Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

RESEARCH ARTICLE

Evaluating tuning strategies for random forest hyperparameters with regards to prediction performance of clinical prediction models and computational time

J.A. Neve*[1,2]

[1]Department of Methodology and Statistics, Utrecht University, Utrecht, Netherlands
[2]Julius Centre, University Medical Centre Utrecht, Utrecht, Netherlands

Correspondence
*Corresponding author Email: j.a.nevedemevergnies@students.uu.nl

Write my abstract here - 250 words max

KEYWORDS:
Write; my; keywords; here - six max

1 | INTRODUCTION

Machine learning models are increasingly used in medicine for diagnostic and prognostic purposes[1]. Risk estimates are used to assist clinical decisions (e.g., whether to undergo treatment). A popular set of techniques in clinical research is tree-based methods[2], among which random forests are particularly prevalent[3,4]. Random forests' results are influenced by hyperparameters (e.g., the number of predictors sampled to make a split)[5], which need to be set before training a model. Previous studies have shown classification accuracy (i.e., the proportion of observations correctly classified) to have little-to-no improvement when using optimal hyperparameters for a given dataset compared to hyperparameter software defaults[6,7]. Classification accuracy is however not sufficient to evaluate a model's clinical utility: identifying a patient as positive or negative may not carry sufficient information, as the patient's predicted risk is a key element of medical decision-making. Moreover, classification accuracy depends on the chosen classification threshold, i.e., the predicted risk value above which an observation is classified as positive. For clinical purposes, discrimination (i.e., positive patients have higher risk predictions than negative patients) and calibration (i.e., risk predictions reflect patients' true risk)[8] are two much more important performance metrics. Models with low discrimination may misclassify patients, while miscalibrated models can lead to over- or undertreatment if over- or underestimating patients' risks. Random forests can heavily suffer from miscalibration[9,10].

A well-known way to improve model performance is to perform hyperparameter tuning, which identifies hyperparameter values for which a performance metric (e.g., deviance) is optimal for a given prediction model. This procedure requires choosing which metric is optimised, which hyperparameters are tuned, and how candidate hyperparameter values are generated. The latter two can greatly impact computational time: generally, the more hyperparameters are tuned or the more candidate values there are, the larger computational time is. Hyperparameter search algorithms may consider all possible hyperparameter values and their combinations, or use information collected earlier in the tuning procedure to tune efficiently. Computational time is important to consider in conjunction to the improvement in model performance: greater computational times and longer training periods of models may lead to a strain on resources and increase carbon emissions. For complex models, this can be as much as five times the average emissions of a car over its lifetime[11].

The optimal hyperparameter tuning strategy for balancing predictive performance and computational time is still an open question, particularly when considering clinically relevant performance metrics. Previous research suggests the most gain in random forest performance as measured by classification accuracy, discrimination, and the Brier score could be made by tuning the number of predictors sampled to make a split and the proportion of the sample used to fit the tree[12]. Yang and Shami[13]

found tuning using most hyperparameter search algorithms improved classification accuracy compared to default hyperparameters. Some search algorithms had greatly increased computational times without improving classification accuracy further than shorter search algorithms. So far, no study has compared different optimisation metrics' effects on model performance or computational time. Moreover, existing studies focusing on tuning procedures for random forests use high-dimensional datasets (datasets with more features than observations) whereas clinical research typically uses low-dimensional datasets (datasets with more observations than features), for which guidance regarding hyperparameter tuning is lacking [14].

The current project aims to identify the optimal hyperparameter tuning strategies for balancing predictive performance and computational time of random forests in low-dimensional settings. There are three simulation studies in this project to identify (i) which hyperparameters to tune, (ii) which metric to optimise, and (iii) which search algorithm to use for optimal model performance.

2 | METHODS

We ran three simulation studies in order to identify optimal tuning strategies in controlled conditions. The studies were done sequentially to integrate conclusions from prior studies. We simulated data on which to compare tuning strategies. In this methods section, we detail the simulation methodology for each study following the ADEMP approach [15].

2.1 | Aim

2.1.1 | Study 1: Hyperparameters to tune

Using datasets from the OpenML platform [16], Probst and colleagues [12] found the number of predictors considered at a split and the sample fraction to be the two most influential hyperparameters on the discriminative performance of prediction models. These findings only investigated the effect of tuning one or two hyperparameters at once. Our first study aimed to extend these findings by considering (i) model calibration in addition to model discrimination and (ii) more combinations of hyperparameters. We examined the effect of tuning different combinations of hyperparameters on the performance of a prediction model and evaluated performance improvements in context of required computational times.

2.1.2 | Study 2: Optimisation metric

This study aimed to identify a metric to optimise in the tuning procedure leading to the best predictive performance of a prediction model. We tuned the combination of hyperparameters considered optimal in Study 1.

2.1.3 | Study 3: Hyperparameter search algorithm

This study aimed to compare hyperparameter search algorithms' performances with respect to predictive performance and computational time. We tuned the combination of hyperparameters considered optimal in Study 1 using the optimisation metric selected in Study 2.

2.2 | Data-generating mechanism

All studies used the same data-generating mechanism. Different datasets were generated for each study.

2.2.1 | Data-generating scenarios

A full factorial simulation design was used to consider the influence of data characteristics on model predictive performance and computational time. The varying simulation factors were the number of candidate predictors p (range: 8, 16), the event fraction EF (range: 0.1, 0.3, 0.5), and the sample size n (range: $0.5N$, $1N$). N is the minimum sample size required to identify effects for a given number of regression coefficients (here, $1.25p$), expected event fraction, and expected AUC (here, 0.8), as calculated by Riley and colleagues [17]. A total of 12 (2*3*2) scenarios were considered. For each scenario, 500 training datasets were generated, yielding a total of 6,000 datasets per study. Scenario sample sizes are detailed in Table 1.

Training and validation datasets were simulated under a logistic model with strong interactions. For each observation i ($i = 1, \ldots, N$), predictors $\mathbf{x}_i$ were drawn from a p-variate normal distribution with mean $\boldsymbol{\mu} = \mathbf{0}$, each predictor's variance set to 1, and

TABLE 1 Data-generating scenarios

p	EF	n
8	0.10	381
16	0.10	762
8	0.30	172
16	0.30	344
8	0.50	193
16	0.50	293
8	0.10	762
16	0.10	1523
8	0.30	344
16	0.30	688
8	0.50	385
16	0.50	585

two-way predictor correlations set to 0.2. We included $0.25p$ two-way interactions, with the h^{th} ($h = 1, ..., 0.25p$) interaction being the product of the h^{th} and the $(h + p/4)^{th}$ predictors. The binary outcome y_i was drawn from a Bernoulli distribution conditional on the predictor values (Formula 1).

$$P(y_i = 1|\mathbf{x}_i) = \frac{1}{1 + exp(-(\beta_0 + \sum_{j=1}^{p/2} \beta x_{ij} + \sum_{k=1+p/2}^{3p/4} 2\beta x_{ik} + \sum_{h=1}^{0.25*p} \gamma x_{ih} x_{i(h+0.25p)}))}. \quad (1)$$

In Formula 1, β_0, β, γ are respectively the intercept, main effect regression coefficient, and interaction effect regression coefficient of the data generating model, hereafter called "true effects".

A validation dataset ($n = 100,000$ for study 1 and $n = 10,000$ for studies 2 and 3) to evaluate model performance was generated for each training dataset. The size of the validation dataset was reduced after study 1 due to storage constraints.

2.2.2 | True effect estimation

True effects were determined for each unique combination of p and EF. Let k denote a given combination. For each combination, the intercept $\beta_0^{(k)}$, predictor main effects $\boldsymbol{\beta}^{(k)}$, and predictor interaction effects $\boldsymbol{\gamma}^{(k)}$ were estimated using a large sample approximation ($n = 100,000$). The first $p/2$ main effects were set to be equal (i.e., $\beta_1^{(k)} = \beta_2^{(k)} = ... = \beta_{p/2}^{(k)}$). The next $p/4$ main effects were set to be twice the strength of the first $p/2$ main effects (i.e., $\beta_{p/2+1}^{(k)} = \beta_{p/2+2}^{(k)} = ... = \beta_{3p/4}^{(k)} = 2\beta_1^{(k)}$). The final $p/4$ main effects were set to 0. All interaction effects $\boldsymbol{\gamma}^{(k)}$ were set to be equal (i.e., $\gamma_1^{(k)} = \gamma_2^{(k)} = ... = \gamma_{0.25p}^{(k)}$). The R function `optim` was used to minimise a loss function measuring the sum of the squared difference between (i) 0.7 and the observed AUC in a model with no interactions, (ii) 0.8 and the observed AUC in a model with interactions, and (iii) the targeted event fraction and the average estimated probability $P(y_i = 1|\mathbf{x}_i)$ in the simulated dataset. This was repeated 20 times for each scenario. Each generated set of true effects was used to generate a dataset of $n = 100,000$, on which the AUC (with and without interaction) and prevalence were measured. For each scenario, the true effect was selected to be the set of effects with the smallest sum of absolute differences between (i) target prevalence and observed prevalence, (ii) 0.7 and the AUC of a model with no interactions, and (iii) 0.8 and the AUC of a model with interactions. True effects' performances were assessed by generating a new dataset and measuring AUC and prevalence (Table 2).

2.3 | Methods

2.3.1 | Study 1: Hyperparameters to tune

We varied which hyperparameters were tuned when fitting a random forest model using the R package `ranger`[18] via the R package `caret`[19]. We used grid search to optimise deviance. 5-fold cross-validation was used.

Considering all possible combinations of 5 hyperparameters leads to 32 tuning procedures on each dataset. Due to the unfeasibility of such a large-scale approach, we reduced the number of combinations studied: only hyperparameter combinations

TABLE 2 Performance of selected true effects

p	EF (target)	β_0	β	γ	AUC (no interaction)	AUC (interaction)	EF (obtained)
8	0.10	-2.98	0.14	0.66	0.71	0.80	0.10
8	0.30	-0.94	0.24	-0.84	0.70	0.80	0.30
8	0.50	-0.22	0.22	0.83	0.70	0.80	0.50
16	0.10	-3.06	0.07	0.44	0.70	0.80	0.10
16	0.30	-0.85	0.14	-0.55	0.70	0.80	0.30
16	0.50	-0.31	-0.13	0.55	0.70	0.80	0.50

TABLE 3 Hyperparameter tuning ranges in studies 1 and 2

Hyperparameter	Default	Range
`mtry`	$\sqrt{p}$ (rounded down)	1-p
`min.node.size`	1	1-10
`replace`	TRUE	TRUE, FALSE
`sample.fraction`	1	0.1, 0.2, ..., 0.9, 1
`splitrule`	gini	gini, hellinger, extratrees*

*For `splitrule` = extratrees, an additional parameter should be considered regarding the number of random splits to consider. This was set to its default of 1.

including at least `mtry` (the number of predictors randomly sampled to make a split) and `min.node.size` (the minimum number of observations for a node to be formed, i.e., the point at which a split should not be computed regardless of impurity) were considered. Other hyperparameters which could be included in combinations were `replace` (whether the data used to fit a single tree is sampled with or without replacement), `sample.fraction` (the proportion of the data used to fit a single tree), and `splitrule` (the way in which a split is picked). Default values and tuning ranges used in this study are presented in Table 3.

This yielded 8 hyperparameter combinations. Hyperparameters not included in a given combination were set to their default value. A model using the default hyperparameters was fit to establish the baseline. All considered combinations were used to fit a random forest on each training dataset.

2.3.2 | Study 2: Optimisation metric

We varied the metric to optimise when fitting a random forest using the R package `ranger`[18] via the R package `caret`[19]. Each dataset was tuned once using each candidate optimisation metric. We used grid search to tune the hyperparameters considered optimal in Study 1. Tuning ranges were identical to those of study 1 (Table 3). 5-fold cross-validation was used as part of the tuning procedure. The considered candidate optimisation metrics were the deviance (Formula 2; measuring the total deviations between predicted risk and the true outcome), the logarithmic loss (Formula 3; measuring the mean deviations between predicted risk and the true outcome, thus), the AUC (a measure of how well the classes are separated), the calibration intercept (the difference between the average predicted risk and the true event rate), the calibration slope (the extent of over- or underestimation of risk), the Brier score (Formula 4; this can be seen as a composite measure of calibration and discrimination[20]), the classification accuracy (the proportion of correctly classified observations) and Cohen's Kappa (the proportion of correctly classified observations, adjusted for chance). These are summarised in Table 4 alongside their target values.

$$\text{Deviance} = -2 * \sum_{i=1}^{n} (y_i * log(\hat{y}_i) + (1 - y_i) * log(1 - \hat{y}_i)). \tag{2}$$

$$\text{Logarithmic loss} = \frac{1}{n} \sum_{i=1}^{n} (y_i * log(\hat{y}_i) + (1 - y_i) * log(1 - \hat{y}_i)). \tag{3}$$

$$\text{Brier score} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2. \tag{4}$$

TABLE 4 Optimisation metric targets

Metric	Target	Range of possible values
Deviance	0	$[0, \infty]$
Logarithmic loss	0	[0, 1]
AUC	1	[0.5, 1]
Brier score	0	[0, 1]
Calibration intercept*	0	$[-\infty, \infty]$
Calibration slope**	1	$[0, \infty]$
Classification accuracy	1	[0, 1]
Cohen's Kappa	1	[0, 1]

*optimised by minimising the squared value.
**optimised by minimising the squared natural log of the value.

2.3.3 | Study 3: Hyperparameter search algorithm

We varied the hyperparameter search algorithm used when fitting a random forest. We tuned the hyperparameters selected from Study 1 and optimised the metric selected from Study 2. Grid search, random search (both using the R package `caret`[19]), and model-based optimisation (using the R package `tuneRanger`[7]) were used.

Grid search takes a set of possible values for each hyperparameter and uses every possible combination to fit a random forest. Random search computes a given number of random combinations of hyperparameter values and uses each of these combinations to fit a random forest. For both of these search algorithms, the combination of hyparparameter values leading to the best performance (here, as measured by the deviance) is selected to be the optimal hyperparameter values. These two algorithms were performed using 5-fold cross-validation. Grid search used a grid with `mtry` going from 1 through to p and `min.node.size` going from 1 through to 10. Random search was done by computing a grid with `mtry` going from 1 through to p and `min.node.size` going from 1 through to $EF * N$, then randomly selected a number of rows equal to $p * 10/2$ (such that the number of iterations was half the number of iterations in grid search).

Model-based optimisation first computes a given number (here, 30) of random combinations of hyperparameter values and uses each of these combinations to fit a random forest and assess its performance (here, using the deviance). Then, it fits a regression model with the hyperparameter values as predictors and the performance as the outcome to identify a combination of hyperparameter that is expected to perform better. This combination is tried, and the regression model is fit again including the new point. This is repeated a given number of times (here, 70). The best hyperparameter combination is then assessed to be the mean of the 5% best-performing combinations. This algorithm uses out-of-bag performance rather than cross-validation. We refer to Probst and colleagues[7] for details.

2.4 | Performance measures

The same outcomes were used in all studies. Primary outcomes were defined as those which were used to advise on the best tuning procedures. These measures were discrimination, calibration slope, root mean square of the log of the calibration slope over all the runs of a scenario (RMSD(slope)), and computational time. Discrimination was measured by the AUC and assessed whether positive observations have higher predicted risk than negative observation. Calibration was measured by the calibration slope (as calculated by Van Calster[21]) and assessed the extent to which predicted risks reflect true risks. RMSD(slope) assessed the extent to which the calibration slopes varied within a data-generating mechanism and how much they differed from the ideal value of 1, as used by Van Calster and colleagues[22]. Computational time was the amount of time for which a given tuning procedure ran, measured in seconds.

Candidate optimisation metrics from Study 2 were also included as secondary outcomes in all studies. These were the classification accuracy, the calibration intercept, the Brier score, the logarithmic loss, and Cohen's Kappa.

Model performance metrics were estimated using the predictions of the model on a validation set independently generated under the same data generating mechanisms. For each data simulation scenario and tuning procedure combination, we computed the average and Monte Carlo error of each of these performance measures.

TABLE 5 Average performance of hyperparameter combinations (aggregated over all scenarios). Rows are sorted in ascending order of runtime.

Tuned hyperparameters	AUC Mean (SD)	Calibration slope Median (IQR)	RMSD(slope)	Runtime (seconds) Mean (SD)
None	0.7 (0.02)	0.89 (0.27)	0.24	1.6 (0001.2)
mtry + min.node.size	0.7 (0.02)	1.06 (0.28)	0.21	238.5 (0273.5)
mtry + min.node.size + replace	0.7 (0.02)	1.04 (0.29)	0.22	546.6 (0635.6)
mtry + min.node.size + splitrule	0.71 (0.02)	1.24 (0.47)	0.37	672.8 (0691.8)
mtry + min.node.size + sample.fraction	0.7 (0.02)	1.15 (0.4)	0.31	1534.7 (1651.4)
mtry + min.node.size + replace + splitrule	0.71 (0.02)	1.21 (0.48)	0.36	1561.5 (1628.4)
mtry + min.node.size + sample.fraction + replace	0.7 (0.02)	1.14 (0.41)	0.32	3378.4 (3689.1)
mtry + min.node.size + sample.fraction + splitrule	0.71 (0.02)	1.24 (0.54)	0.48	4349.8 (4202.8)
mtry + min.node.size + sample.fraction + replace + splitrule	0.71 (0.02)	1.21 (0.54)	0.48	9660.4 (9508.0)

We evaluated and compared model predictive performance between hyperparameter combinations, optimisation metrics, and search algorithms, in studies 1, 2, and 3, respectively. We visualized whether certain hyperparameter combinations, optimisation metrics, or hyperparameter search algorithms had a notably larger runtime compared to others, without showing a substantial increase in performance. The best hyperparameter combination, optimisation metric, and hyperparameter search algorithm were assessed considering all primary outcomes.

2.5 | Software

Data was simulated and tuning procedures were performed on a high-performance computer. This was done in R version 4.2.2, using packages `MASS`, `dplyr`, `ranger`, `pROC`, `psych`, and in the case of study 3, `tuneRanger` [INCLUDE CITATIONS FOR ALL OF THESE]. Summary statistics and figures were generated on a personal computer using R version 4.2.3 [23] using packages `tidyverse`, `viridis`, and `xtable` [CITE ALL OF THESE].

2.6 | Reproducibility

All code used to generate and analyse data is available at [LINK TO THE GITHUB REPO].

Data cannot be perfectly reproduced due to the nature of this study: the runtime variable is highly dependent on the system on which the script is run and other tasks being performed, so it can never be perfectly reproduced. Additionally, the high-performance computer had two operating systems responding differently to seeds. Runs were assigned to operating systems at random. However, these issues are only a concern with regard to an exact reproduction of the data, and should only lead to minor differences in results and no differences in interpretation.

3 | RESULTS

3.1 | Study 1: Hyperparameters to tune

The average and Monte Carlo error of primary outcomes for each hyperparameter combination are presented in Table 5. Effects were homogeneous across scenarios. We illustrate results specific to the scenario where $p = 8$, $EF = 0.3$, and $n = 1N$ in Figure 1. Calibration plots for the scenarios where $EF = 0.5$ are presented in Figure 2. Results for other scenarios can be visualised at [THIS WILL BE A LINK TO A SHINY APP].

Large increases in runtime were observed when tuning 4 or more hyperparameters, with noticeably worse calibration performance and no observable change in other performance measures (Figure 1, Table 5). Varying hyperparameter tuning combinations had very limited effects on discrimination, with a mean AUC of 0.7 (SD = 0.02) across all hyperparameter combinations. A large effect was observed on model calibration and computational time. Model calibration varied considerably within (Figure 1) and between (Table 5, Figure 1, Figure 2) combinations. The best calibration performance was obtained by tuning `mtry` and `min.node.size`, which had a median calibration slope of 1.06 across all scenarios. This performance was among the

TABLE 6 Average performance of hyperparameter combinations (aggregated over all scenarios). Rows are sorted in ascending order of RMSD(slope).

Optimisation metric	AUC	Calibration slope	RMSD(slope)	Runtime (seconds)
	Mean (SD)	Median (IQR)		Mean (SD)
Logarithmic loss	0.7 (0.02)	1.05 (0.28)	0.20	261.7 (301.4)
Calibration slope	0.7 (0.02)	0.98 (0.27)	0.21	263.0 (301.9)
Deviance	0.7 (0.02)	1.05 (0.28)	0.21	261.7 (301.4)
Brier score	0.7 (0.02)	1.03 (0.29)	0.22	261.4 (300.8)
Calibration intercept	0.7 (0.02)	1.13 (0.31)	0.23	262.9 (301.3)
AUC	0.7 (0.02)	1.04 (0.33)	0.27	262.2 (301.6)
Classification accuracy	0.7 (0.02)	0.9 (0.33)	0.30	262.0 (301.5)
Cohen's Kappa	0.7 (0.02)	0.78 (0.36)	0.42	262.1 (301.5)

most stable with an IQR of 0.28. Tuning `mtry`, `min.node.size`, and `replace` performed similarly well, but took twice as long to run. `mtry` and `min.node.size` was thus selected as the best combination to tune. This is also visible in the calibration plots: calibration seems best when doing minimal tuning [OR SOMETHING TO THIS EXTENT ONCE I HAVE SAID PLOTS]

Secondary outcomes showed similar patterns (Figure 1). Classification accuracy, Brier score, and logarithmic loss showed very little variation both within and between hyperparameter combinations. Calibration intercept and Cohen's Kappa showed variations within, but not between, hyperparameter combinations.

3.2 | Study 2: Optimisation metric

We tuned `mtry` and `min.node.size`. The average and Monte Carlo error of primary outcomes for each optimisation metric are presented in Table 6. Effects were homogeneous across scenarios. We illustrate results specific to the scenario where $p = 8$, $EF = 0.3$, and $n = 1N$ in Figures 3 and 5. Calibration plots for the scenarios where $EF = 0.5$ are presented in Figure 4. Results for other scenarios can be visualised at [THIS WILL BE A LINK TO A SHINY APP].

Varying optimisation metrics had very limited effects on performance and computational time. Discrimination was equal across all optimisation metrics. All optimisation metrics showed similar model calibration performance, with the exception of optimising Cohen's Kappa and classification accuracy, which showed a worse median calibration performance. On the other hand, there are no visible differences in the calibration plots between optimisation metrics (Figure 4). Runtime was comparable across all optimisation metrics. We select logarithmic loss to be optimised in Study 3, as it is among the best performing metrics and is an in-built metric in random forest optimisation packages (e.g. `tuneRanger`[7]), unlike deviance.

Secondary outcomes showed no substantial difference in performance (Figure 3) between optimisation metrics. We further investigated whether optimising a given metric in the training dataset led to better performance on that metric in the validation dataset. This is presented in Figure 5. For all metrics, it seems performance was not improved by optimising for that metric.

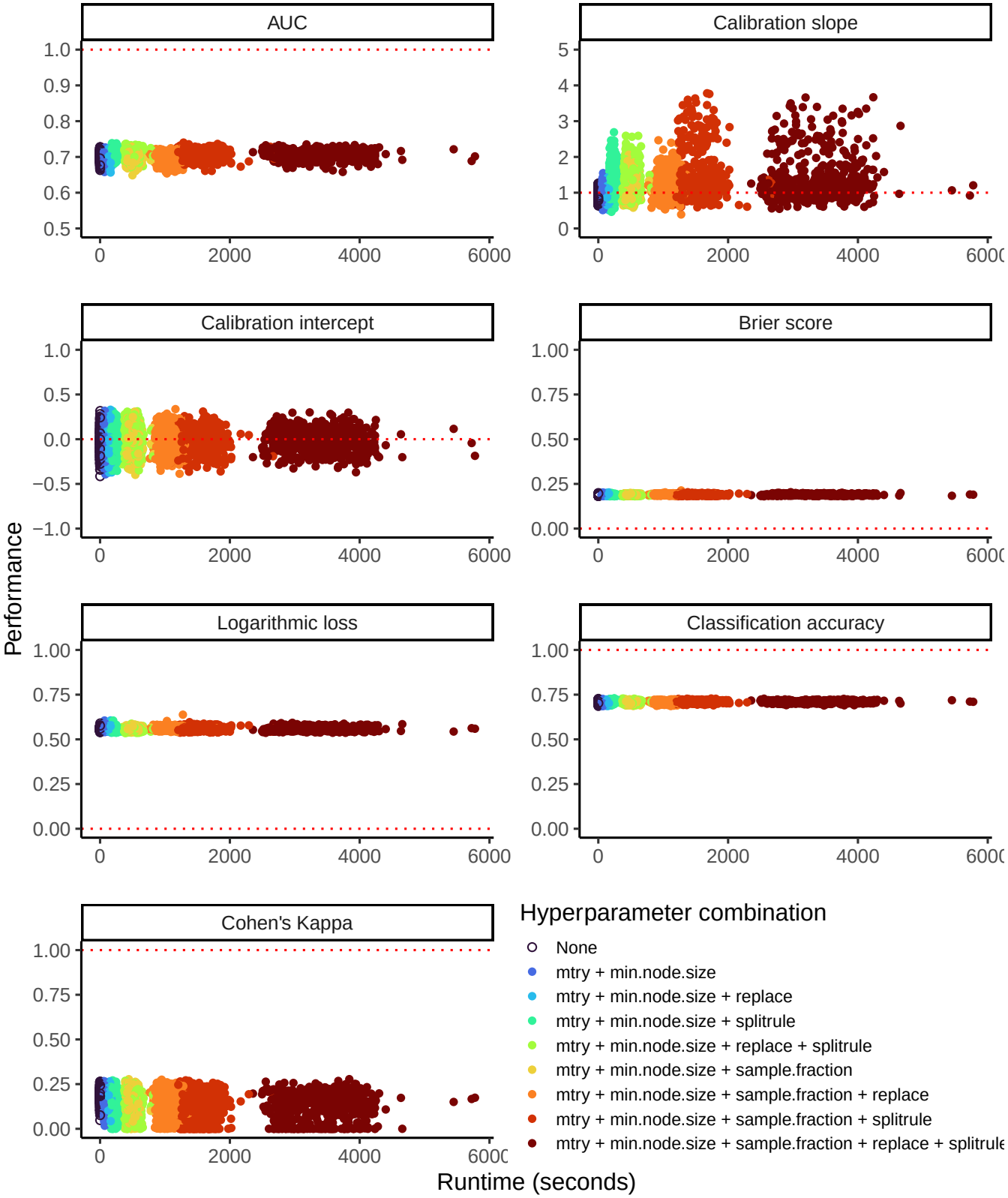

FIGURE 1 Performance of each hyperparameter tuning combination in the example scenario where $p = 8$, $EF = 0.3$, $n = 1N$

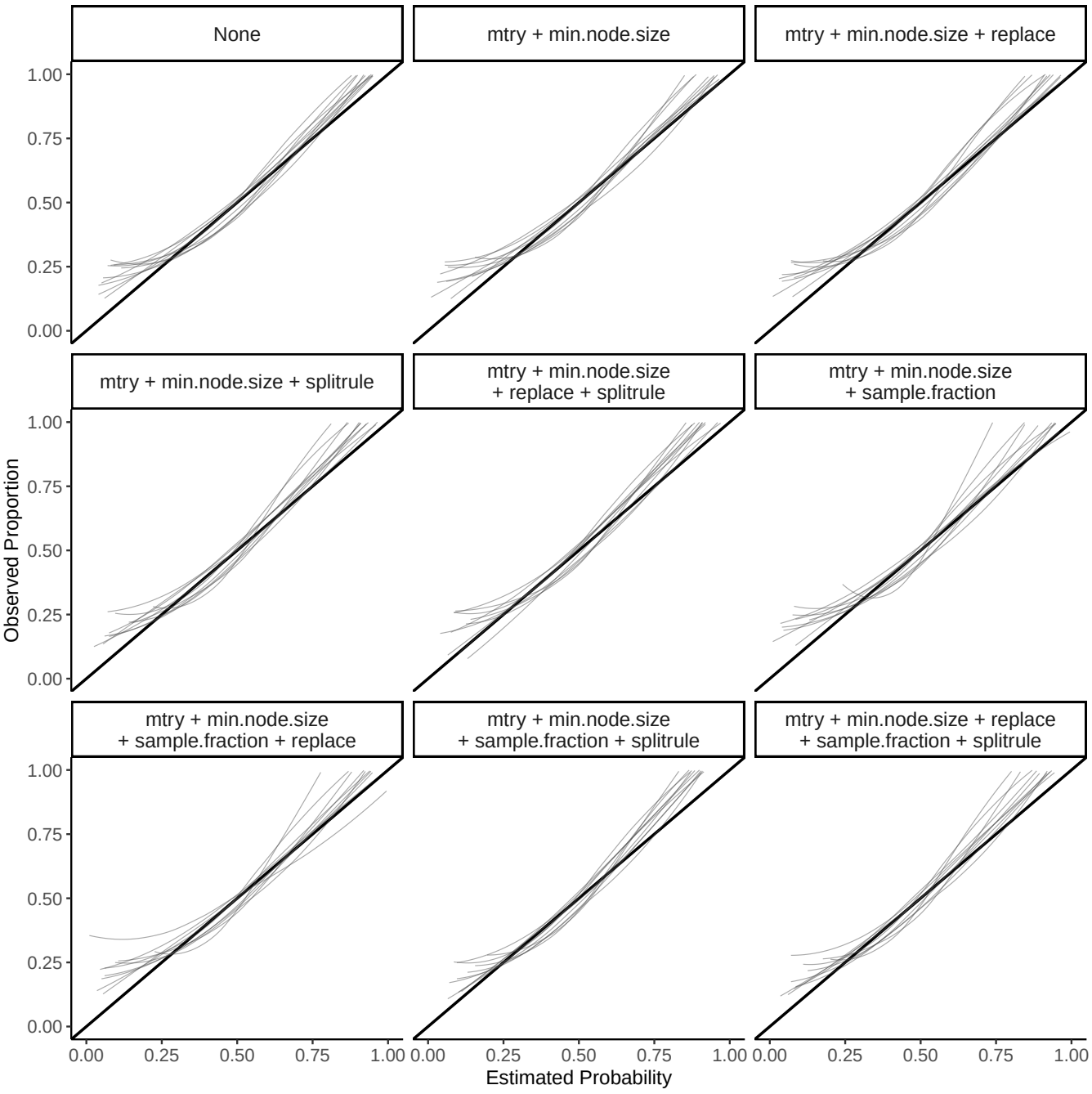

FIGURE 2 Calibration plots for a sample of the datasets where $EF = 0.5$

References

1. Wessler BS, Paulus J, Lundquist CM, et al. Tufts PACE clinical predictive model registry: update 1990 through 2015. *Diagnostic and prognostic research* 2017; 1(1): 1–8.

2. Mitchell TM, Mitchell TM. *Machine learning*. 1 . 1997.

3. Uddin S, Khan A, Hossain ME, Moni MA. Comparing different supervised machine learning algorithms for disease prediction. *BMC Medical Informatics and Decision Making* 2019; 19(1): 281.

4. Andaur Navarro CL, Damen JAA, Smeden vM, et al. Systematic Review Identifies the Design and Methodological Conduct of Studies on Machine Learning-Based Prediction Models. 2022; 154: 8–22. doi: 10.1016/j.jclinepi.2022.11.015

5. Mantovani RG, Horváth T, Cerri R, Junior SB, Vanschoren J, Carvalho dACPdLF. An empirical study on hyperparameter tuning of decision trees. 2019. doi: 10.48550/arXiv.1812.02207

6. Bernard S, Heutte L, Adam S. *Influence of Hyperparameters on Random Forest Accuracy*. Lecture Notes in Computer Science . 2009

7. Probst P, Wright MN, Boulesteix AL. Hyperparameters and tuning strategies for random forest. *WIREs Data Mining and Knowledge Discovery* 2019; 9(3): e1301. doi: 10.1002/widm.1301

8. Van Calster B, McLernon DJ, Smeden vM, et al. Calibration: the Achilles heel of predictive analytics. *BMC Medicine* 2019; 17(1): 230.

9. Benedetto U, Sinha S, Lyon M, et al. Can Machine Learning Improve Mortality Prediction Following Cardiac Surgery?. 2020; 58(6): 1130–1136. doi: 10.1093/ejcts/ezaa229

10. Djulbegovic B, Berano Teh J, Wong L, Hozo I, Armenian SH. Diagnostic Predictive Model for Diagnosis of Heart Failure after Hematopoietic Cell Transplantation (HCT): Comparison of Traditional Statistical with Machine Learning Modeling. 2019; 134: 5799. doi: 10.1182/blood-2019-130764

11. Strubell E, Ganesh A, McCallum A. Energy and Policy Considerations for Deep Learning in NLP. 2019. doi: 10.48550/arXiv.1906.02243

12. Probst P, Boulesteix AL, Bischl B. Tunability: Importance of hyperparameters of machine learning algorithms. *The Journal of Machine Learning Research* 2019; 20(1): 1934–1965.

13. Yang L, Shami A. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing* 2020; 415: 295–316. doi: 10.1016/j.neucom.2020.07.061

14. Ellenbach N, Boulesteix AL, Bischl B, Unger K, Hornung R. Improved Outcome Prediction Across Data Sources Through Robust Parameter Tuning. *Journal of Classification* 2021; 38(2): 212–231. doi: 10.1007/s00357-020-09368-z

15. Morris TP, White IR, Crowther MJ. Using simulation studies to evaluate statistical methods. *Statistics in Medicine* 2019; 38(11): 2074–2102. doi: 10.1002/sim.8086

16. Bischl B, Casalicchio G, Feurer M, et al. OpenML Benchmarking Suites. 2021(arXiv:1708.03731). doi: 10.48550/arXiv.1708.03731

17. Riley RD, Ensor J, Snell KIE, et al. Calculating the sample size required for developing a clinical prediction model. *BMJ* 2020; 368: 441. doi: 10.1136/bmj.m441

18. Wright MN, Ziegler A. ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R. *Journal of Statistical Software* 2017; 77(1): 1–17. doi: 10.18637/jss.v077.i01

19. Kuhn M. Building Predictive Models in R Using the caret Package. *Journal of Statistical Software, Articles* 2008; 28(5): 1–26. doi: 10.18637/jss.v028.i05

20. Luijken K, Groenwold RHH, Van Calster B, Steyerberg EW, Smeden vM. Impact of predictor measurement heterogeneity across settings on the performance of prediction models: A measurement error perspective. *Statistics in Medicine* 2019; 38(18): 3444–3459. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sim.8183doi: 10.1002/sim.8183

21. Van Calster B. benvancalster/classimb_calibration. 2022. https://github.com/benvancalster/classimb_calibration/blob/ad521b46b32ec4

22. Van Calster B, Smeden vM, De Cock B, Steyerberg EW. Regression shrinkage methods for clinical prediction models do not guarantee improved performance: Simulation study. *Statistical Methods in Medical Research* 2020; 29(11): 3166–3178. doi: 10.1177/0962280220921415

23. R Core Team . R: A Language and Environment for Statistical Computing. 2021. version 4.1.2.

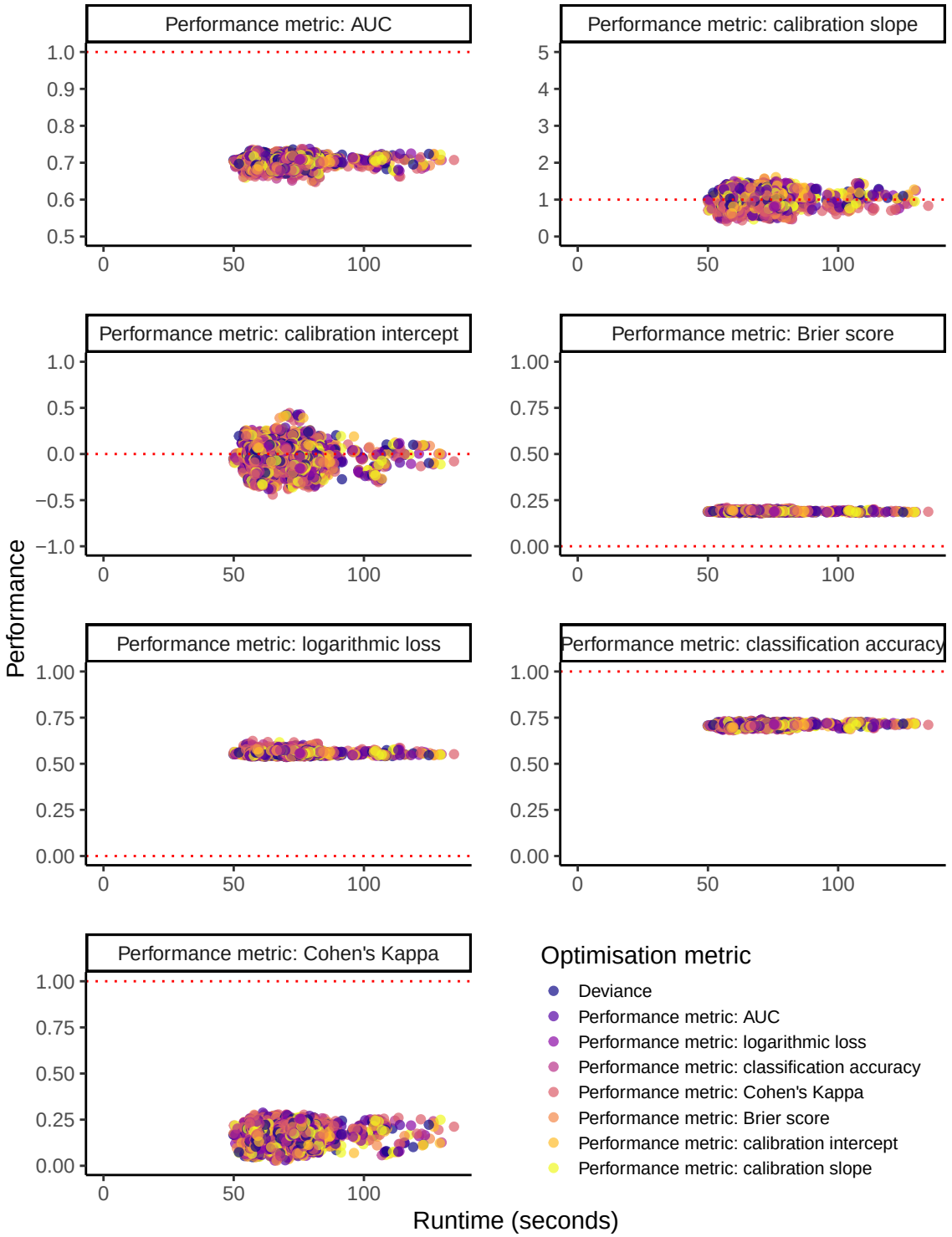

FIGURE 3 Performance of each optimisation metric in the example scenario where $p = 8$, $EF = 0.3$, $n = 1N$

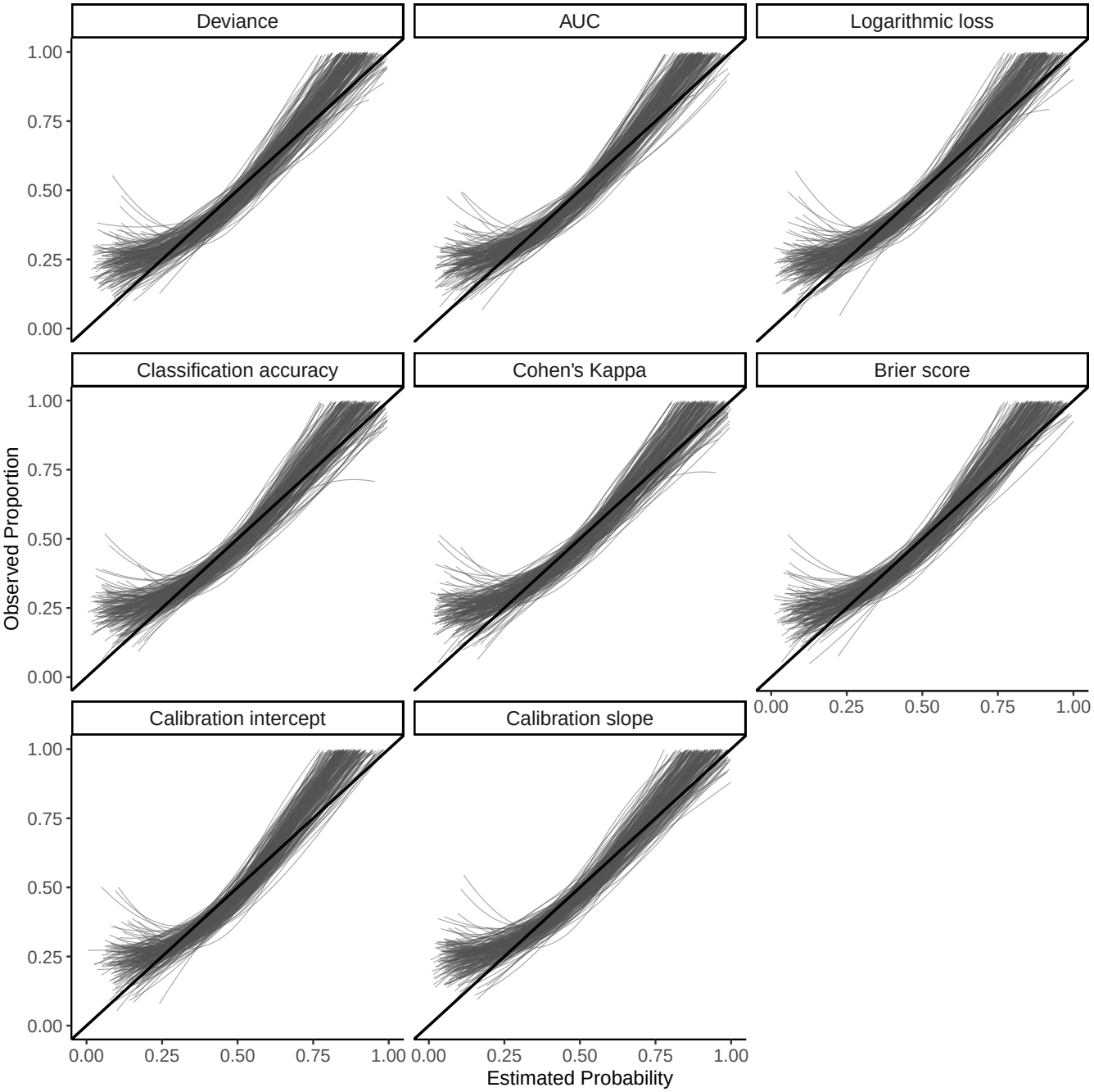

FIGURE 4 Calibration plots for 200 of the datasets where $EF = 0.5$

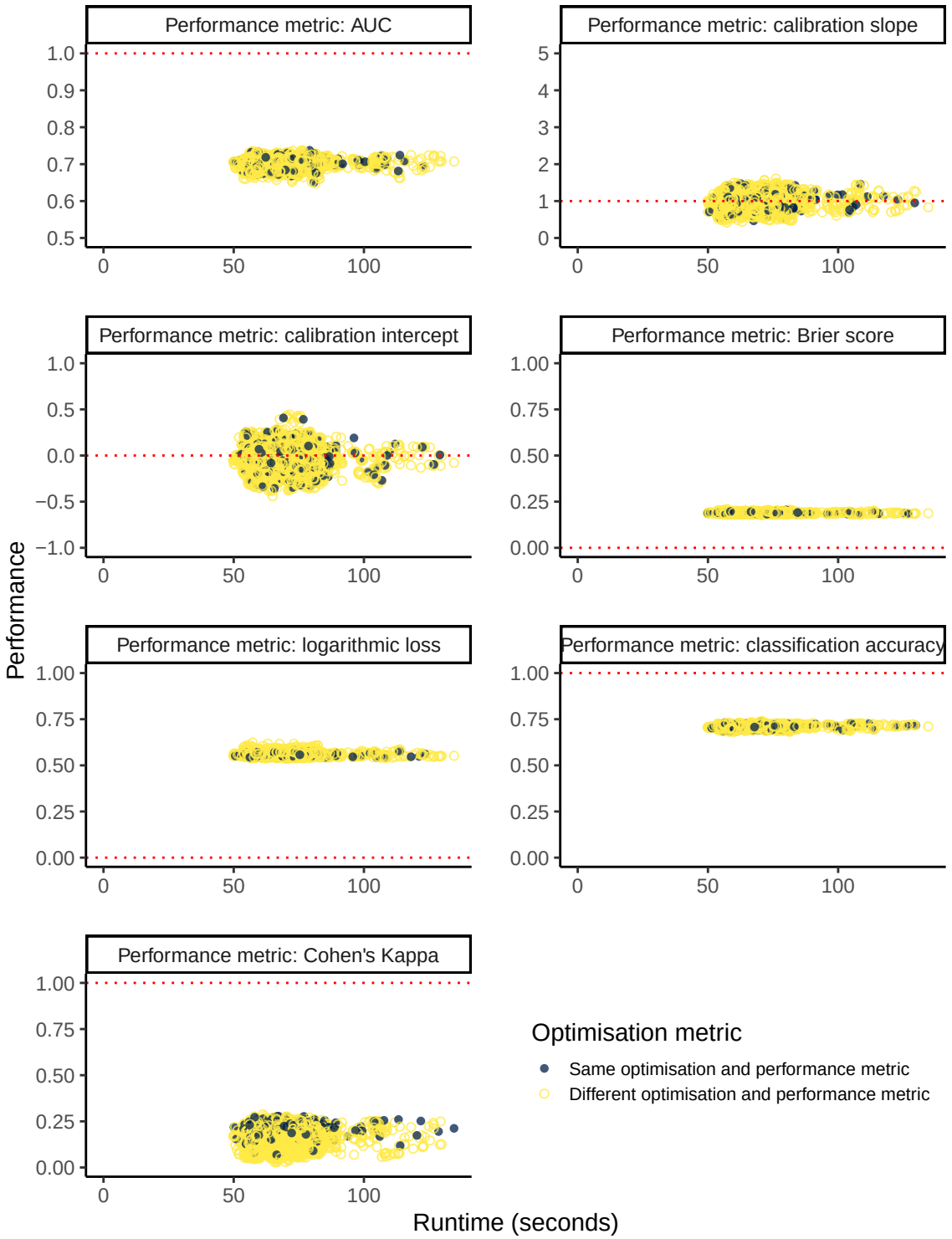

FIGURE 5 Comparison of the performance of optimisation metrics and corresponding performance metrics in the example scenario where $p = 8$, $EF = 0.3$, $n = 1N$