University of Brasília
Departament of Statistics

Recalibration of Gaussian Neural Network regression models: the *recalibratiNN* package

Carolina Musso

Project submitted to the Department of Statistics at the University of Brasília in fulfilment of the requirements for obtaining the degree of Bachelor in Statistics.

Brasília
2023

Carolina Musso

Recalibration of Gaussian Neural Network regression models: the *recalibratiNN* package

Surpevisor: Prof. Guilherme Souza Rodrigues

Project submitted to the Department of Statistics at the University of Brasília in fulfilment of the requirements for obtaining the degree of Bachelor in Statistics.

Brasília
2023

I dedicate this work to the cherished memory of my beloved grandfather and academic inspiration, Fernando Musso.

Acknowledgements

First and foremost, I should thank myself for bringing me in (mostly) one piece from the beginning of this journey to the finish line. It was indeed challenging and demanded a great deal of perseverance. Jokes aside, this journey wouldn't have been possible without the love, support, and inspiration from my family, friends, teachers, and my husband, all of whom had to make their own sacrifices (though not as great as my own, of course).

First, I would like to extend my heartfelt thanks to my parents, Dr. Musso and Dra. Carmen, for being my eternal role models and sources of inspiration, always supporting my decisions, although often they may not fully understand what they are. Additionally, my gratitude goes to my stepmother Teté and stepfather Ricardo, whom I love like parents and who, I assume, love me as their own daughter. A special shoutout to my brother Bruno, who most of the time truly gets me. Thank you for maintaining a balance in my life with exercise and wine in equal measure. I also want to thank my sister-in-law, Amanda, for her graciousness and love, and for bringing the beautiful Menina Amélia, Mig Miguito, and Baby-Oli into the world, who are the joy of our lives and a constant reminder of what's truly important. Lastly, to all my grandparents, biological and borrowed, wherever they are, in this world or another, I am forever grateful for all their love and care.

Many thanks also go to the University of Brasília, which has been like a second home to me since 2005. I want to acknowledge all the teachers and staff who have been integral to my journey since my first bachelor's degree. Special thanks to Prof. Heloisa Miranda, whose enthusiastic support for my decision to become a statistician was invaluable. I'm not sure if Prof. Antonio Francisco even remembers me, but it was his guidance that planted the seed of programming and mathematics, prepareing me for what layed ahead. So, a heartfelt thank you is due to you, Chicão.

During my statistics studies, I had the privilege of learning from outstanding professionals in both statistics and mathematics. My deepest gratitude to Prof. Maria Tereza for her unwavering support, to Professor George for his, lets say, 'disruptive' perspective and exceptional classes, and Prof. Gilardoni for enlightening me with the elegant Bayesian approach to understanding the world. A special bow to Professors Ana Maria, Cira, Alan, Cátia, Raul, Fiorucci... Also, a big thank you to Profa Thais and Prof. Donald for their collaborative efforts in refining this work (gently, please). Many thanks to Ricardo Torres, whom I've never met in person, yet who has played an active and patient role in helping me understand the various functions and method.

Last but definitely not least, the best supervisor and teacher, and the one whose fan club I proudly lead: Prof. Guilherme. Thank you for your patience, respect, receptivity, trust and for gently encouraging me to believe in myself. The transformative journey through statistics was significantly influenced by your dedication to teaching and your commitment to introducing advanced topics. Your classes reached that sweet spot between mind-blowiness and despair (*chuckles*), providing the ideal motivation to reach the next level. Aside from the fact that you haven't personally tested the package we are presenting here, I have absolutely no complaints of any nature.

There are so many more people who played crucial roles, big or small, in leading me to where I am today. My friends from my first bachelor's degree, in Biology, my friends from Portugal (who, I expect, will now become my clients), and my coworkers who recognized the value I brought to our team. Special thanks to Thay, João, and Pri, who have become not only my partners but also my dear friends, helping to lighten this burden. My gratitude extends to all my youthful friends from the Statistics course, who made this journey far more enjoyable and lighter. I must also mention the special group of people who have been particularly close and inspiring: Rafael and César (with whom I formed the council of elder wise(wo)men), Pedro Henrique (without him, I would have likely flunked out in Introduction to Probability right from the start), Ju, Thay, and Jess for their unwavering support, especially during that one semester when I almost failed the unnameable subject. My heartfelt thanks to the many others who brought joy and a special touch of human connection to my life: Tamara, Laurinha, Kevin, Erica, Malu, Fabiana, Stefan, Luis, Henrique, Elisa, Léos, Stefan, José Vitor (the favorite son, who brought honor to our family), José, Silvania, Antonios and Joões, Gabi, Rayssa, Max, and everyone else I forgot... thank you all!

Finally, to my love, my husband Eduardo. Even though you were often (always) unaware of the specifics, you were always there for me with a comforting shoulder and empowering words. Your love, support, and ability to bring me back to earth whenever necessary have been invaluable. You bring balance, fun, art, laughter, music, and all those intangible things that are essential in life. I love you.

Abstract

Artificial Neural Networks (ANNs) are powerful models in representation learning, known for their high prediction performance. However, their predictions often lack proper calibration, leading to unreliable uncertainty estimation. Various methods have been proposed to address this issue, but implementing and selecting the appropriate technique can be challenging. This project aimed to develop an R package that provides a computational implementation of a quantile-based post-processing technique. The functions provided can recalibrate Gaussian models, such as neural networks adjusted with the MSE loss function. The method leverages information from cumulative probabilities, allowing the generation of Monte Carlo samples from the recalibrated predictive distribution. The *recalibratriNN* package also includes diagnostic functions to identify miscalibration. We believe the availability of a user-friendly and optimized toolset will facilitate the recalibration process and encourage researchers and practitioners to obtain reliable uncertainty estimates from their Gaussian models, thereby enabling and encouraging calibration practices.

Key-Words: R package, calibration, coverage, Deep Learning.

Resumo

Redes Neurais Artificiais (ANNs) são modelos poderosos em aprendizado de representações, conhecidos por seu alto desempenho preditivo. No entanto, suas previsões não são calibradas, levando a estimativas de incerteza pouco confiáveis. Diversos métodos de recalibração foram propostos para abordar essa questão. Entretanto, implementar e selecionar a técnica apropriada pode ser desafiador. Este projeto teve como objetivo desenvolver um pacote em R que fornece uma implementação computacional de uma técnica quantílica de pós-processamento. As funções fornecidas podem recalibrar modelos Gaussianos, como redes neurais ajustadas com a função de perda MSE. O método faz uso das probabilidades cumulativas do modelo ajustado, permitindo a geração de amostras de Monte Carlo de uma distribuição preditiva desconhecida, mas recalibrada. O pacote *recalibratriNN* também inclui funções de diagnóstico para visualizar a falta de calibração. Acreditamos que a disponibilidade de um conjunto de ferramentas amigável e otimizado facilitará o processo de recalibração e incentivará pesquisadores e profissionais a obterem estimativas de incerteza confiáveis de seus modelos Gaussianos, permitindo e incentivando práticas de calibração.

Palavras-Chave: Pacote R, calibração, cobertura, *Deep Learning*.

List of Tables

1 Kolmogorov-Smirnov Tests results for the observations globally and in the vicinity of green and blue points shown in Fig. 1 20

List of Figures

1 Confidence Interval coverage throughout the covariate space of a simple linear regression. 16

2 Confidence Interval coverage throughout the covariate space for a multiple regression 17

3 (a) Global evaluation of the PIT value distributions. (b) Local evaluation of the PIT value distributions. (c) Scatter plot of the estimated and empirical PIT values, globally (grey), and local (blue/green). (d) Corresponding points from the green/blue histogram, with the respective predictive distributions (red lines) of the fitted linear model. 19

4 Density of global PIT-values. a) Default output. b) Customized parameters . . . 28

5 Predicted VS Empirical Cumulative Distribution. 29

6 Local PIT-Values distribution 30

7 Predicted vs Empirical PIT-values distribution. a) Default, b) Customized. . . . 31

8 Local visualization using the facet=t option. 32

9 Further customization allowed as in any ggplot. 33

10 Global and Local PIT-values distribution analysed in the new (test) set, using the calibrated predictions. 35

11 Global and Local PIT-values distribution using local calibration with propotion of calibration = 1. 36

12 Local PIT-values distribution using local claubration with propotion of calibration = 1. 37

13 Coverage of recalibrated samples. True vc recalibrated Mean 38

14 PIT-values distribution in the a) uncalibrated ANN and b) recalibrated. 41

15 Coverage of the IC and prediction of mean in the test (new) set after recalibration of ANN in the covariate space. 42

Contents

1 Introduction 11

2 Background and related work 14

2.1 How packages can help 14

2.2 Diagnosis of miscalibration 15

2.3 Recalibration methods 20

3 Methodology and Results 23

3.1 Calculation PIT-Values 23

3.2 Visualizing miscalibration 27

3.2.1 Global miscalibration 27

3.2.2 Local miscalibration 30

3.3 Recalibation 34

3.3.1 Linear model 34

3.3.2 Neural network 38

4 Conclusions and further work 43

Appendices 46

1 Introduction

In recent years, machine learning methods incorporating artificial neural networks (ANNs) have gained substantial attention. While this field has a rich history dating back to the early twentieth century, advancements in computing power have made computationally intensive methods more accessible than ever before. Moreover, researchers have developed new and improved techniques, such as Adam optimization, batch normalization, and dropout, which have significantly enhanced model performance. Furthermore, the exponential growth in available data has opened up exciting possibilities for exploring increasingly complex models. This surge in data availability has led to a substantial expansion in the applications of ANN in various fields. These advancements have paved the way for solving complex problems across a wide range of domains (GOODFELLOW; BENGIO; COURVILLE, 2016).

Multi-Layer Artificial Neural Networks, also known as deep learning algorithms (DL), are an integral part of the field of representation learning. These methods are particularly valuable in scenarios where it is challenging to define the crucial features necessary for accurately mapping outputs. Therefore, DL offers a solution by providing simpler concepts that allow computers to construct more complex ones. In this regard, ANNs excel in capturing complex, nonlinear relationships between observations and predictions. They possess the remarkable ability to represent a wide range of phenomena, making them a compelling area of research with countless potential applications. The flexibility of ANNs to model intricate relationships, coupled with their exceptional predictive performance, underscores their significance in various fields.

The field continues to make strides in enhancing model accuracy and uncovering emerging patterns within complex datasets, showcasing ongoing development. However, while accurate outcome predictions are crucial, they are not the sole defining characteristic of an effective ANN. A prevalent issue with trained models is their inadequate ability to quantify prediction uncertainty correctly. Such models are considered uncalibrated. Interestingly, some techniques employed to improve ANN performance have been observed to deteriorate proper calibration (GUO et al., 2017). To address this challenge, it is imperative to focus not only on improving model accuracy but also on refining uncertainty estimation. By doing so, we can enhance the reliability of predictions, foster trust in the model, and enable better decision-making in various domains.

From a quantile perspective, poorly calibrated models can be understood as models that exhibit a discrepancy between empirical and predicted distributions. In other words, they generate confidence intervals that fail to encompass the corresponding percentage of true outcomes. For instance, when dealing with a sufficiently large dataset, a 95% confidence interval should ideally capture 95% of the results, neither less nor more.

More formally, a sufficient calibration condition for a confidence level p is defined by Kuleshov, Fenner e Ermon (2018) as:

$$\mathbb{P}(Y \leq \hat{F_Y}^{-1}(p)) = p, \forall\, p \in [0,1] \tag{1.0.1}$$

for continuous outcomes; where $\hat{F}$ is the neural network cumulative predictive distribution and Y is the predicted outcome for that network. The same concept can be applied to discrete outcomes, particularly binary outcomes, in the following manner:

$$\mathbb{P}(Y = 1 | \hat{F_Y}(y_t) = p) = p, \forall\, p \in [0,1]$$

While the development of new and sophisticated models often takes the spotlight, calibration is an equally significant concept that deserves attention. It's worth noting that without calibration, the output of a model cannot be interpreted probabilistically. This renders calibration critical, especially in scenarios where decisions impact human lives. The application of uncalibrated models in real-world decisions can lead to significant problems and implications(GUO et al., 2017). Consequently, reliable methods for measuring uncertainty are just as important as achieving high accuracy. Therefore, with calibration, we can ensure that model outputs are not only accurate but also properly interpreted. Furthermore, the development of calibration techniques can significantly enhance the reliability and usefulness of models in real-world applications.

In this context, some methods were designed to improve model calibration. There are methods that are useful in specific scientific domains or that require complex modifications of the model (LAKSHMINARAYANAN; PRITZEL; BLUNDELL, 2017; GNEITING; RAFTERY, 2005). Nevertheless, there are also methods that propose to perform calibration as a postprocessing step, rendering them model-agnostic and simpler. These techniques, commonly known as recalibration methods, have witnessed notable progress. Yet, the use of recalibration remains suboptimal, particularly concerning regression tasks when compared to classification problems. One contributing factor could be the limited availability of code/software to facilitate these recalibration tasks.

Quantile-based global recalibrations are among the most common methods for addressing regression problems. They are particularly useful for being non-parametric and therefore agnostic of the model usded for fitting. Kuleshov, Fenner e Ermon (2018) introduced a technique involving the training of an auxiliary model, specifically an isotonic regression, which contrasts on the empirical versus theoretical distribution of cumulative probabilities to achieve calibrated predictive distribution. In the domain of R packages, it is worth mentioning the `probably` package (KUHN; VAUGHAN; RUIZ, 2023), which is part of the `tidymodels` framework. This package offers three recalibration methods: GAM (General Additive Models), Isotonic Regression, and Beta Regression. While it provides comprehensive documentation for recalibration in classification models, its options for regression models are more limited, offering only a single visualization type—a scatter plot of observed versus predicted values—to assess miscalibration. In contrast, the Python package `ml_insights` (LUCENA, 2018) offers additional visualization tools, such as reliability diagrams and histograms, and implements spline recalibration to refine the predicted class probabilities. However, both packages provide only global calibration, that is, that implicitly assumes that the miscalibration behaves consistently through the covariate space. It is also important to note that, if not using packages, it may be hard for the researchers to understand the theory necessary to implement the technique themselves.

More recently, the studies by Torres (2023) and Kuleshov e Deshpande (2021) have proposed other methods for local model calibration. In this context, the present work introduces an R package designed to visually diagnose local miscalibration, while also providing an option to recalibrate the model locally. It incorporates the method proposed by Torres (2023). This method is adaptable for calibration across various representations of the covariate space, making it particularly useful for recalibrating Artificial Neural Networks (ANNs). The current version of the package is compatible with Gaussian models, such as ANNs that have been trained using the Mean Squared Error loss function.

We developed a package consisting of two fundamental components: model diagnostics and model recalibration. The purpose of model diagnostics is to assess the extent and nature of the lack of calibration in the fitted model. Upon performing the diagnostics, users will have the option to proceed with recalibration, applying it either globally or locally.

2 Background and related work

2.1 How packages can help

Addressing this challenge, packages emerge as invaluable tools for code sharing, enabling reproducibility and simplifying task execution. In the realm of machine learning, libraries play a pivotal role, and among them, R stands out as a popular language for package development and data science. The Comprehensive R Archive Network (CRAN) currently hosts, as of November 2023, an extensive collection of over 19 thousand packages, providing robust statistical capabilities and facilitating rapid visualizations, among other essential tasks. These libraries not only save significant time and effort but also empower practitioners to leverage techniques that might be challenging to implement from scratch. Moreover, through the open-source community, scientists can share their unique contributions, reaching a broader audience and enhancing the overall code quality.

There are many R packages available for fitting, visualizing, and interpreting ANNs, such as `neuralnet` (GÜNTHER; FRITSCH, 2010), `RSNNS` (BERGMEIR; BENíTEZ, 2012) and `NeuralNetTools` (BECK, 2018), in addition to the famous `Keras` (ALLAIRE; CHOLLET, 2023) and `Torch` (FALBEL; LURASCHI, 2023). As far as our knowledge extends, there is currently only one existing R package (`probably`) designed to facilitate the recalibration of trained networks, as discussed before. This project aims to address this gap by offering a codebase that implements an efficient post-processing techniques for artificial ANNs and other Gaussian models. By providing this package, we seek to bridge the existing void and empower researchers and practitioners to recalibrate ANNs effectively, facilitating more reliable and trustworthy predictions.

Packages, also known as libraries, play a crucial role in making code easily accessible to others. As mentioned earlier, the Comprehensive R Archive Network (CRAN) boasts an extensive collection of nearly 20 thousand readily downloadable packages that can be installed directly from the R console. Most users of R are familiar with the base functions `install.packages()` and `library()`, which enable the utilization of these packages. In addition to CRAN, packages published on GitHub can also be shared and utilized. This approach expedites the publishing process as there is no need for formal approval, allowing for the dissemination of up-to-date versions in near real-time. For installation, the `devtools` package can be utilized, employing `install_github()` along with the repository address. Alternatively, the package manager `pacman` provides the functions `p_load()` or `p_load_gh()` for seamless installation and loading of packages from either CRAN or GitHub, respectively.

To demonstrate, a simple function has been shared in a GitHub repository, which can be readily installed on your computer with the code below:

```r
if (!require("pacman")) install.packages("pacman")
pacman::p_load_gh("cmusso86/UmPacoteAleatorio")
```

We see that the process of sharing methods through packages offers a straightforward way to assist others in solving their current problems. Moreover, building packages can prove invaluable for future work, as it saves time by automating tasks. This is possible due to the standardized development conventions that packages adhere to. Fortunately, there are numerous resources available today to guide the package-building process, such as Wickham e Bryan (2015). These guidelines helps to garantee the package is built correctly, with working functions, documentations and examples, enabling it to be easily shared on github. In this way, we aimed to simplify knowledge sharing and enhance the overall workflow for users, fostering a more efficient and productive development environment.

Furthermore, the RStudio IDE, along with specific packages, greatly facilitates package development. Essential packages like `devtools`, `usethis`, and `roxygen2` automate common development tasks such as function wrapping, documentation and dependency files creation, as well as code compilation. RStudio offers numerous features that simplify the package development workflow, making it an excellent environment for this purpose. Documentation plays a crucial role in ensuring package usability. Regarding this, RStudio allows the use of Rmarkdown to create vignettes, while the `roxygen2` package enables the creation of `.Rd` files for basic package and function documentation. To see an example of the output generated by the `roxygen2` documentation, you can run `?mediaDificil` in your console after loading the toy package mentioned earlier. This code will display the function documentation in the Help Panel within RStudio.

2.2 Diagnosis of miscalibration

As described in the introduction section, misscalibration of a regression model can be defined as in the expression 1.0.1. Furthermore, the calibration of a model can be evaluated by comparing observed values with their respective estimated conditional (or predictive) distributions. This evaluation can be conducted globally, examining overall calibration, or locally, investigating calibration at specific parts of the covariate space. Lets take some artificial examples to better visualize this concept.

In the following example we fitted a simple linear regression to non-linear heteroscedastic data. Therefore, in this toy example, the model consists has only 3 parameters (β_0, β_1 and σ) and one covariate, and the data can be represented in a scatter plot. The data were simulated and fitted with:

```
set.seed(42)

data <- tibble(x  = runif(10^5, 2, 20),
               mu = 10 + 5*x^2,
               sg = 30*x,
               y  = rnorm(mu, sg))

model <- lm(y~x, data)
```

Upon analyzing the percentage of points that fall within the 95% confidence interval, we would typically expect around 95% of the points to be encompassed. Indeed, for this model, when quantifying this percentage globally, it amounts to 94.45%, aligning with expectations. However, as illustrated in the graph below (Figure 1), there is a noticeable asymmetry in coverage.

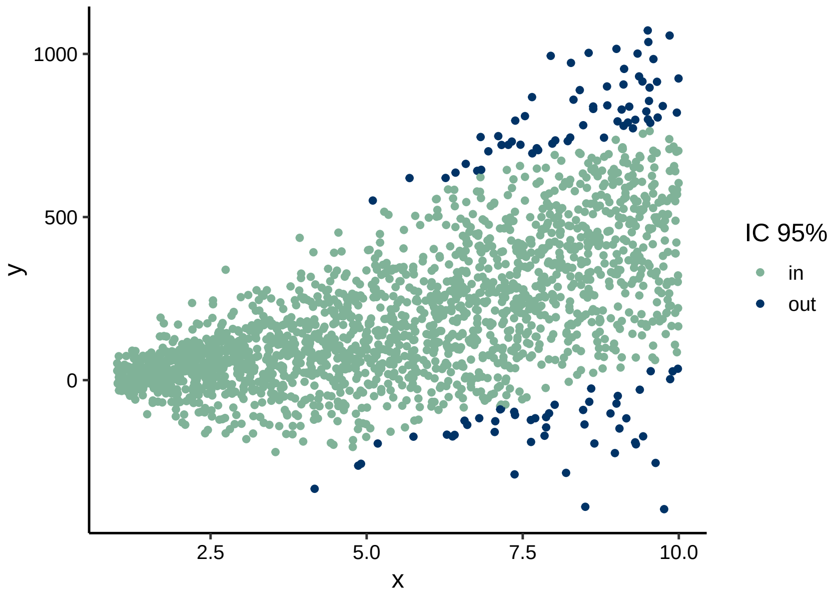

Figure 1: Confidence Interval coverage throughout the covariate space of a simple linear regression.

In another example, we fit a linear regression model with two covariates to assess local miscalibration. This issue becomes particularly evident when attempting to represent a non-linear surface using a linear model. The data and model were created as follows:

```
set.seed(42)

data <- tibble(x1 = runif(10^5, -3, 3),
               x2 = runif(10^5, -3, 3)) %>%
  mutate(mu = abs(x1^3 - 30*sin(x2) + 10),
         y  = rnorm(10^5, mean = mu, sd=1))

model <- lm(y~., data)
```

In this instance, global coverage again appears to be adequate at 95.7%. Nonetheless, a closer examination of local calibration reveals a distinct pattern. Consequently, there are specific regions within the space where the model is systematically generating incorrect predictions, as illustrated in Figure 2.

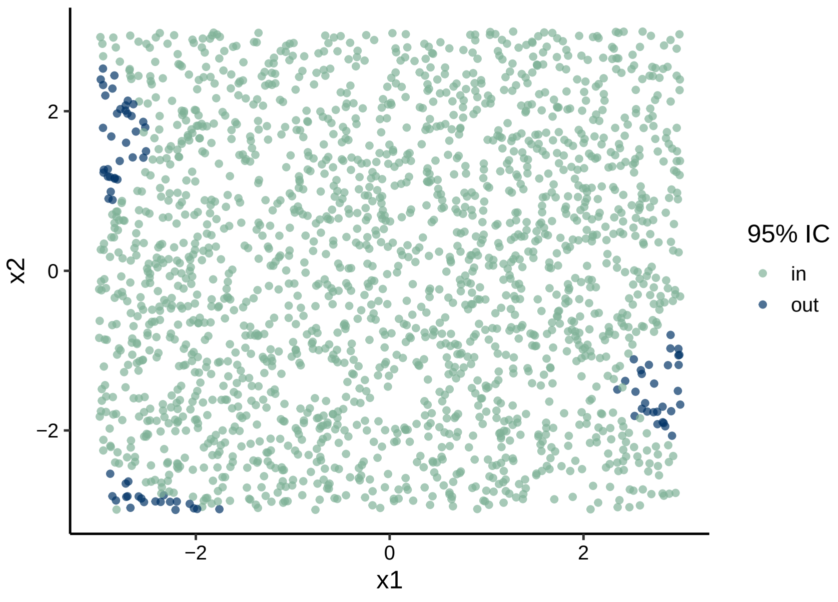

Figure 2: Confidence Interval coverage throughout the covariate space for a multiple regression

In higher-dimensional spaces, the task of visualization becomes impractical, especially when working with models built using Neural Networks that frequently employ high-dimensional data. The complexity of intuitively visualizing these models increases significantly in such scenarios. To achieve a 2-D visualization in these cases, dimensionality reduction techniques are necessary. Methods like Principal Component Analysis (PCA), t-Distributed Stochastic Neighbor Embedding (t-SNE), or Autoencoders can be employed to effectively condense the data into a more visually comprehensible form.

An alternative approach to analyzing global calibration involves examining the histogram of Probability Integral Transform (PIT) values, which is unrelated to dimensionality issues. IT values represent the estimated cumulative probability in the predicted distribution for each observed value. This technique involves constructing a histogram of the cumulative distribution functions estimated by the model applied to each corresponding observation. The Probability Integral Transformation theorem (ENGELHARDT; BAIN, 1987) validates this approach, stating that when a cumulative distribution is applied to data well-defined by such distribution, it yields a uniformly distributed variable. Therefore, if the model is well calibrated, the PIT values would be uniformly distributed within the interval [0, 1]. Notice that uniformity of the PIT values is a necessary but not sufficient condition for calibration.

Generally, the distribution of PIT values will yield four basic patterns (TORRES, 2023). First, a uniform distribution indicates global calibration. Second, higher frequencies accumulated in the tails suggest an underestimation of the true variance. Third, a peak somewhere within the graph indicates an overestimation of the true variance. Finally, shifts in these peaks, apart from indicating an overestimation of variance, can also signal an underestimation of the mean (if the peak is to the right) or an overestimation of the mean (if the peak is to the left).

The technique mentioned above provides global diagnostics for evaluating the model. However, relying solely on global evaluation can be misleading since miscalibration may vary across different regions of the observational space, as discussed above. Therefore, the described principle can be adapted to assess calibration locally. In their work, Torres (2023) demonstrated how this localized calibration evaluation, followed by corresponding local corrections, can lead to more substantial improvements in the model compared to a global approach.

Figure 3 showcases the global VS local calibration diagnostics of a model. To make the analysis of PIT value distributions more comprehensible, we will visualize these principles based on the same example already presented in Figure 1 . Because the model depends on a single covariate, the data can be represented in a scatter plot; which is an instructive illustration to grasp how the PIT-values histogram informs us about model calibration.

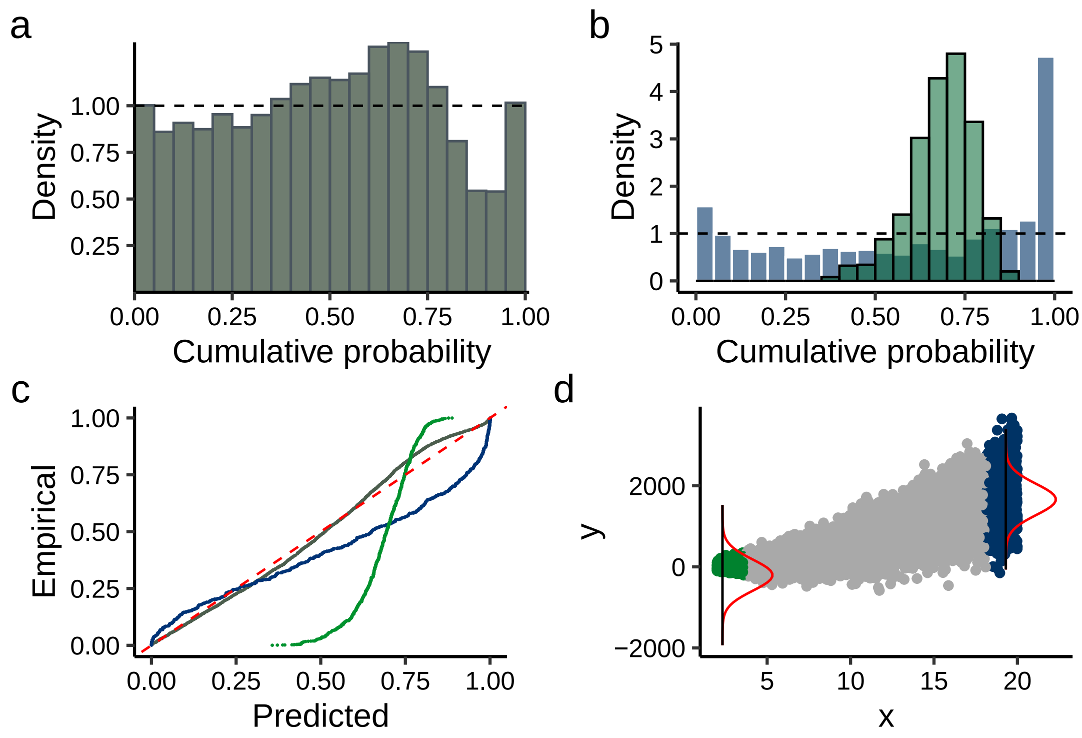

Figure 3: (a) Global evaluation of the PIT value distributions. (b) Local evaluation of the PIT value distributions. (c) Scatter plot of the estimated and empirical PIT values, globally (grey), and local (blue/green). (d) Corresponding points from the green/blue histogram, with the respective predictive distributions (red lines) of the fitted linear model.

In Figure 3 (a), the global evaluation reveals some deviation from a uniform distribution, particularly towards the right tail. This can be observed in the Predicted x Empirical PIT-values distribution plot displayed. However, Figure 3 (b), a more pronounced local miscalibration becomes evident. Specifically, in the vicinity of the green points (Figure 3 (d)), the predicted distribution (represented by the red curve) tends to overestimate the variance and underestimate the true mean. This discrepancy is reflected in Figure 3 (b), which exhibits a concentration with a peak shifted to the right. Conversely, in the vicinity of the blue points (Figure 3 (d)), the model underestimates the variance, resulting in a histogram pattern (Figure 3 (b)) where the distribution is concentrated in the tails.

It is also possible to test for the distribution of PIT values. One common test is the Kolmogorov-Smirnov test, which can be implemented using the `ks.test()` function from the `stats` package, among other possible tests . The former test assesses whether the values are drawn from a target distribution, say the uniform distribution. In Table 1, we present the results of these tests for both the global (gray) and local (green and blue data) evaluations shown in Figure 3. Our findings indicate that the distribution does not conform to a Uniform [0,1] distribution in either case, with the deviation being more pronounced in the local evaluation. This is evident from the higher test statistics in the local situations, although all p-values are numerically zero.

Table 1: Kolmogorov-Smirnov Tests results for the observations globally and in the vicinity of green and blue points shown in Fig. 1

Group	Statistic	p-value	Method
Local green	0.7068	0	Kolmogorov-Smirnov test
Local blue	0.4972	0	Kolmogorov-Smirnov test
Global	0.4982	0	Kolmogorov-Smirnov test

2.3 Recalibration methods

After model diagnostics, the user may choose to perform a recalibration. The method implemented in this package is the one provided by Torres (2023) that proposes to generate samples of the recalibrated distribution, and can be applied to perform local calibration. This feature is particullarly interesting in the case of ANN since they are known to underfit in low-density regions (XIONG et al., 2023). The authors of this method propose an algorithm that capitalizes on the identification of biases within the histograms of PIT value distributions to improve model calibration. Additionally, this method allows for the utilization of an intermediate layer to diagnose local biases, while simultaneously mitigating the challenges associated with the curse of dimensionality. This approach draws inspiration from the work of Rodrigues, Prangle e Sisson (2018) in the context of Approximate Bayesian Computation, where it was initially employed to calibrate estimated posterior distributions. In Torres (2023) study, the algorithm is adapted specifically for the case of an ANN.

In this scenario, the insights gained from the diagnostic stage can be utilized to correct the predictive distribution. Moreover, the recalibration process can be performed locally, incorporating the definition of the kernel size (k). The kernel size determines the weighting assigned to the vicinity of a particular point. For a more comprehensive discussion on the implications of choosing smaller or larger kernel sizes, as well as the optimal layer for performing recalibration, please refer to the study by Torres (2023). In summary, the choice of k size involves a trade-off between the level of refinement desired in capturing local biases and the tolerable Monte Carlo error. Additionally, selecting a specific layer for recalibration can serve as a strategy to address the challenges of dimensionality and employ a suitable representation of the data.

To implement this method, a recalibration set of size n is required, which can be the validation set itself. From this set, a sample of size k observations will be selected for *each new* observation we want to predict. It is important to note that for these new observations, no true outcomes were observed. Once the sample of size k is obtained, we proceed with the calculation of the predicted values for these new observations in the layer of interest. Subsequently, we calculate the distance between these new predictions and each of the *n* predictions obtained from the observations in the recalibration/validation set.

The samples described above are the *k*-nearest neighbors from *n* outputs for each prediction generated by the new *m* observations. The selection of neighbors is based on a distance measure, such as the Euclidean norm, and a weight that is assigned to each neighbour proportional to the distance. As a result, for each observation in the new (test) set, we now have *k* corresponding observations from the validation set, forming a sample of size *k* for each of the *m* new observations. Consequently, we have *k* predictions, *k* true outcomes (which are available for the validation set), and *k* Probability Integral Transform (PIT) values associated with each of these new *m* observations.

Simultaneously, the *m* observations are each associated with an estimated cumulative distribution $\hat{F}$, in our context, the Normal distribution. By applying the inverse quantile function $\hat{F}^{-1}(U)$, where U represents samples from a uniform distribution, one can obtain samples from $\hat{F}$ itself, as per the Inverse Transformation Theorem. However, instead of using an uniform distribution, one can assign non-uniform values within the [0,1] range to generate samples from a distribution different from $\hat{F}$.

In this context, we are specifically interested in incorporating the selected *k* PIT values mentioned earlier, which would only be uniformly distributed if the model were already calibrated, as discussed previously. Consequently, as a result, *k* $\tilde{y}$ samples from a distinct $\tilde{F}$ distribution are generated. These samples are designed to be from a better-calibrated model. Intuitively, we can infer that these non-uniform PIT values shift the predictions precisely to their intended positions, compensating to some extent for any undesired behavior. A comprehensive algorithm outlining the entire procedure is provided below, adapted from Torres (2023) (Algorithm 1).

Algorithm 1 Torres *et al.* (2023) method implemented in the package

```
Input:
 1:
      • Recalibration set, ${y^{(i)}_{\text{rec}}, \mathbf{x}^{(i)}_{\text{rec}}\}^n_{i=1}$, and new set, $\{\mathbf{x}^{(j)}_{\text{new}}\}^m_{j=1}$.
      • A neural network and its associated predictive distribution, $\hat{F}(\cdot \mid \mathbf{X})$.
      • A positive integer $l$ defining the network's layer where the samples are to be compared.
      • Neural network's outputs of the $l$-th layer on the recalibration set, $\{\mathbf{h}^{(i)}_{\text{rec}}\}^n_{i=1}$.
      • A smoothing kernel $K_u(d)$ with scale parameter $u > 0$, which may be defined indirectly from a positive integer $k$ that represents the number of observations to be used for recalibration.
Cumulative probabilities (PIT-values)
 2: for $i \leftarrow 1$ to $n$ do
 3:     Set $p^{(i)}_{\text{rec}} = \hat{F}_i(y^{(i)}_{\text{rec}} | \mathbf{x}^{(i)}_{\text{rec}})$.
 4: end for
Recalibration
 5: for $j \leftarrow 1$ to $m$ do
 6:     Compute $\mathbf{h}^{(j)}_{\text{new}} = g(\mathbf{x}^{(j)}_{\text{new}})$, where $g$ denotes the network's mapping to the $l$-th layer.
 7:     Apply the approximate KNN search method to identify the set of indices, $I_j$, corresponding to the observations in $\{y^{(i)}_{\text{rec}}, \mathbf{x}^{(i)}_{\text{rec}}\}^n_{i=1}$ for which $\|\mathbf{h}^{(i)}_{\text{rec}} - \mathbf{h}^{(j)}_{\text{new}}\|$ are within the $k$-smallest values.
 8:     for $i \in I_j$ do
 9:         Set $\tilde{y}^{(j)}_i = \hat{F}^{-1}_j(p^{(i)}_{\text{rec}} | \mathbf{x}^{(j)}_{\text{new}})$ and assign it a weight $w^{(j)}_i \propto K_u(\|\mathbf{h}^{(i)}_{\text{rec}} - \mathbf{h}^{(j)}_{\text{new}}\|)$.
10:     end for
11: end for
Output:
12: A set of $k$ weighted samples $\{(\tilde{y}^{(j)}_i, w^{(j)}_i)\}$ from the recalibrated predictive distribution
        $\tilde{F}_j(\cdot \mid \mathbf{x}^{(j)}_{\text{new}})$,
    for $j = 1, \ldots, m$.
```

As an output, for each observation in the test set, you would now have *k* pairs of corrected $\tilde{y}$ outputs, along with their respective weights *w* at that location. With this information, you can calculate the weighted averages, variances, and corresponding intervals. The study by Torres (2023) further demonstrates that this procedure consistently improves model metrics across various scenarios.

3 Methodology and Results

In our project, we adhered to the standards outlined by Wickham (2015) for R package development in RStudio. The initial version of our package is now publicly accessible on our GitHub repository.

The package can be installed and loaded using the following code:

```r
if (!require("pacman")) install.packages("pacman")
pacman::p_load_current_gh("cmusso86/recalibratiNN")
```

At present, the package comprises seven functions, each designed to accept only numeric vectors or matrices for the regressor(s) and response variables in separate objects. These functions do not support data frames and tibbles. The package relies on 10 dependencies: the `stats` package, six from the `tidyverse` suite (`dplyr`, `ggplot2`, `purrr`, `tidyr`, `magrittr`, and `tibble`), `RANN` for implementing the Approximate KNN method, `Hmisc` for weighted variance calculations, and `glue` for enhanced customization options in visualizations.

The functions are broadly compatible with Gaussian models, encompassing linear models, neural networks, and others, as long as they adhere to these specified assumptions. This includes linear models adjusted by least squares, neural network models trained using the least squares MSE loss function, or, equivalently, by maximizing the likelihood function of a normal distribution. Additionally, the adaptability of our approach allows for calibration across various representations of the covariate space, a feature particularly beneficial for neural networks.

The seven functions include two functions for calculating PIT Values (local and global) and four functions dedicated to visualizing miscalibration, employing two distinct visualization methods for both local and global diagnostics. Additionally, there is one function designed to carry out the recalibration process using the Torres (2023) method described above.

3.1 Calculation PIT-Values

The typical workflow for using this package begins with the calculation of PIT-values. To do this, users need a trained model, its predictive function, and a calibration set, which can be identical to the validation set. The first step is to calculate the PIT-values. In Gaussian models, this calculation is relatively straightforward. It involves using the inverse Cumulative

Density Function. This requires the actual output values, their predicted counterparts, and the Mean Squared Error (MSE) of the validation set. In R, the `qnorm()` function is typically used for this purpose. However, our package offers an equivalent, `PIT_global()`, which is based on `qnorm()` but differs in that it requires the MSE instead of the Standard Deviation from the calibration set. This ensures consistency of names within the package.

Below, we present an example where a heteroscedastic model is generated and then fitted using simple linear regression (as already illustrated in Figure 3). This serves as a practical demonstration for calculating PIT-values, offering an intuitive understanding of the analysis captured in the graphs. Additionally, this specific example is also provided in the package documentation for user reference.

```r
n <- 10000 # number of observations
split <- 0.8 # proportion for training

# Auxiliary functions
mu <- function(x1){
  10 + 5*x1^2
}
sg <- function(x1){
  30*x1
}

# Generating non-linear heteroscedastic data
set.seed(42)
x <- runif(n, 1, 10)
y <- rnorm(n, mu(x), sg(x))

# Train set
x_train <- x[1:(n*split)]
y_train <- y[1:(n*split)]

# Calibration/Validation set.
x_cal <- x[(n*split+1):n]
y_cal <- y[(n*split+1):n]

# New observations or the test set.
```

```
x_new <- runif(n/5, 1, 10)

# Fitting a simple linear regression,
# which will not capture the heteroscedasticity
model <- lm(y_train ~ x_train)

# predictions and mse of the calibration set
y_hat_cal <- predict(model,
                     newdata = data.frame(x_train=x_cal))
MSE_cal <- mean((y_hat_cal - y_cal)^2)
```

Now, we have all the information needed to calculate the PIT-values. The Global PIT-values can be obtained by:

```
pit <- PIT_global(ycal = y_cal, # true values from calib. set.
                  yhat = y_hat_cal, # predictions for calb. set.
                  mse  = MSE_cal) # MSE from calibration set.

head(pit, 5) # observe the first values.
```

```
## [1] 0.04551257 0.42522358 0.81439164 0.69119416 0.44043239
```

As printed above, this function returns a numeric vector of (probably uncalibrated) PIT-values, that will be used as input in other functions of the package.

The package also provides a way of calculating local PIT-values, named `PIT_local()`. In this case, the user must provide other arguments for the calculation. This function is designed to initially use a K-means algorithm to segment the design matrix into clusters, calculating the centroid of these groups. By default, it partitions the data into six clusters, but this number can be adjusted via the `clusters` argument. In the context of a simple linear regression, this partitioning results in six distinct, ordered subsets along the x-axis. This idea will enhance the comprehension of the example we are discussing.

After clustering, the function identifies a specific number of neighbors for each centroid. By default, it selects neighbors comprising 10% (a proportion of 0.1) of the calibration set, though this proportion can be modified using the `p_neighbours` parameter. The nearest neighbors are determined using a K-Nearest Neighbors (K-NN) method. If `p_neighbours=1` is chosen, the outcome will be equivalent to a global evaluation.

Subsequently, the function calculates the PIT-values for these neighbors using the `PIT_global()` function mentioned earlier. This approach allows for flexibility in the evaluation process, especially if other distributions are implemented in the future, as these can be incorporated as auxiliary parameters into this function.

This function ultimately produces a data frame encompassing the partition name, the computed PIT-values, and other crucial information that supplies the visualization functions with the necessary elements.

```r
local_pit <- PIT_local(xcal = x_cal,          # design matrix
                       ycal = y_cal,          # true output
                       yhat = y_hat_cal,      # predictions
                       mse  = MSE_cal,        # calibration mse
                       clusters = 4,          # k-means clusters
                       p_neighbours = 0.5)# neighbours prop.

local_pit %>%
  group_by(part) %>%   # to observe all parts
  sample_n(1)          # one observation each
```

```
## # A tibble: 4 x 5
## # Groups:   part [4]
##   part    y_cal y_hat   pit     n
##   <glue> <dbl> <dbl> <dbl> <dbl>
## 1 part_1  18.5  75.6 0.374  1000
## 2 part_2 205.  217.  0.475  1000
## 3 part_3  54.8 188.  0.228  1000
## 4 part_4  88.1 192.  0.279  1000
```

3.2 Visualizing miscalibration

The package offers two distinct types of graphs: density functions for the PIT-values and a line graph comparing theoretical and empirical cumulative probabilities. Both graph types can be applied on a local or global scale.

3.2.1 Global miscalibration

Global miscalibration can be analyzed using the `gg_PIT_global()` or `gg_CD_global()` functions. The `gg_PIT_global()` function offers a feature to calculate a p-value for hypothesis testing, which evaluates the conformity of the PIT-Values with a uniform distribution. The essential input for this function is the global PIT-values, which should be derived using the `PIT_global` function. Alternatively, users can input a compatible vector of PIT-values obtained through different means.

The default visualization returns a density graph. The function offers a range of additional customization arguments. Users can tailor graphical aspects such as fill color, transparency, the inclusion of p-values, and the specific type of plot according to their preferences. Examples of these customization are demonstrated in Figure **??**.

```r
# Default
gg_PIT_global(pit)

# Customizing parameters

gg_PIT_global(pit    = pit,          # from PIT_global
              fill   = "pink",       # desired color
              alpha = 1,             # opacity
              type   = "histogram",# graph type
              print_p = F)           # print p-values
```

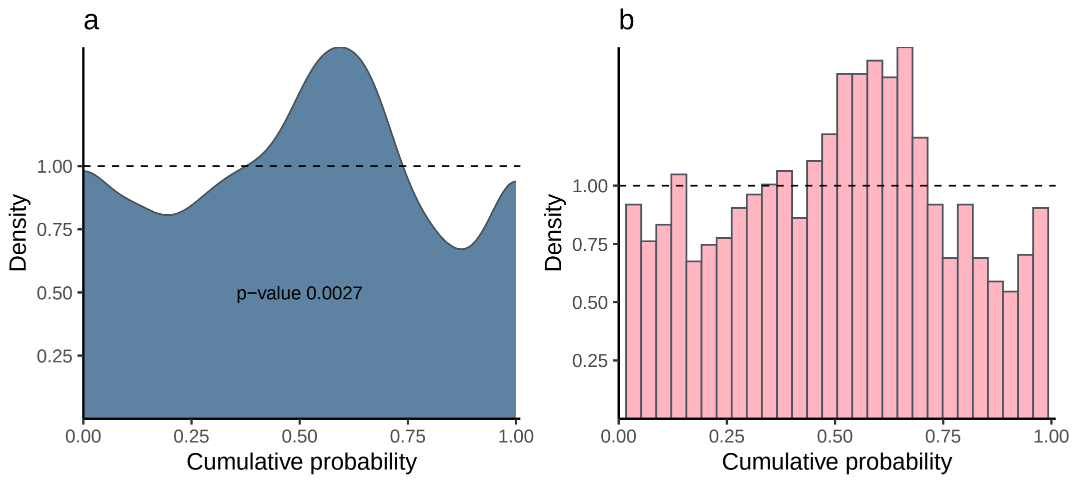

Figure 4: Density of global PIT-values. a) Default output. b) Customized parameters

In this scenario, it's evident that the distribution is not uniform, yet interpreting the pattern is not straightforward. As previously mentioned, the true model for this data was specifically designed to exhibit various types of miscalibration throughout the covariate space. Consequently, a global diagnosis alone is insufficient to capture these nuances.

Another method for visualizing global miscalibration is available through a different graph type within the package. Figure 5 displays a comparison of Predicted versus Empirical Cumulative Distribution. In this case, it is evident that a portion of the cumulative distribution is underestimated by the model, while another portion is overestimated.

In this method, we analyze the distribution of Probability Integral Transform (PIT) values in comparison to empirical distributions. To construct the dataset for our graph, we compute the PIT values for the recalibration dataset $[x_{rec}^{(i)}, y_{rec}^{(i)}]$ alongside the empirical distribution, which is essentially a sample proportion. Specifically, the PIT values are obtained using the Cumulative Distribution Function of a Normal distribution, applied to the observation values in the recalibration set $\vec{pit} = [F^{(i)}(y_{rec}^{(i)})]$. This calculation can be performed using the `pnorm()` function in base R or the `PIT_global()` function from a relevant package. The empirical distribution $\hat{P}^{(i)}(\vec{pit})$ is calculated as the proportion of `ycal` values that are less than or equal to the `qnorm()` quantile at each predicted point, that is, $\hat{P}^{(i)}(\vec{pit}) = [Mean(\mathbb{I}_{y_{rec} \leq F_i^{-1}(pit^{(i)})})]$, where $\mathbb{I}$ is an indicator function. For additional information, please refer to Kuleshov (2018).

The red dashed line shows where a perfect alignment of theoretilal and empirical distributions should lie.

```
gg_CD_global(pit   = pit,       # from PIT_global
             ycal  = y_cal,     # observation from calib. set
             yhat  = y_hat_cal,# predicted values of calib. set
```

```
            mse   = MSE_cal)   # MSE from calibration set
```

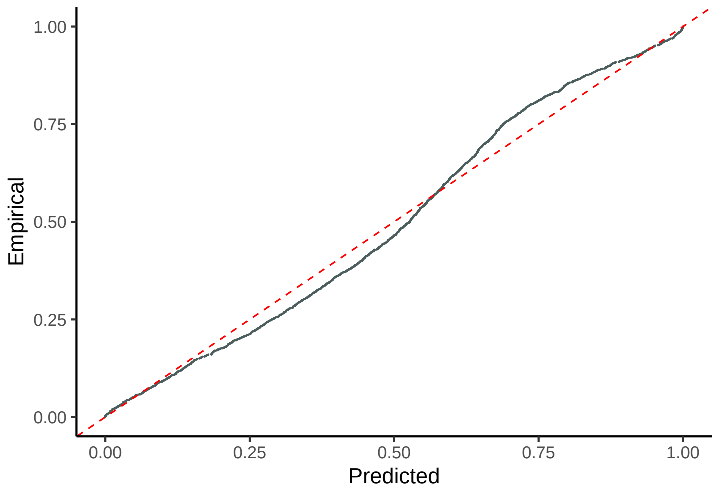

Figure 5: Predicted VS Empirical Cumulative Distribution.

3.2.2 Local miscalibration

The package additionally offers a method for diagnosing local miscalibration. For this purpose, users need to supply local PIT-values, which can be computed using the PIT_local() function previously described. An example of how to observe local densities is displayed in Figure 6.

```
gg_PIT_local(local_pit)

## Adding customization

gg_PIT_local(pit_local = local_pit,
             alpha     = 0,        # more transparency
             linewidth = 0.5,      # thinner lines
             pal       = "RdYlGn") # from RColorBrewer
```

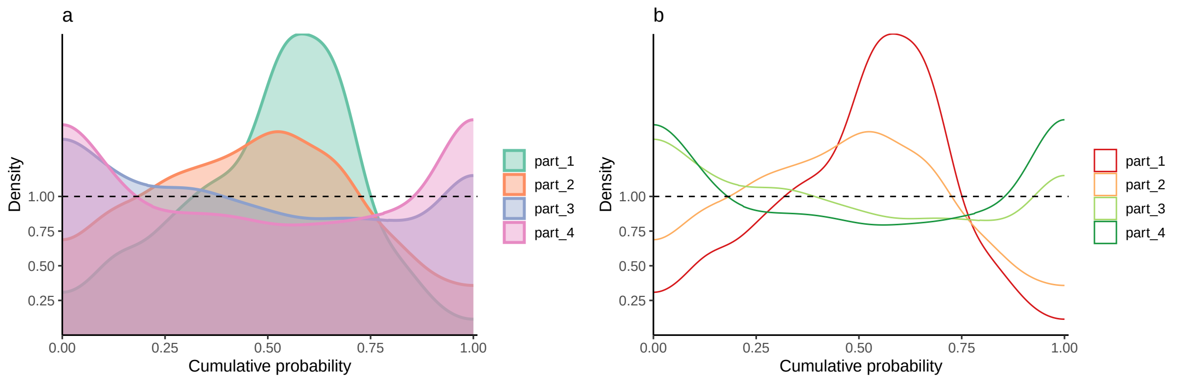

Figure 6: Local PIT-Values distribution

In this example, since the fitted model is a simple linear regression, part_1 corresponds to the lower range of x_cal values, whereas part_6 represents the higher range. It is observable that in the initial segments (part_1), the model tends to underestimate the mean while overestimating the variance. Conversely, in the final segment (part_6), although the model appears more accurately calibrated, it underestimates the variance. This behavior is consistent with the heteroscedastic nature inherent in the data when modeled using simple linear regression, as depicted in the Figure 3.

The graph will illustrate as many partitions as are determined by the local PIT-values calculated with the `PIT_local()` function. If fewer partitions are desired, it is necessary to carry out the partitioning process beforehand. By default, this visualization makes use of the

RColorBrewer ‘Set2’ color palette, set with an opacity level of 0.4 and a linewidth of 1, while omitting faceting. However, these parameters are adjustable to suit user preferences (Figure 7).

```
# Default
gg_CD_local(local_pit)

## Customized
gg_CD_local(local_pit,
            psz = 0.9,      # point size
            abline="red",   # color of the diagonal line
            pal="PiYG")     # palette from RColorBrewer
```

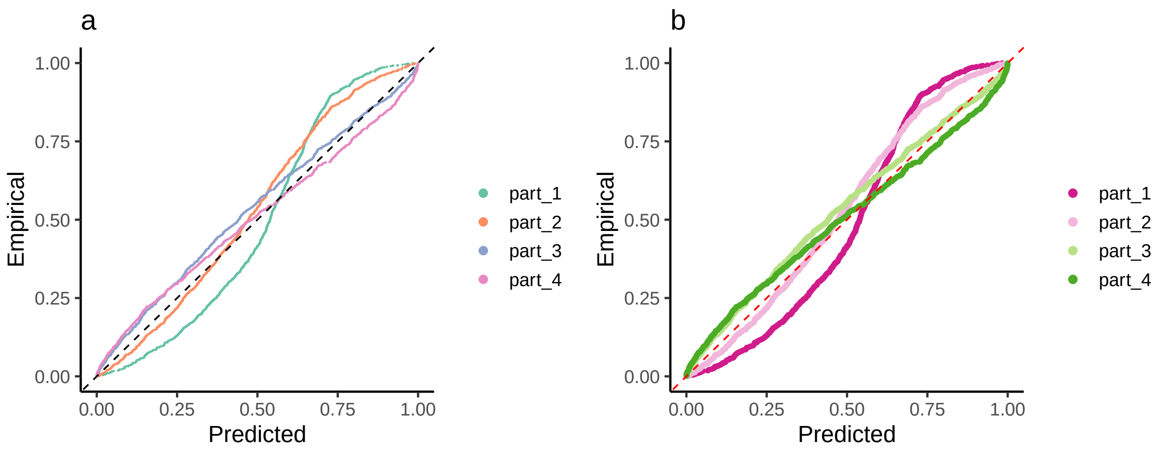

Figure 7: Predicted vs Empirical PIT-values distribution. a) Default, b) Customized.

Users have the option to view the curves individually by setting facet=T. When this option is selected for the PIT density graph, the ks.test() is executed , and the resulting p-value is displayed directly on the graph. Figure 8 illustrates this type of visualization. Various other customization options are also available, as detailed in the previously examples.

```
# Local PIT-values
gg_PIT_local(local_pit,
             facet=T) # facet the graph

# Empirical vs Predictive PIT-values
gg_CD_local(local_pit,
             facet=T) # facet the graph
```

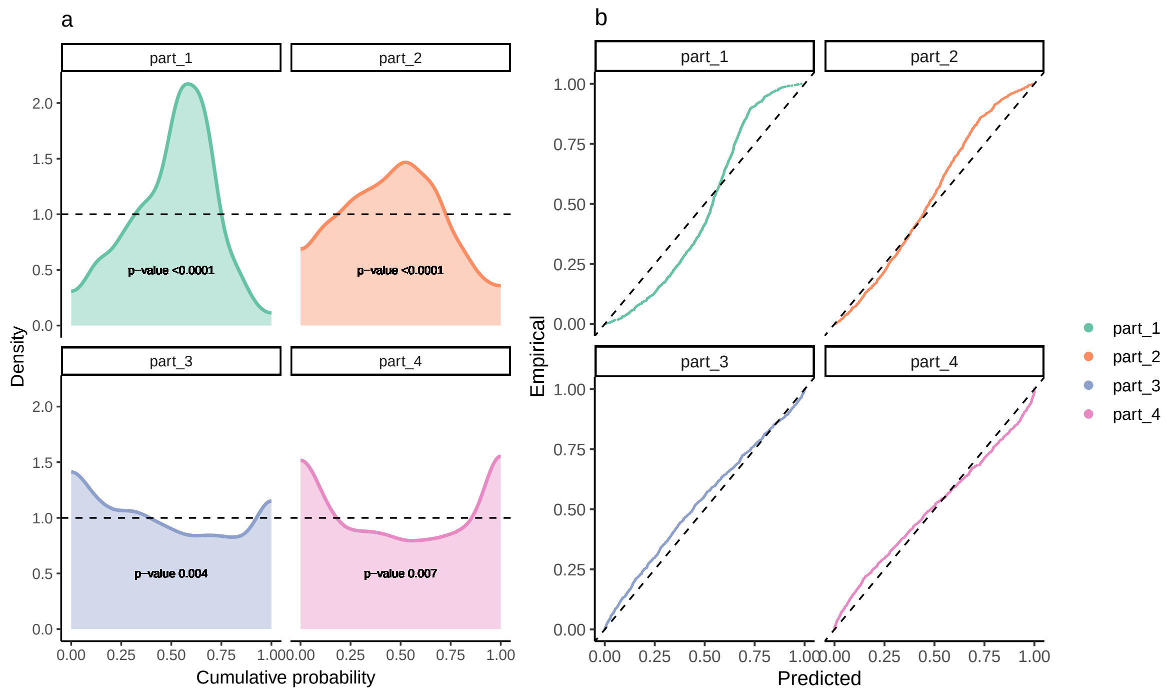

Figure 8: Local visualization using the facet=t option.

Once again, if a visualization with more or fewer clusters is desired, it is necessary to calculate the local PIT values for this scenario beforehand.

Upon examining these graphs, we reaffirm our previous observations. It appears that the central portion of the data is relatively well calibrated, whereas the beginning (part_1) exhibits poor calibration, echoing the issues of miscalibration discussed earlier.

We are confident that this array of visualization and customization options will not only prove beneficial for diagnostics but will also enable the presentation of miscalibration in intuitive and engaging ways whenever required.

For all visualizations, further customization can be done manipulating ggplot objects as usual (Figure 9).

```
graph <- gg_CD_local( pit    = local_pit,# from PIT_global
                       facet = T )


graph +
  labs(title = "Some title",
       subtitle = "subtitle",
       caption = "Source") +
  theme_classic(base_size = 16) +
scale_color_manual("Partition",
                   values=c("blue",
                            "aquamarine",
                            "steelblue",
                            "steelblue1")) +
facet_wrap(~ part,
           nrow = 1) +
theme(strip.text.x =
        element_text(
          size = 12, color = "red",
          face = "bold.italic"),
      legend.position = "bottom",
      strip.background = element_rect(
   color = "black", fill = "steelblue",
   size = 1.5, linetype = "solid"
   ))
```

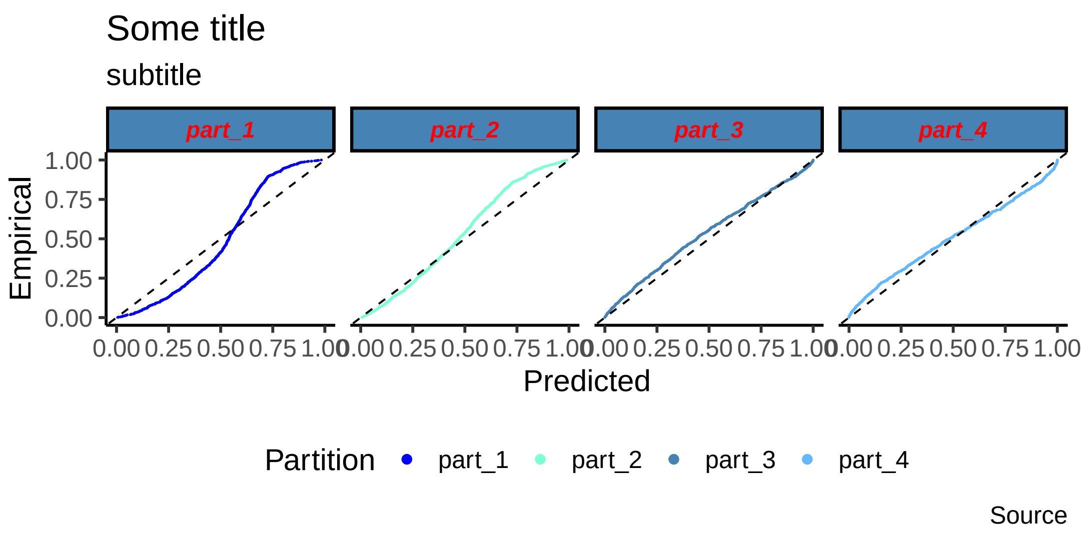

Figure 9: Further customization allowed as in any ggplot.

3.3 Recalibration

At last, the package offers a function, `recalibrate()`, for conducting the recalibration process itself. The method employed is detailed in Algorithm 1. For global recalibration, users can opt for a ‘standard’ approach, which assigns equal weight to each observation regardless of distance, offering a quicker process. Alternatively, one can perform a local calibration, but making use of 100% of the calibration data. In this approach, even though the calibration is less localized, weights are assigned based on the distance between points using an Epanichnikov kernel. Finally, smaller proportions of neighbours can be choosen. The smaller the proportion, the more localized the calibration will be; however, the Monte Carlo error will increase. This trade-off must be studied for each individual case.

3.3.1 Linear model

Let’s continue our analysis with the linear model applied to the heteroscedastic data we have been exploring. We will focus on understanding the main required arguments for the `recalibrate()` function. Firstly, we aim to recalibrate predictions for new observations, which lack actual outputs. Therefore, the primary argument this function requires is the model’s predictions for these new observations. Secondly, it’s necessary to provide the global PIT values generated by the PIT_global() function on the calibration set. This requirement holds true even when opting for local calibration, as the localization will be performed in a different way from that used for generation of pit_values for visualization. Finally, the function requires the Mean Squared Error of the calibration set.

In possession of those arguments, the user may proceed with a standard global calibration, seting the `type="global"` argument.

```r
y_hat_new <- predict(model,
                     newdata = data.frame(x_train = x_new))

recal_glob_1 <- recalibrate( yhat_new = y_hat_new,
                             pit_values = pit,
                             mse = MSE_cal,
                             type= "global")
```

This function generates a list containing several outputs: the revised predicted means, the updated predicted Mean Squared Error (MSE), and n new samples, each corresponding in length to `pit` vector, where *n* correspond to the size of `y_hat_new`.

In this hypothetical scenario, since we are aware of the process that generated the data, we can simulate the true values of `y_new()`. This allows us to assess whether the global calibration had any impact on the empirical distribution of the PIT values. Figure 10 shows that the global calibration seems to have enhanced global PIT-values distribution toward uniformity, whereas, the local calibration improved, but remains poor.

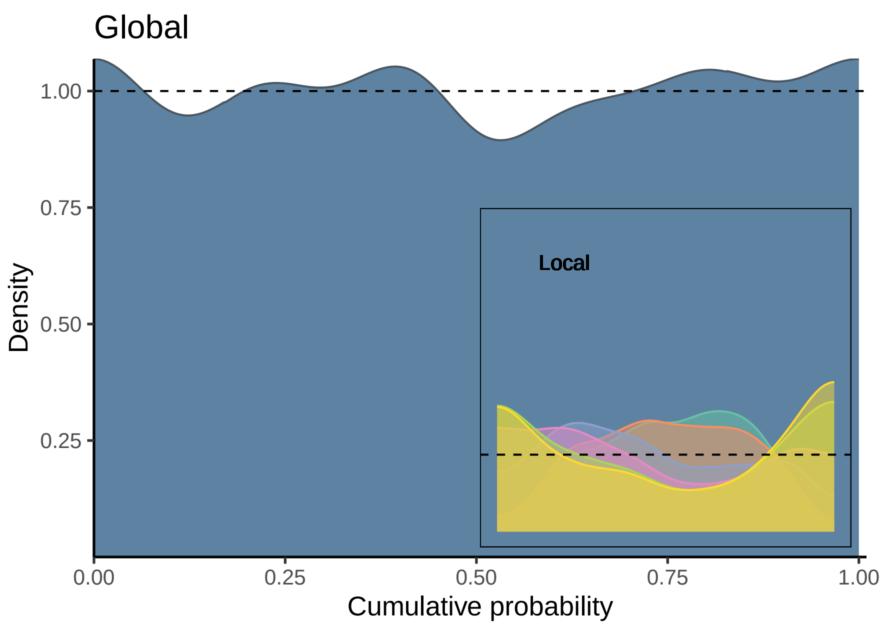

Figure 10: Global and Local PIT-values distribution analysed in the new (test) set, using the calibrated predictions.

Users also have the option to conduct a ‘local’ calibration using the entirety of the calibration set. This approach requires additional parameters. Initially, the function requires specification of the space within which the Aproximate K-Nearest Neighbors (KNN) method will operate to locate neighbors. It is crucial that the calibration space be coherent with the new data space. That being met, the assigned values may consist any representation of the covariate space, including intermediate layers or even the output. In this example, we will use the covariate space itself. The default proportion of neighbors is set at 0.1, but for this instance, we adjust it to 1 to perform a less localized calibration.

```
recal_glob_2 <- recalibrate(
  yhat_new = y_hat_new,
  pit_values = pit,
  mse = MSE_cal,
  type = "local",
  space_cal = x_cal,
  space_new = x_new,
  p_neighbours = 1)
```

In this scenario, the function additionally returns the weights for each sample. Moreover, we provide the weighted sample, which already incorporates these weights, along with the raw samples. These can be used with the weights for any task deemed necessary by the user. Nevertheless, the outcome is akin to what was previously observed, showing some improvement (Figure 11).

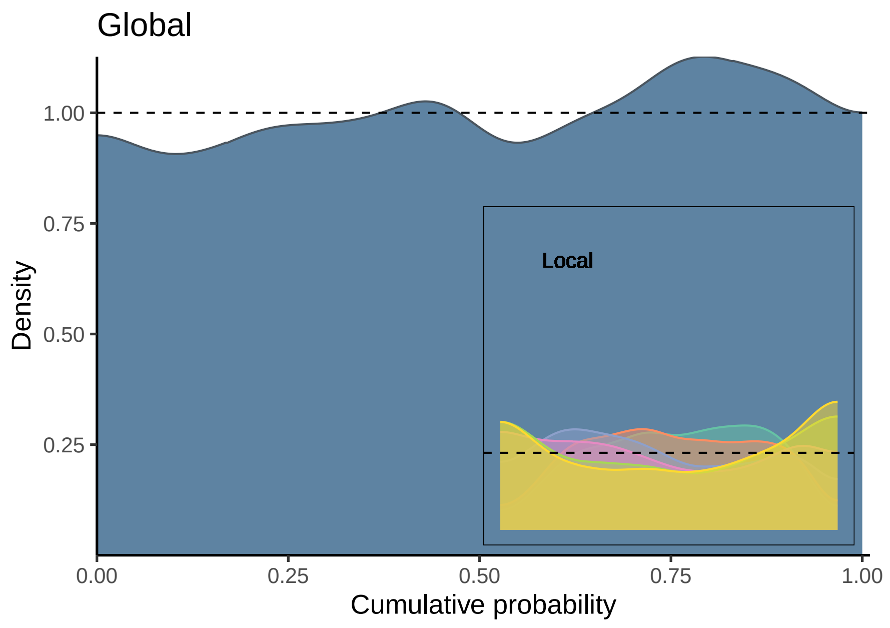

Figure 11: Global and Local PIT-values distribution using local calibration with propotion of calibration = 1.

We observe that both global methods seemingly achieve a uniform PIT-value distribution on a global scale, yet they both inadequately address local miscalibration issues. Upon comparing these methods, it becomes apparent that the use of global calibration with weights only offers a slight enhancement in mitigating this shortfall. Furthermore, it's crucial to highlight that the global calibration strategy, particularly when it includes weights, demands substantially more computational resources.

Finally, let's explore the impact of a "proper" local calibration in this example. To achieve this, we need to specify `type=local` and select a proportion of neighbors less than 1. Below, the function is implemented with the proportion of 0.2 (representing 20% of the calibration set). We also will consider an acceptable error bound of 0.1 in the Aproximate KNN method, provided by the `epsilon=0.1` parameter. The default is set to 0.0, which implies exact nearest neighbour search. Therefore, if nothing is specified, the function will calculate the exact distance, which is the default, as it was performed in the previous example. The function's output, as described previously, includes weights along with both raw and weighted samples. When applied to the new set, used here as a test set, we notice a marked improvement in rectifying local miscalibration (Figure 12).

```r
recal_loc <- recalibrate(
  yhat_new = y_hat_new,
  pit_values = pit,
  mse = MSE_cal,
  type = "local",
  space_cal = x_cal,
  space_new = x_new,
  p_neighbours = 0.4, # neighbours proportion
  epsilon = 0.1) # error bound in KNN
```

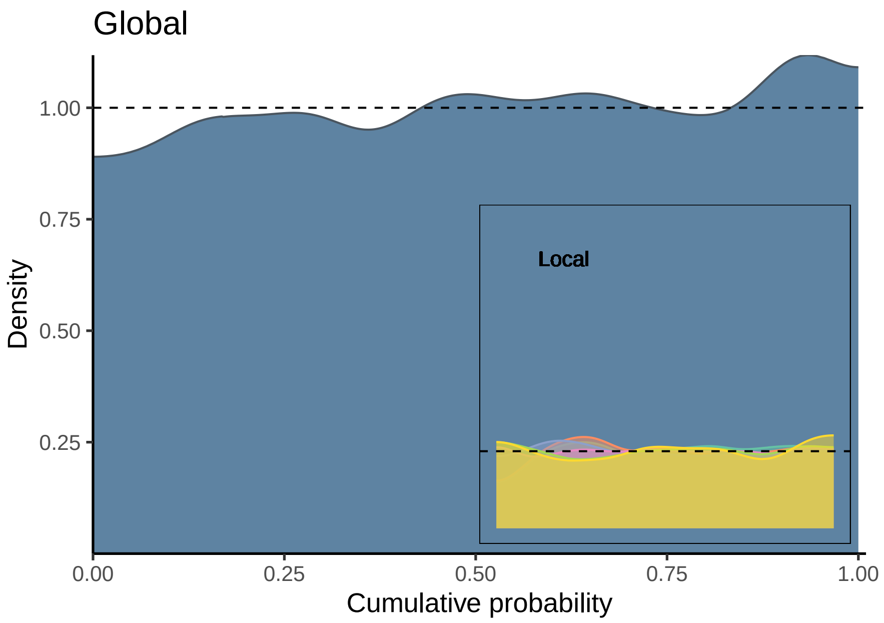

Figure 12: Local PIT-values distribution using local claubration with propotion of calibration = 1.

Indeed, the coverage across the dataset has improved, and the model now more accurately predicts the actual mean (Figure 13). For additional examples demonstrating the effectiveness of this method, please refer to Torres (2023). In this document, we have replicated the main results to verify the correct implementation of the method.

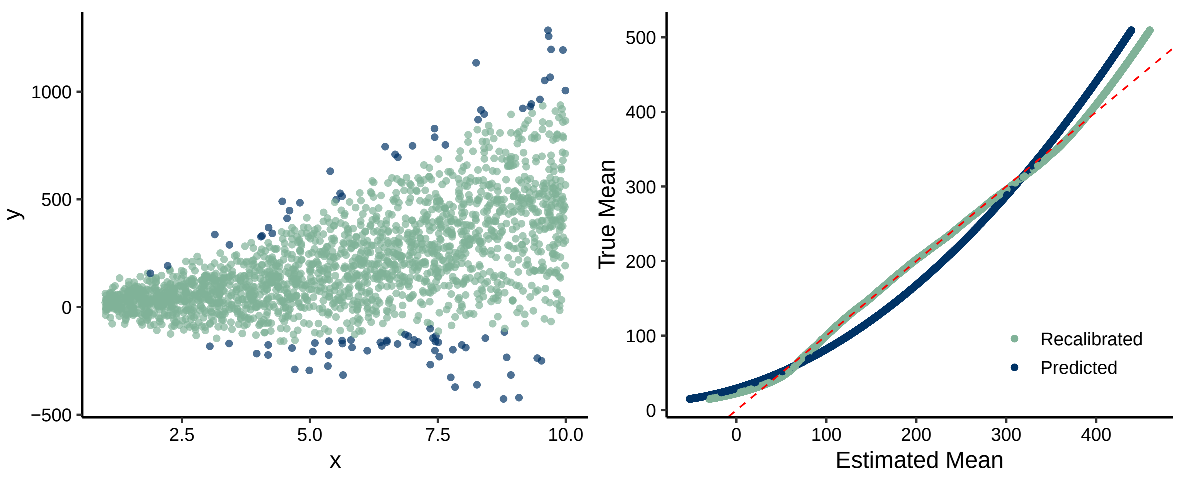

Figure 13: Coverage of recalibrated samples. True vc recalibrated Mean

3.3.2 Neural network

The package's name indicates its intended use, which we will demonstrate below. As discussed previously, we will also show the flexibility of our approach that allows for calibration across various representations of the covariate space. This is particularly useful for ANN, for example, in cases of high-dimensional covariate space, calibrating in a lower-dimensional layer may be more efficient. We will demonstrate this using the same homoscedastic model as before, but now fitted with a simple neural network. The recalibration function primarily relies on vectors of predictions or representations, which can be conveniently exported and imported as necessary. Therefore, it can be applied in numeric vectors provenient from any model.

Below, we provide the code and architecture for fitting a neural network in the same data generated before, along with instructions on how to export the prediction vectors. Through this, we aim to demonstrate the agnosticism and flexibility of these functions as a post-processing method. One could choose, for example to fit the model outside of R, and then import the vectors to proceed with the recalibration.

In this example, we utilized an ANN comprising two layers. The first layer features 200 neurons with a sigmoid activation function, while the output layer employs a linear activation. To keep the example straightforward, we did not incorporate dropout or batch normalization. The network was optimized using the Adam optimizer with a learning rate of 0.1, and we employed Mean Squared Error (MSE) as the loss function. because of that, we are implicitly assuming the output follows a Normal Distribution.

```r
# creating architeture
model <- keras_model_sequential() # simple MLP

model %>%
  layer_dense(input_shape = 1, # one covariate
              units = 200,      # 200 neuron
              activation = "sigmoid") %>% # non-linearity
  layer_dense(units = 1, # output
              activation = "linear") # regression

# compiling model
model %>%
  compile(optimizer = optimizer_adam(
    learning_rate = 0.01),
    loss="mse") # normal distribution

# fitting model
model %>%
  fit(x = x_train, y = y_train,
      validation_data = list(x_cal, y_cal),
      callbacks = callback_early_stopping(
        monitor = "val_loss",
        patience = 20,
        restore_best_weights = T),
      batch_size = split*n, # batch size
      epochs = 500)

# predicting output values
y_hat_cal <- predict(model, x_cal) # predictions cal
y_hat_new <- predict(model, x_new) # predictions new


# predicition intermediate layer
layer_name <- paste('dense', 1, sep = '_')
layer_model <- keras_model(
  inputs = model$input,
  outputs = get_layer(model, layer_name)$output
)

h_new <- layer_model %>% predict(x_new)
h_cal <- layer_model %>% predict(x_cal)



# MSE
MSE_cal <-model %>%
  evaluate(x_cal, y_cal)
MSE_cal <-as.numeric(MSE_cal)

# export vectors (optional)
saveRDS(y_hat_cal, file="y_hat_cal.rda")
saveRDS(y_hat_new, file="y_hat_new.rda")
saveRDS(h_new, file="h_new.rda")
saveRDS(h_cal, file="h_cal.rda")
saveRDS(MSE_cal, file="MSE_cal.rda")
```

Then, you can import it into R:

```r
# load vectors
MSE_cal_nn <- readRDS(file = "MSE_cal.rda")
h_cal <- readRDS(file = "h_cal.rda")
h_new <- readRDS(file = "h_new.rda")

# transforming to numeric vector, as Keras returns a matrix.
y_hat_cal_nn <- as.numeric(readRDS(file = "y_hat_cal.rda"))
y_hat_new_nn <- as.numeric(readRDS(file = "y_hat_new.rda"))
```

We notice this model is also poorly calibrated (Figure 14). So we choose to proceed with the recalibration of the Neural Network using the covariate space as the basis for the search of neighbours. Consequently the model appears more well calibrated (Figure 14).

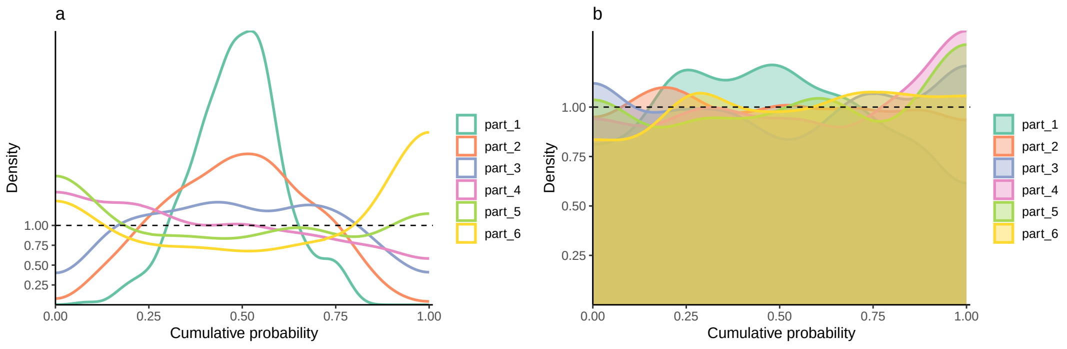

Figure 14: PIT-values distribution in the a) uncalibrated ANN and b) recalibrated.

We can recalibrate using a different space, although in this specific example, it's not particularly beneficial due to the intermediate layer having more dimensions than the covariate space. Nevertheless, this serves as a useful illustration that recalibration is effective for higher-dimension matrices, not just for the univariate models we've demonstrated thus far.

```
recal_NN_h <- recalibrate(
 yhat_new = y_hat_new_nn,
 pit_values = pit,
 mse= MSE_cal_nn,
 space_cal = h_cal, # matrix of representations
 space_new = h_new, # matrix of representations
 type="local"
)
```

Finally, we demonstrate that the recalibrated ANN achieves good coverage across observations. It also predicts the mean more accurately, even without calibration, compared to linear regression. This is anticipated, given the neural network's superior ability to capture non-linear relationships. Moreover, we observe an enhancement in mean prediction accuracy after calibration, as shown in Figure 15.

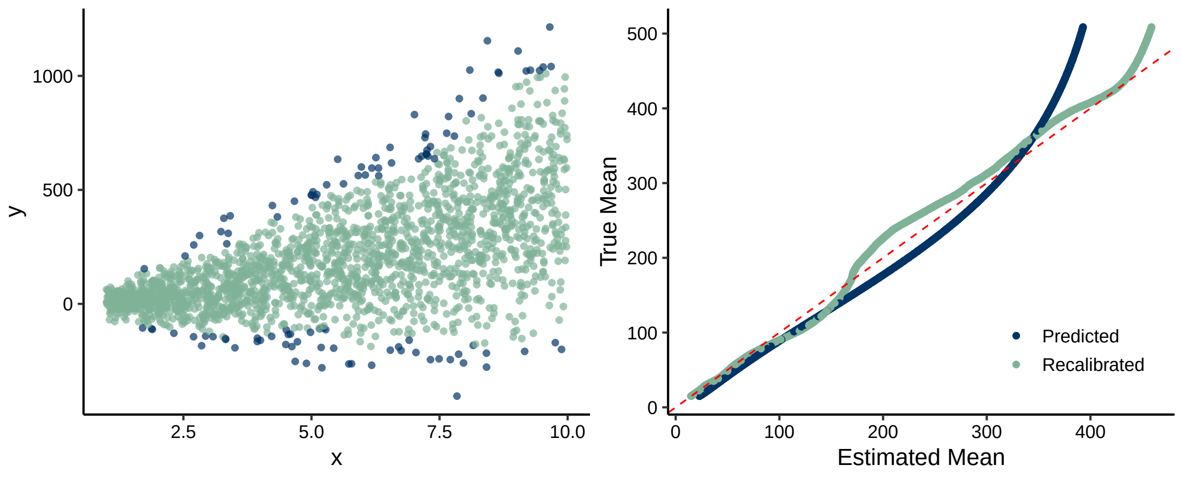

Figure 15: Coverage of the IC and prediction of mean in the test (new) set after recalibration of ANN in the covariate space.

4 Conclusions and further work

This work delves into the recalibration of Gaussian Models, particularly in the application of Neural Networks (ANNs), leading to the development of the recalibratiNN package in R. This is crucial, as despite the high prediction performance of ANNs, they often suffer from miscalibration, which impacts their reliability in uncertainty estimation. The recalibratiNN package addresses this issue by providing a computational implementation of a quantile-based post-processing technique to recalibrate Gaussian models, like neural networks that used the MSE loss function.

This research contributes to the statistical community by providing a user-friendly package for identifying miscalibration and proceeding with recalibration. Therefore, the recalibratiNN package contributes to the field by enabling more reliable uncertainty estimates and encouraging calibration practices.

The functions in this study effectively visualize miscalibration, and Torres' method has been accurately implemented, yielding predictions that are more calibrated than the original models, particularly with local calibration. The algorithm improves calibration by using the diagnostic capabilities of Probability Integral Transform (PIT) values histograms to identify distinct bias patterns. This insight informs the creation of Monte Carlo samples for a more calibrated distribution, thus enhancing model performance and reliability, notably in neural networks, by recalibrating at an arbitrary network layer.

For global calibration, using a uniform kernel has expedited recalibration, yielding results comparable to those obtained with an global Epanichnikov kernel. Local recalibration, particularly with the heteroscedastic data in this study, proves more beneficial, with only 10%-20% of the calibration set being sufficient. The localization employs an efficient algorithm, the approximate K-Nearest Neighbours. This method of local calibration, combined with the capability of recalibrating at intermediate layers rather than the original input-space, is a distinctive feature not typically found in current R or Python packages.

Overall, this study highlights the crucial role of calibration in Artificial Neural Networks (ANNs) and the effectiveness of the recalibratiNN package in addressing these issues. Future improvements would benefit from better integration with other packages, including support for a broader range of input types, compatibility with cross-validation methods like k-fold or bootstrap, hyperparameter tuning, and the ability to handle models with various predictive distributions, not just those limited to the Gaussian family.

References

ALLAIRE, J.; CHOLLET, F. *keras: R Interface to 'Keras'*. [S.l.], 2023. R package version 2.11.1. Disponível em: <https://CRAN.R-project.org/package=keras>.

BECK, M. W. Neuralnettools: Visualization and analysis tools for neural networks. *Journal of statistical software*, NIH Public Access, v. 85, n. 11, p. 1, 2018.

BERGMEIR, C.; BENíTEZ, J. M. Neural networks in r using the stuttgart neural network simulator: Rsnns. *Journal of Statistical Software*, v. 46, n. 7, p. 1–26, 2012. Disponível em: <https://www.jstatsoft.org/index.php/jss/article/view/v046i07>.

ENGELHARDT, M.; BAIN, L. J. Statistical analysis of a compound power-law model for repairable systems. *IEEE Transactions on Reliability*, R-36, n. 4, p. 392–396, 1987.

FALBEL, D.; LURASCHI, J. *torch: Tensors and Neural Networks with 'GPU' Acceleration*. [S.l.], 2023. R package version 0.10.0. Disponível em: <https://CRAN.R-project.org/package=torch>.

GNEITING, T.; RAFTERY, A. E. Weather forecasting with ensemble methods. *Science*, v. 310, n. 5746, p. 248–249, 2005. Disponível em: <https://www.science.org/doi/abs/10.1126/science.1115255>.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>.

GÜNTHER, F.; FRITSCH, S. neuralnet: Training of neural networks. *R J.*, v. 2, p. 30, 2010.

GUO, C. et al. On calibration of modern neural networks. In: PMLR. *International conference on machine learning*. [S.l.], 2017. p. 1321–1330.

KUHN, M.; VAUGHAN, D.; RUIZ, E. *probably: Tools for Post-Processing Class Probability Estimates*. [S.l.], 2023. R package version 1.0.2. Disponível em: <https://CRAN.R-project.org/package=probably>.

KULESHOV, V.; DESHPANDE, S. Calibrated and sharp uncertainties in deep learning via density estimation. In: *International Conference on Machine Learning*. [S.l.: s.n.], 2021.

KULESHOV, V.; FENNER, N.; ERMON, S. Accurate uncertainties for deep learning using calibrated regression. In: PMLR. *International conference on machine learning*. [S.l.], 2018. p. 2796–2804.

LAKSHMINARAYANAN, B.; PRITZEL, A.; BLUNDELL, C. *Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles*. 2017.

LUCENA, B. Spline-based probability calibration. *arXiv preprint arXiv:1809.07751*, 2018.

RODRIGUES, G.; PRANGLE, D.; SISSON, S. Recalibration: A post-processing method for approximate bayesian computation. *Computational Statistics and Data Analysis*, v. 126, p. 53–66, 2018. ISSN 0167-9473. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167947318300859>.

TORRES, R. *Quantile-based Recalibration of Artificial Neural Networks*. Dissertação (Mestrado) — University of Brasília, Distrito Federal, Brazil, 2023.

WICKHAM, H.; BRYAN, J. *R packages: organize, test, document, and share your code*. O'Reilly Media, Inc., 2015. Disponível em: <https://r-pkgs.org>.

XIONG, M. et al. *Proximity-Informed Calibration for Deep Neural Networks*. 2023.

Appendices

Package: recalibratiNN

Title: Quantile Recalibration For Regression Models

Version: 0.0.0.9000

Authors@R: c(person(given = "Carolina", family = "Musso", role = c("aut", "cre"), email = "cmusso86@gmail.com", comment = c(ORCID = "0000-0002-8107-6458")), person(given = "Guilherme", family = "Rodrigues", role = c("ctb","ths"), email = "guilhermerodrigues@unb.br", comment = c(ORCID = "0000-0003-2009-4844")), person(given = "Ricardo", family = "Torres", role = c("ctb"), email = "ricardotbreis@gmail.com", comment = c(ORCID = "0009-0000-9100-7125”)), person(given = "João", family = "Reis", role = c("ctb"), email = "j.rodriguesreis@gmail.com"))

Description: The recalibratiNN R package empowers users to enhance the calibration of regression models. It offers global and local calibration visualization tools and provides two distinct recalibration methods: one inspired by Approximate Bayesian Computation (ABC) and the other based on isotonic regression. These methods enable users to adjust model predictions, improving alignment with observed data. By allowing for global and local recalibration, the package ensures precise adjustments tailored to specific data subsets or regions of interest. With recalibratiNN, users can increase the reliability and accuracy of their regression models, making them better suited for real-world applications.

License: MIT + file LICENSE

Encoding: UTF-8

Roxygen: list(markdown = TRUE)

RoxygenNote: 7.2.3

URL: https://github.com/cmusso86/recalibratiNN

BugReports: https://github.com/cmusso86/recalibratiNN/issues

Imports: dplyr, ggplot2, purrr, RANN, stats, tidyr, tibble, glue, magrittr, Hmisc

Suggests: testthat (>= 3.0.0)

Config/testthat/edition: 3

RemoteType: github

RemoteHost: api.github.com

RemoteRepo: recalibratiNN

RemoteUsername: cmusso86

RemoteRef: HEAD

RemoteSha: 88d5c6b9bebae35526fec32fc27f91976393957c

GithubRepo: recalibratiNN

GithubUsername: cmusso86

GithubRef: HEAD

GithubSHA1: 88d5c6b9bebae35526fec32fc27f91976393957c

NeedsCompilation: no

Packaged: 2023-12-22 19:19:13 UTC; carolinamusso

Author: Carolina Musso [aut, cre] (<https://orcid.org/0000-0002-8107-6458>), Guilherme Rodrigues [ctb, ths] (<https://orcid.org/0000-0003-2009-4844>), Ricardo Torres [ctb] (<https://orcid.org 0000-9100-7125>), João Reis [ctb]

Maintainer: Carolina Musso <cmusso86@gmail.com>

Built: R 4.3.1; ; 2023-12-22 19:19:13 UTC; unix