-likelihood van het validatiebestand verslechtert (overfit op training bestand)
- Het stoppen na een vast aantal iteraties, waarbij visueel wordt geconstateerd dat de ontwikkeling van de log-likelihood voldoende is afgevlakt. Voor eenvoudige netwerken volstaan 100 tot 500 iteraties, voor complexere netwerken 5000 tot 10000 iteraties

In het voorbeeld verderop werken we met het gehele data bestand en vaste aantallen iteraties.

Globale werking

- In de forward propagation wordt het netwerk van de input tot de output doorgerekend
- Voor het continue model wordt in de forward propagation ook de afgeleide bepaald met betrekking tot de y-variabele, zodat naast F ook f wordt berekend. Differentiëren in de forward propagation is overigens niet standaard
- In de backward propagation wordt het netwerk in de omgekeerde volgorde doorgerekend, waarbij de gradiënt van f voor alle parameters in één keer wordt berekend
- Er wordt dus twee keer gedifferentieerd in het continue model, echter zonder expliciet afgeleide functies te bepalen

Formules voor continue verdeling (1)

Definieer:

- L is aantal lagen met neuronen. Laag 1 ontvangt de (gewogen) input variabelen, laag L is de output laag
- $\mathrm{W}^{(l)}$ is gewichten matrix en $\mathrm{b}^{(l)}$ is bias vector van laag l
- $\mathrm{u}^{(l)}$ en $\mathrm{x}^{(l)}$ zijn de input resp. output vector van laag l
- $\mathrm{x}^{(0)} = \mathrm{x}$ is de vector van input variabelen

Voor elke neuron i in laag l met activatie functie s geldt

$$u_i^{(l)} = \sum_j W_{ij}^{(l)} \cdot x_j^{(l-1)} + b_i^{(l)} \tag{1}$$

$$x_i^{(l)} = s(u_i^{(l)}) \tag{2}$$

Formules voor continue verdeling (2)

De afhankelijke variable y wordt toegevoegd aan de input variabelen of aan een laag $1 \leq l < L$, afhankelijk van de gekozen structuur. Laat $z_i^{(l)}$ de afgeleide van $x_i^{(l)}$ zijn naar y, laat s' de afgeleide functie zijn van s en definieer

$$v_i^{(l)} = \sum_j W_{ij}^{(l)} \cdot z_j^{(l-1)} \tag{3}$$

Uit (1) and (2) en de kettingregel van differentiëren volgt

$$z_i^{(l)} = s'(u_i^{(l)}) \cdot v_i^{(l)} \tag{4}$$

Formules voor continue verdeling (3)

Merk op dat de 'eerste' niet-nul $z_i^{(l)}$ de afhankelijke variabele y zelf is en dat z dan gelijk is aan 1. In de volgende lagen ontstaan niet-triviale waarden van z. Na afronding van de forward propagation resulteren de geschatte kansen

$$F(y \mid x_1, \cdots, x_k) = x_1^{(L)} \tag{5}$$

$$f(y \mid x_1, \cdots, x_k) = z_1^{(L)} \tag{6}$$

en de geschatte log-likelihood h gegeven door

$$h = \log(f(y \mid x_1, \cdots, x_k)) = \log(z_1^{(L)}) \tag{7}$$

Formules voor continue verdeling (4)

Backward propagation start met het bepalen van gradiënten als functie van de outputs $x_i^{(l)}$ en de y-gedifferentieerde outputs $z_i^{(l)}$. Voor laag L geldt

$$\frac{\partial h}{\partial x_1^{(L)}} = 0 \tag{8}$$

$$\frac{\partial h}{\partial z_1^{(L)}} = \frac{1}{z_1^{(L)}} \tag{9}$$

Gebruik van de kettingregel van differentiëren geeft

$$\frac{\partial h}{\partial x_i^{(l)}} = \sum_j W_{ji}^{(l)} \cdot \left\{ \frac{\partial h}{\partial x_j^{(l+1)}} \cdot s'(u_j^{(l+1)}) + \frac{\partial h}{\partial z_j^{(l+1)}} \cdot s''(u_j^{(l+1)}) \cdot v_j^{(l+1)} \right\} \tag{10}$$

$$\frac{\partial h}{\partial z_i^{(l)}} = \sum_j W_{ji}^{(l)} \cdot \frac{\partial h}{\partial z_j^{(l+1)}} \cdot s'(u_j^{(l+1)}) \tag{11}$$

Formules voor continue verdeling (5)

De resulterende vergelijkingen voor de gradiënten van de log-likelihood als functie van de gewichten en biases worden

$$\frac{\partial h}{\partial W_{ij}^{(l)}} = \frac{\partial h}{\partial x_i^{(l)}} \cdot s'(u_i^{(l)}) \cdot x_j^{(l-1)} + \frac{\partial h}{\partial z_i^{(l)}} \cdot \left\{ s''(u_i^{(l)}) \cdot v_i^{(l)} \cdot x_j^{(l-1)} + s'(u_i^{(l)}) \cdot z_j^{(l-1)} \right\} \tag{12}$$

$$\frac{\partial h}{\partial b_i^{(l)}} = \frac{\partial h}{\partial x_i^{(l)}} \cdot s'(u_i^{(l)}) + \frac{\partial h}{\partial z_i^{(l)}} \cdot s''(u_i^{(l)}) \cdot v_i^{(l)} \tag{13}$$

Generalisaties van bovenstaande zijn in de literatuur [3] vindbaar.

[3] Zie bijvoorbeeld 'Cardaliaguet, P and Euvrard, G, Approximation of a function and its derivative with a neural network, Neural Networks Vol. 5 (1992), 207–220.'

Formules voor continue verdeling (6)

De eerste en tweede afgeleide van de gebruikte activatie functies kunnen als volgt worden berekend:

$$\begin{aligned} \tanh'(x) &= 1 - \tanh^2(x) \\ \tanh''(x) &= -2\tanh(x) \cdot \tanh'(x) \\ \sigma'(x) &= \sigma(x) \cdot (1 - \sigma(x)) \\ \sigma''(x) &= (1 - 2\sigma(x)) \cdot \sigma'(x) \end{aligned} \tag{14}$$

Formules voor categorische verdeling (1)

Voor een categorische verdeling gebruiken we de dichtheid en daarom is differentiëren niet nodig om de log-likelihood te bepalen. Voor de forward propagation zijn alleen de uitdrukkingen (1) en (2) nodig. Laat m het aantal categorieën zijn en $x_i^{(L)} = P(y = c_i \mid y \notin \bigcup_{j<i} \{c_j\}\,; \mathrm{x})$ voor $1 \leq i \leq m-1$. Uitschrijven geeft

$$f(y \mid \mathrm{x}) = \begin{cases} x_1^{(L)} & \text{if } y = c_1 \\ (1 - x_1^{(L)}) \cdot x_2^{(L)} & \text{if } y = c_2 \\ \dots & \dots \\ (1 - x_1^{(L)}) \cdot \dots \cdot (1 - x_{m-1}^{(L)}) & \text{if } y = c_m \end{cases} \tag{15}$$

Formules voor categorische verdeling (2)

De geschatte log-likelihood h wordt

$$h = \log(f(y \mid \mathrm{x})) = \begin{cases} \log(x_1^{(L)}) & \text{if } y = c_1 \\ \log(1 - x_1^{(L)}) + \log(x_2^{(L)}) & \text{if } y = c_2 \\ \dots & \dots \\ \log(1 - x_1^{(L)}) + \dots + \log(1 - x_{m-1}^{(L)}) & \text{if } y = c_m \end{cases} \tag{16}$$

Uit (16) volgt dat de backward propagation start met

$$\frac{\partial h}{\partial x_i^{(L)}} = \begin{cases} \frac{1}{x_i^{(L)} - 1} & \text{if } y \in \bigcup_{j<i} \{c_j\} \\ \frac{1}{x_i^{(L)}} & \text{if } y = c_i \\ 0 & \text{anders} \end{cases} \tag{17}$$

Formules voor categorische verdeling (3)

Vergelijkingen (10)-(13) reduceren voor een categorische verdeling tot

$$\frac{\partial h}{\partial x_i^{(l)}} = \sum_j W_{ji}^{(l)} \cdot \frac{\partial h}{\partial x_j^{(l+1)}} \cdot s'(u_j^{(l+1)}) \tag{18}$$

$$\frac{\partial h}{\partial W_{ij}^{(l)}} = \frac{\partial h}{\partial x_i^{(l)}} \cdot s'(u_i^{(l)}) \cdot x_j^{(l-1)} \tag{19}$$

$$\frac{\partial h}{\partial b_i^{(l)}} = \frac{\partial h}{\partial x_i^{(l)}} \cdot s'(u_i^{(l)}) \tag{20}$$

Hiermee is het formularium voor de gradiënten compleet.

Invloed van de grootte van het netwerk

- Het kleinst mogelijke netwerk bevat alleen σ-neuronen. Er geldt:
  - continue verdeling (één σ-neuron): de gewichten tonen de globale verhouding tussen de variabelen. Deze vorm heeft in de praktijk slechts een analytische betekenis, vergelijkbaar met het Linear Model (kleinste kwadraten methode)
  - categorische verdeling (één of meer σ-neuronen): deze vorm is equivalent met het Logistic Model
- Te kleine netwerken geven een underfit, te grote netwerken zorgen voor een overfit
- Keuze: optimale grootte op basis van een informatie criterium. Het gehanteerde likelihood principe maakt dit mogelijk, net als bij het Generalized Linear Model

Informatie criteria (1)

- Een informatie criterium geeft een penalty op de log-likelihood. Een betere model-fit van een groter netwerk wordt hierdoor afgewogen tegen de hoeveelheid benodigde extra parameters
- Bekendst zijn het Akaike Information Criterion (AIC) en het Bayesian Information Criterion (BIC)
- In formules is $\hat{L}$ de geschatte log-likelihood, p het aantal parameters (gewichten en biases) en n het aantal observaties

$$\text{AICc} = -2\hat{L} + 2p \cdot \frac{n}{n - p - 1}$$

$$\text{BIC} = -2\hat{L} + p \log(n)$$

Informatie criteria (2)

- AICc betekent AIC inclusief noodzakelijke correctie-term (breuk in formule) in het geval van kleinere aantallen. Deze correctie gebruiken we hier echter altijd
- Voor AICc en BIC geldt: hoe lager de waarde, hoe beter de model-score
- Starten met eenvoudig netwerk en dit gestaag uitbreiden
- Technische eis AICc/BIC: alleen geldig bij 'geneste' modellen, daarom strikt genomen alleen bij uitbreiding neuronen in bestaande lagen
- AICc is minder kritisch dan BIC. Toepassing BIC geeft 'generalistischer' model, AICc geeft 'scherper' model

Programmeertaal Julia

- Het algoritme en formularium vereisen intensieve berekeningen. Hiervoor is maatwerk programmatuur in combinatie met een snelle programmeertaal van belang
- Implementatie kan in bijvoorbeeld R of Python. Ik heb echter gekozen voor het nieuwere en snellere Julia [4]
- Julia code wordt gecompileerd en benadert de snelheid van C
- Focus ligt op performance. Voorbeeld: alle data en parameters in arrays met uitsluitend float-datatype
- Julia code kan ook als package worden gebruikt vanuit bijvoorbeeld R

[4] Bezanson, Jeff and Edelman, Alan and Karpinski, Stefan and Shah, Viral B, Julia: A fresh approach to numerical computing, SIAM Rev. 59-1 (2017), 65-98.

Benodigde kernfuncties

De volgende kernfuncties zijn benodigd voor implementatie:

- Schatten model: w, b <- fitDist(data, layers1, (layers2,) (categories,) iterations, spread=5, α=0.03, β=0.8, γ=0.999, c=10)
- Kansen: f <- evalPDF(observation, layers1, (layers2,) (categories,) w, b) en F <- evalCDF(observation, layers1, (layers2,) (categories,) w, b)
- Limieten: inf, sup <- limitsCDF(layers1, layers2, w, b)
- Inverse cumulatieve verdeling: quantileCDF(covariats, p, layers1, (layers2,) (categories,) w, b)

Model gebruiken zonder formules

- Een gebruiker heeft in het algemeen geen netwerkformules, gewichten en biases nodig
- Een gebruiker wil wel functies om random samples te kunnen maken voor één- of meerdimensionale simulaties
- Daarnaast wil een gebruiker functies om kwantielen en momenten (voor verwachting, variantie etc.) te berekenen
- Kortom: de gebruiker wil vooral gegenereerde data en kan daarmee analyses maken in eigen vertrouwde tools, zoals R, Excel en Power BI. De geschatte modellen zelf zijn daarbij onzichtbaar aanwezig
- Deze werkwijze bevordert een breed gebruik van het model

Dataset en scatter plot

De kunstmatig geconstrueerde voorbeeld dataset kent drie variabelen: x en y zijn continue variabelen en de kleur z is een categorische variabele (rood=1, groen=2, blauw=3). De set bestaat uit 1000 onafhankelijke en gestandaardiseerde observaties. In een scatter plot wordt de structuur van de data goed zichtbaar.

Eenvoudigste model voor $F(x)$

Het meest eenvoudige netwerk om de verdeling van x te schatten bestaat uit één neuron met een logistische activatie functie. Feitelijk schatten we hiermee een logistische verdeling. In minder dan 500 iteraties ontstaat een stabiele oplossing. Met behulp van een histogram van x kunnen we de geschatte kansdichtheid visueel beoordelen. We concluderen dat de fit nog niet goed is.

Meer modellen voor $F(x)$

Voor acht netwerkstructuren is $F(x)$ geschat met 500 iteraties. Bij complexere netwerken wordt de dichtheid vaak zichtbaar complexer.

Optimaliseren model voor $F(x)$

De gekozen modellen leiden tot de volgende uitkomsten:

nr	structuur	parameters	log-likelihood	AICc	BIC
1	[],[]	2	-1438.1	2880.2	2890.0
2	[],[1]	4	-1445.4	2898.8	2918.4
3	[],[2]	7	-1410.8	2835.7	2869.9
4	[],[3]	10	-1410.8	2841.8	2890.6
5	[],[1,1]	6	-1446.1	2904.3	2933.7
6	[],[2,2]	13	-1409.9	2846.3	2909.7
7	[],[3,3]	22	-1405.3	2855.5	2962.5
8	[],[8,6,5]	111	-1405.2	3060.4	3577.1

Overigens geeft de standaard normale verdeling een log-likelihood van -1418.4, tussen het eerste en derde model in.

Het optimale model voor $F(x)$

Het derde model met één extra laag met twee neuronen blijkt optimaal.
De linkerlimiet van F is 0.000042 en de rechterlimiet is 0.999975, dichtbij de gewenste waarden 0 respectievelijk 1.

Eigenschappen optimale model voor $F(x)$

De formule voor de cumulatieve verdeling van x is

$$F(x) = \sigma(W_{11}^{(2)} \tanh(W_{11}^{(1)} x + b_1^{(1)}) + W_{12}^{(2)} \tanh(W_{21}^{(1)} x + b_2^{(1)}) + b_1^{(2)})$$

Met de geschatte waarden voor de gewichten en biases wordt dit

$$F(x) = \sigma(4.77 \tanh(0.68x - 1.76) + 5.57 \tanh(0.48x + 1.03) + 0.26)$$

De verwachting van x is 0.003 en de standaarddeviatie is 1.004 (met eerder genoemde benadering), dichtbij 0 respectievelijk 1. Enkele andere resultaten zijn:

- $F^{-1}(0.025) = -1.830$
- $F^{-1}(0.5) = -0.055$ (mediaan)
- $F^{-1}(0.975) = 1.953$

Optimaliseren model voor $F(y \mid x)$

Er zijn drie modellen doorgerekend met 10000 iteraties. Op basis van de informatie criteria wordt het tweede model gekozen. De linkerlimiet van $F(y \mid x)$ is 0.000001, de rechterlimiet is 0.998731.

nr	structuur	parameters	log-likelihood	AICc	BIC
1	[2,2],[2,2]	27	-92.0	239.5	370.5
2	[3,3],[3,3]	49	103.2	-103.1	132.2
3	[4,4],[4,4]	77	128.2	-89.4	275.5

Het optimale model voor $F(y \mid x)$

Hieronder staan de observaties met de geschatte verwachtingen en 95% betrouwbaarheidsintervallen. Een random sample van 1000 trekkingen (x, y) bevestigt het goede schattingsresultaat.

Optimaliseren model voor $F(z \mid x, y)$

Voor de categorieën $[1, 2, 3]$ zijn twee σ-neuronen nodig in de laatste laag. Twee modellen zijn doorgerekend met 10000 iteraties. Op basis van de informatie criteria wordt het eerste model gekozen.

nr	structuur	parameters	log-likelihood	AICc	BIC
1	[2,2]	18	-707.2	1451.2	1538.8
2	[3,3]	29	-701.9	1463.7	1604.2

Het optimale model voor $F(z \mid x, y)$

Hieronder staan de observaties en een random sample van 1000 trekkingen (x, y, z) uit de optimale modellen afgebeeld.

Conclusies

- Neurale netwerken blijken ook nuttig in statistical learning, waar die tot nu toe vrijwel ontbreken
- Gebruik van generieke distributies maakt het model breed inzetbaar en daarmee is het ook geschikt als benchmark
- Tegenover intensieve berekeningen staat weinig mensenwerk
- Het model is uitgewerkt in een volledig gespecificeerde Machine Learning toepassing
- Het model zal zich in de Business Analytics praktijk moeten gaan bewijzen