Confidence intervals of survival predictions with neural networks trained on molecular data

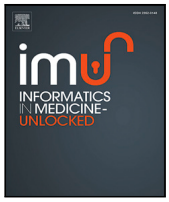
Elvire Roblin, Paul-Henry Cournède, Stefan Michiels

Informatics in Medicine Unlocked 44 (2024) 101426

Contents lists available at ScienceDirect

Informatics in Medicine Unlocked

journal homepage: www.elsevier.com/locate/imu



Confidence intervals of survival predictions with neural networks trained on molecular data

Elvire Roblin [a,b,c,*], Paul-Henry Cournède [a], Stefan Michiels [b,c]

[a] MICS - Laboratory of Mathematics and Computer Science, CentraleSupélec, Paris-Saclay Unversity, France
[b] Oncostat U1018, Inserm, Paris-Saclay University, labeled Ligue Contre le Cancer, Villejuif, France
[c] Department of Biostatistics and Epidemiology, Gustave Roussy, Paris-Saclay University, France

ARTICLE INFO

Keywords:
Machine learning
Multi-layer perceptrons
Confidence intervals
Uncertainty
Survival prediction

ABSTRACT

In medicine, an important objective is predicting patients' survival based on their molecular and clinical characteristics. In this context, neural networks have recently been used for their ability to capture complex interactions in the data. Measuring the uncertainty associated with survival estimates obtained by neural networks is essential to enhance predictions' reliability. We compared four methods adapted to multilayer perceptrons (MLPs) for building confidence intervals at the patient level. The methods were based either on bootstrap with Boot (Efron, 1979), ensembling with DeepEns (Lakshminarayanan et al., 2016), or Monte-Carlo Dropout with MCDrop and BMask (Gal and Ghahramani, 2016; Mancini et al., 2020). A comparison was made through MLP-based survival models: CoxCC and CoxTime (Kvamme et al., 2019) in a continuous time framework, DeepHit (Lee et al., 2018) and PLANN (Biganzoli et al., 1998) in a discrete time framework. We applied the methods to a simulation study, enabling us to estimate a coverage rate of the estimated confidence intervals. We also applied them to real-world datasets, and predicted the survival probability for patients with breast cancer and patients with lung cancer.

In the simulation study, CoxCC and CoxTime obtained the mean C-indices numerically closest to those from the Oracle model (mean C-index of 0.723 for CoxCC, 0.726 for CoxTime, versus 0.743 for the Oracle model). Regarding the confidence intervals of survival probabilities, Boot with CoxCC obtained a coverage rate of 96.5%, the closest to the nominal value of 95%. MCDrop was slightly anticonservative and obtained a coverage rate of 89.8% with CoxTime. This method may represent a reasonable compromise in terms of coverage with regards of computational time. In the breast cancer cohort, MLPs had difficulty capturing additional prognostic information from the molecular data. In contrast, in the lung cancer cohort, the models led to substantially stronger discrimination values when adding molecular data to the clinical variables. In conclusion, we were able to represent uncertainty in the survival estimates at particular time points at the patient level using MLPs in the form of 95% confidence intervals. We recommend using CoxTime with either Boot or, for a less intensive computation time, MCDrop.

1. Introduction

In recent years, machine learning models have been increasingly used in various domains, particularly in medical research. Indeed, in a framework of high dimensional data, these models can discover and identify patterns and relationships between biomarkers for complex datasets and effectively predict future outcomes. Among them are Multi-Layer Perceptrons (MLPs), a type of model developed by Rosenblatt [1] that can learn non-linear and complex relationships in patients' data. While neural networks are well-suited for complex datasets, they cannot be directly applied to survival or time-to-event analysis.

Survival analysis consists of analyzing the time until an event occurs. When an event of interest is unknown because of the patient's withdrawal from the study or the end of the study follow-up, the data is said to be censored. Common events studied are death, disease, clinical relapse or progression, and recovery. MLPs were not applied to survival analysis until the work of Faraggi and Simon [2], as they were not originally designed to handle time-to-event data. Since then, multiple applications of MLPs have been developed [3–6].

MLP models should only be deployed in clinical settings with an associated measure of uncertainty [7]. Indeed, this measure is what

* Corresponding author at: Oncostat U1018, Inserm, Paris-Saclay University, labeled Ligue Contre le Cancer, Villejuif, France.
E-mail address: elvire.roblin@centralesupelec.fr (E. Roblin).

https://doi.org/10.1016/j.imu.2023.101426
Received 3 August 2023; Received in revised form 17 November 2023; Accepted 4 December 2023
Available online 12 December 2023

makes the model interpretable and trustworthy. For instance, the inclusion of this measure can help identify complex diagnostic cases for further evaluation [8]. Such measure has been developed in other contexts: for classification problems under dataset shift [9], for image classification using Bayesian deep neural networks [10,11], or for regression or classification tasks using other machine learning frameworks such as random forests [12]. Yet few machine learning models have addressed the assessment of predictive uncertainty for time-to-event output.

In predictive models, multiple sources of uncertainty exist [13]. One source comes from the noise in the data, which stems from technical issues involved with data collection and measurement. Another source comes from adversarial examples or dataset shifts that force deep learning models to extrapolate predictions vastly different from the observed data. Together, these two sources are called aleatoric uncertainty and caused by irreducible structural relationships within the data. A second type of uncertainty is epistemic uncertainty, which depends on the model's parameters, hyperparameters, or even the chosen model's underlying structure. It is linked to the estimator.

MLPs are estimators that only return point predictions and do not provide a direct measure of uncertainty. Indeed, the weights of an MLP characterizing the predictions are usually fixed, implying that the output is deterministic. In a frequentist framework, one way to overcome this issue is to construct a predictive model by combining multiple learners. In this context, we applied different existing methods to generate multiple functions from an MLP model, enabling us to associate point predictions with uncertainty estimation in the form of 95% confidence intervals. The first method we applied consists of manipulating the training samples using bootstrap [14]. Another strategy includes training multiple neural networks with ensembling. With Deep Ensembles [15], multiple neural networks are trained by randomly initializing the neural network's weights. A third strategy is based on Monte-Carlo Dropout. MCDrop [16] is a combination of models obtained by randomly applying dropout at test time, while BMask [17] is an extension of MCDrop that uses a fixed mask of units being dropped out.

In the present work, these 4 methods are implemented. Each method is applied to different strategies of time-to-event MLPs, and compared for uncertainty quantification. For each method considered, we derive interval forecasts from approximate predictive distributions and thus assess the uncertainty about these MLP models' expected survival predictions. A simulation study is set up to evaluate the performance of the proposed measures of uncertainty on synthetic data with known survival probabilities. We compare the deep learning models with an oracle model in terms of prediction accuracy. We also develop expected survival predictions with uncertainty quantification and evaluate the added value of gene expression data to clinical covariates, using two real patient cohorts in oncology: the METABRIC cohort [18] in early breast cancer and the Lung Cancer Explorer [19] data sets.

2. Related work

In survival analysis, different methods have been proposed to associate an uncertainty measure with survival predictions. The classic approach is to build confidence intervals using the Cox Proportional Hazards (CoxPH) model [20]. Another consists of defining a model in a Bayesian framework and building credible intervals.

2.1. Notations

Let $X = \{x_1, \dots, x_p\}$ be a vector of covariates. Let the random variable T denote the survival time and C, the censoring time. T_i is the survival time for individual i, $1 \leq i \leq N$, with N the total number of individuals. Let us assume that survival times and censoring times are independent, and let $S(t) = P(T > t)$ and $G(t) = P(C > t)$ be the survival function at time t for T and C, respectively. $\hat{S}(t)$ is the Kaplan–Meier estimate of $S(t)$, and $\hat{G}(t)$ is the Kaplan–Meier estimate for the censoring distribution. Let D be the censoring indicator: if T_i is observed, $D_i = \mathbf{I}(T_i \leq C_i) = 1$, with $\mathbf{I}(.)$ the indicator function, while if the corresponding data is censored, $D_i = 0$. The observed time, denoted by $\tilde{T}$, is such that $\tilde{T}_i = T_i$ if $D_i = 1$ and $\tilde{T}_i = C_i$, if $D_i = 0$. Thus, we have $\tilde{T}_i = min(T_i, C_i)$.

2.2. CoxPH model and confidence intervals

The most commonly used model in survival analysis and medical research is the CoxPH model. It is a regression method that simultaneously evaluates the effect of all variables on survival and can be written as:

$$h(t|X_i) = h_0(t)\exp(\phi(X_i)) \text{ with } \phi(X_i) = \beta^T X_i, \tag{1}$$

where h denotes the hazard function at time t for the individual i, and h_0 represents the baseline hazard. The CoxPH is based on a linear predictor $\phi(X_i)$, with β the vector of regression coefficients. This semi-parametric model relies strongly on two assumptions: that the hazard ratio for any two patients is constant over time, and that a linear relationship exists between the log hazard and the covariates.

With high-dimensional data, a penalization term can be added to the CoxPH partial likelihood to determine the most relevant variables among all the variables. This penalization is based on the regression coefficients and generally depends on a single parameter, either positive or null, denoted λ. One of the main objectives of penalized regressions is to force the regression coefficients to tend toward the null value. The most used one is the Least Absolute Shrinkage and Selection Operator (LASSO) penalty because it allows convex optimization and is interpretable in terms of variable selection. First introduced in the context of linear regression by Tibshirani [21] and then adapted to survival analysis [22], the LASSO penalty corresponds to the L_1 norm of the regression coefficients.

From the estimated CoxPH model, the expected survival probability of each patient $i = (1, \dots, n)$ at time t can be obtained by inserting the estimated parameters into the survival function:

$$\hat{S}_i(t|X_i) = exp(-\hat{H}_0(t)\exp(\beta^T X_i)), \tag{2}$$

where $\hat{H}_0(t)$ is the cumulative baseline hazard at time t.

Expected survival probabilities can be associated with confidence intervals at level $1-\theta$ by applying a non-parametric bootstrap (Boot) approach [14]. With this approach, M bootstrap sets are sampled from the original data, and M models are trained. It allows for the estimation of M survival probabilities at time t for a given patient i of the test set, noted as $\hat{S}_{i(boot)}(t) = \{\hat{S}_{i(1)}(t), \dots, \hat{S}_{i(M)}(t)\}$. The percentile method is then used to obtain confidence intervals at level $1-\theta$ for $\hat{S}_i(t)$ based on the distribution of $\hat{S}_{i(all)}(t)$:

$$IC_{1-\theta}\left(\hat{S}_i(t)\right) = \left[q_{\frac{\theta}{2}}\left(\hat{S}_{i(all)}(t)\right); q_{1-\frac{\theta}{2}}\left(\hat{S}_{i(all)}(t)\right)\right], \tag{3}$$

where $q_{\frac{\theta}{2}}$ and $q_{1-\frac{\theta}{2}}$ are percentiles computed using the empirical distribution of the M survival probabilities. Ternès et al. [23] implemented Eq. (3) by using a penalized high-dimensional CoxPH model. They were able to estimate the expected survival probability of future patients at a given time point and the associated confidence intervals.

2.3. Bayesian framework and credible intervals

Bayesian inference is a learning procedure combining explicit prior knowledge with observed data to obtain a predictive model. The prior knowledge is defined by a probability distribution over possible models. Then, based on Baye's rule, the prior distribution is combined with the observations, resulting in the posterior distribution. A set of models can be sampled from the posterior distribution and used to make predictions.

The posterior distribution can be summarized using measures of uncertainty. For instance, a credible interval contains a specific percentage of the posterior distribution's probable values. In survival analysis, 95% credible intervals are obtained from the quantiles of the posterior survival distribution using Bayesian models adapted to time-to-event data. One example is the Bayesian additive regression trees model (BART), a flexible Bayesian non-linear regression approach set in a hierarchical modeling framework [24]. Another example is BDNNSurv [25], a Bayesian hierarchical deep neural network combined with pseudo observations to handle censoring.

In a Bayesian framework, the posterior distribution is a natural quantification of uncertainty and allows for the construction of credible intervals. However, the choice of a prior distribution is not straightforward. In a context of frequentist framework, an MLP only returns point predictions and does not provide a direct measure of uncertainty. To overcome this issue, multiple learners must be trained independently in a non-bayesian way to obtain uncertainty measures.

3. Methods

In the present work, we apply 4 MLP models designed to handle censored time-to-event data. After describing them, we present the methods that enabled us to obtain multiple MLP learners for each MLP models and thus build confidence intervals. Then, we define the evaluation metrics used to compare these methods. Finally, we present the training procedure for searching for the MLP hyperparameters.

3.1. Survival predictions with MLPs

In this work, the linear predictor of the CoxPH model is replaced by an MLP adapted to survival data. More precisely, we compare 4 existing MLP models. Each is defined either in a continuous or discrete time framework and uses a specific loss function to handle time-to-event data.

3.1.1. Continuous time models

We apply 2 MLP models set in a continuous time framework and introduced by Kvamme et al. [4].

CoxCC is a neural network that uses a loss based on a case-control approximation: control samples are not fixed, and a new set of controls is randomly sampled at each iteration. The loss is written as:

$$\mathcal{L}_{\text{CoxCC}} = \frac{1}{N}\sum_{i:D_i=1}\log\Big(\sum_{j\in\tilde{R}^i}\exp[\phi(X_j)-\phi(X_i)]\Big), \tag{4}$$

with $\tilde{R}^i$ a subset of the risk set R^i at time t including individual i. With this loss (Eq. (4)), the neural network can be fitted using a mini-batch gradient descent algorithm.

A second version of this model, named CoxTime, is introduced. It is not constrained by the proportionality assumption, as the time variable is added as input variable. The loss function can be rewritten as:

$$\mathcal{L}_{\text{Cox-Time}} = \frac{1}{N}\sum_{i:D_i=1}\log(\sum_{j\in\tilde{R}^i}\exp[\phi(t_i,X_j)-\phi(t_i,X_i)]). \tag{5}$$

3.1.2. Discrete time models

Commonly used survival models can also be extended using MLPs in a discrete-time framework, which enables overcoming the proportional hazards assumption.

With the Partial Logistic Artificial Neural Network (PLANN) model [3], the time is divided into L time intervals $A_l =]t_{l-1}, t_l]$, $l = 1,\dots,L$, with midpoint a_l. The input of the model is composed of the variables and the time variable a_l, while the output corresponds to the discrete instantaneous hazard, written as:

$$\begin{aligned}\hat{h}_{il} &= \hat{h}_l(X_i, a_l)\\ &= P(T_i \in A_l | T_i > t_{l-1} | X_i),\end{aligned}$$

with T_i the survival time for individual i at time a_l. With the inclusion of the time variable as input of the model, the p variables of each i are repeated for each time interval. More precisely, each patient of the training set is repeated for the number of intervals being observed, whereas, on the test set, each subject is repeated for all time intervals. As the index of the time interval is used as an explanatory variable, smooth estimates of the hazard rate can be obtained, and interactions between time and variables are considered.

The cost function corresponds to the binary cross-entropy :

$$\mathcal{L}_{\text{PLANN}} = -\sum_{i=1}^{n}\sum_{l=1}^{l_i}\{(d_{il}\log\hat{h}_{il}) + (1-d_{il})[1-\log\hat{h}_{il}]\}, \tag{6}$$

where $\hat{h}_{il}(x_i)$ is estimated as output value of the MLP. d_{il} is the event indicator: if the patient died in the interval A_l, then $d_{il} = 1$; if the patient is censored, then $d_{il} = 0$. l_i corresponds to the number of intervals for which the individual i is observed. Thus we have $l_i \leq L$ and $d_{i0},\dots,d_{i(l_i-1)} = 0$.

DeepHit [5] is an adaptation of PLANN, with a loss function that combines the log-likelihood with a ranking loss. We define the time as: $0 = l_0 < \cdots < l_m$ and the output of the neural network as $y(x) = [y_0(x),\dots,y_m(x)]^T$. Given a patient i with covariates X_i, $y_k(X_i), k \in [0,m]$, is the probability that the patient will experience the event at time k. Thus we have:

$$\hat{S}(l_j|X_j) = 1 - \sum_{k=1}^{j} y_k(X). \tag{7}$$

The first term of the loss function is then written:

$$\begin{aligned}\mathcal{L}_{\text{DeepHit 1}} = -\frac{1}{N}\sum_{i=1}^{N}\Big(&\underbrace{(D_i\log(y_{\kappa_i}))}_{\text{Patients with event}}\\ &+\underbrace{(1-D_i)\log(\hat{S}(\tilde{T}|X_i))}_{\text{Censored patients}}\Big).\end{aligned} \tag{8}$$

κ_i is the index of the event time for individual i.

A loss enforcing the discrimination capacity of the model is added to the loss term in Eq. (8). It is an extension of the concordance index:

$$\mathcal{L}_{\text{DeepHit 2}} = \sum_{i,j} D_i\mathbb{I}\{\tilde{T}_i < \tilde{T}_j\}\exp\Big(\frac{\hat{S}(\tilde{T}_i|X_i) - \hat{S}(\tilde{T}_i|X_j)}{\sigma}\Big). \tag{9}$$

$\mathcal{L}_{\text{DeepHit 2}}$ (Eq. (9)) aims at improving the discrimination of the model by forcing the model to focus on times when there are many events. To apply DeepHit, we discretize the survival times using an equidistant time grid. We use the following convex combination of $\mathcal{L}_{\text{DeepHit 1}}$ and $\mathcal{L}_{\text{DeepHit 2}}$ [4]:

$$\mathcal{L}_{\text{DeepHit}} = \alpha\mathcal{L}_{\text{DeepHit 1}} + (1-\alpha)\mathcal{L}_{\text{DeepHit 2}}, \alpha \in [0,1]. \tag{10}$$

3.2. Building confidence intervals

We present different approaches that induce diversity in the MLP models introduced in the previous section. Thus, we can obtain a set of MLPs and output a range of values to build confidence intervals. Indeed, it is not possible to evaluate the accuracy of predictions if we only output point estimates, whereas it is if we associate intervals with these forecasts.

The first method applied to train multiple MLP learners is the bootstrap method. With this method, each model is trained on a different subset of the original training set. However, if the underlying base learner has multiple local optima, as is the case typically with MLPs, the bootstrap can sometimes hurt performances since the base learner is not trained using all data points. The 3 other methods introduced here are based on randomization approaches, either using ensembling or by applying Monte-Carlo Dropout. The resulting predictions differ as

the model is applied to the same data at each iteration but with other parameters.

We introduce the 4 methods we implemented to build confidence intervals with MLPs. Each method enables to output M ($M = 500$ times) survival probabilities for individual i for a given time t. We then have: $\hat{S(t)}_{i(all)} = \{\hat{S}_{i(1)}(t), \dots, \hat{S}_{i(M)}(t)\}$.

3.2.1. Bootstrap method

The first approach we use to build confidence intervals with MLPs is bootstrap, introduced in Section 2.2.

In this approach, M subsets of the training set are generated by random sampling with replacement from the training set. Each observation of the training set has the same probability of being extracted. The bootstrapped training sets are drawn with the same size as the original training set, and thus, several training examples appear multiple times in the set. Due to the number of bootstrapped training sets, the neural network model is estimated several times. As a result, there are exactly M models. Each bootstrapped replicate of the original training set contains, on average, 63.2% of the initial training set.

We also implement the bootstrap-t (Boot-t) method [26], a nested version of the bootstrap algorithm that enables higher coverage for smaller sample sizes. First, we train the model on the entire training set, and then we compute the survival probability at time t $\hat{S}_i(t)$ for a given patient i of the test set. Next, the training set is randomly sampled M times, and we obtain the survival probabilities of patient i using these outer samples: $\hat{S}_{i(outer\ boot)}(t) = \{\hat{S}_{i(1)}(t), \dots, \hat{S}_{i(M)}(t)\}$. For each $m \in \{1, \dots, M\}$, the bootstrap sample is then resampled K times ($K = 50$), and each inner sample set is used to fit the model. For sample m, the survival probabilities for patient i computed with the inner samples are noted as $\hat{S}_{i(m,inner\ boot)}(t) = \{\hat{S}_{i(m,1)}(t), \dots, \hat{S}_{i(m,K)}(t)\}$. Using the K inner sample replicates $\hat{S}_{i(m,inner\ boot)}(t)$, we obtain the standard error of each $\hat{S}_{i(m)}(t)$. Thus we can compute the t-statistic as $t^{stud}_{(m)} = \frac{\hat{S}_{i(m)}(t) - \hat{S}_i(t)}{SE(\hat{S}_{i(m)}(t))}$, with SE the standard error of $\hat{S}_{i(m)}$ on the K inner samples, and build studentized confidence intervals as described in 3.2.4. The major drawback of this algorithm is that it is computationally more intensive than Boot.

3.2.2. Ensembling method

The Boot methods are compared to Deep Ensembles (DeepEns). Lakshminarayanan et al. [15] introduced this method of ensembling. Here, unlike with Boot, the entire training set is used for training each model. The randomness is introduced from within the algorithm as the network parameters are randomly set. Thus, an ensemble of M deterministic MLPs is trained by varying the random seed of the previously tuned set of hyperparameters. Indeed, simply changing the random seed is enough for MLPs to vary in their individual predicted probabilities. Another source of randomness is added by randomly shuffling all the data points of the training set, applying a permutation at each initialization of the model. For each iteration, the model is trained on a random set of parameters and outputs a probability on the test set. The model outputs a different probability per patient of the test set for each initialization. Let M denote the number of MLPs in the ensemble. The method outputs predictions over M models to obtain a predictive distribution.

3.2.3. Monte-Carlo dropout based methods

Typically, dropout is a technique that has been used to prevent overfitting. It randomly excludes units before each neural network layer during training with a chosen probability p. At each iteration, we obtain a different neural network. With the MCDrop method, Gal and Ghahramani [16] suggested activating dropout during test time to output model uncertainty. They placed their work in a Bayesian framework to show that activating dropout with neural networks is equivalent to using variational inference with Gaussian processes. It follows that averaging forward passes through the network with dropout is similar to a Monte Carlo integration over a Gaussian process posterior approximation. Using MCDrop, the output of the model is not a single point estimate but rather a distribution of probabilities sampled from an approximate posterior distribution.

We implement the MCDrop by randomly activating dropout during training and test time. After M iterations on the test set, we obtain an ensemble of M survival predictions at time t for all test set patients.

As an alternative to MCDrop, Mancini et al. [17] introduced the fixed Bernoulli mask (BMask). M fixed Bernoulli masks are defined beforehand, where M sets of vectors are sampled from the Bernoulli distribution before training. Then, each BMask is applied to the neural network architecture for each model during the training and test phases. In these two phases, weights are randomly initialized, and the mask is kept constant.

3.2.4. Percentile confidence intervals

Using the M survival probabilities outputted by a given method, we construct confidence intervals at level $1 - \theta$ by applying the percentile method introduced in Section 2.2. A confidence interval represents a range of values likely to contain a future individual observation from the values of the input predictors that are considered in the model. Here, $\theta = 5\%$ and the 2.5th and 97.5th percentiles are computed using the empirical distribution of the M survival probabilities.

For the Boot-t method, the confidence intervals are studentized. It is supposed to improve the coverage of percentile bootstrap confidence intervals, especially for smaller sample sizes. It is computed as:

$$IC_{1-\theta}\left(\hat{S}_i(t)\right) = \left[\hat{S}_i(t) - \hat{s}q_{1-\frac{\theta}{2}}(t^{stud}), \hat{S}_i(t) - \hat{s}q_{\frac{\theta}{2}}(t^{stud})\right]. \quad (11)$$

$\hat{s}$ is the standard error of $\hat{S}_i(t)$. It is computed with the M Boot estimates: $\hat{S}_{i(outer\ boot)}(t) = \{\hat{S}_{i(1)}(t), \dots, \hat{S}_{i(M)}(t)\}$. The limit of this method is that it can produce estimates outside of the range of plausible values.

3.3. Evaluation metrics

We compare the MLP models using the concordance index and the Brier score, which measures both the discrimination capacity and the calibration of the model. For the simulation study, we evaluate the quality of our survival predictions using bias. We also compute the coverage rate to compare the quality of confidence intervals built using the ensemble of survival predictions.

3.3.1. Concordance index

The concordance index, or C-index, measures the discrimination ability of a model, that is, its ability to distinguish high-risk and low-risk patients. Specifically, it estimates the probability of agreement, i.e., the probability that two randomly selected patients are ordered similarly in terms of survival prediction and their observed survival data. We use here a concordance measure [27] that accounts for the censored data using the inverse probability of censoring weighting. The concordance index for time t is then:

$$\hat{C}(t) = \frac{\sum_{i=1}^{N}\sum_{j=1}^{N} D_i \hat{G}(\tilde{T}_i)^{-2}\mathbb{I}\{\tilde{T}_i < \tilde{T}_j, \tilde{T}_i < t\}\mathbb{I}\{\hat{S}(t|X_i) < \hat{S}(t|X_j)\}}{\sum_{i=1}^{N}\sum_{j=1}^{N} D_i \hat{G}(\tilde{T}_i)^{-2}\mathbb{I}\{\tilde{T}_i < \tilde{T}_j, \tilde{T}_i < t\}}. \quad (12)$$

The value of the C-index lies between 0.5 and 1, with 0.5 equivalent to a random prediction and 1 corresponding to a perfect ability to rank.

3.4. The brier score

The Brier Score (BS) is used to evaluate the accuracy of the predicted survival function at a given time t. It is based on the root mean square error and focuses on the difference between the observed survival status and the predicted probability of survival. It lies between

0 (best possible value) and 0.25. The BS for uncensored data is written as :

$$\begin{aligned}\hat{BS}(t) &= \frac{1}{n}\sum_{i=1}^{n}\left[\mathbb{I}\{T_i > t\} - \hat{S}(t|X_i)\right]^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}\Big[\hat{S}(t|X_i)^2\mathbb{I}\{T_i \le t\} \\ &\quad + (1-\hat{S}(t|X_i))^2\mathbb{I}\{T_i > t\}\Big].\end{aligned} \tag{13}$$

With censored data, only a subset of the event times is observed. A weighting of the BS based on the inverse probability of censoring (IPCW) can be introduced [28]. It can be rewritten as:

$$\begin{aligned}\hat{BS}(t)_{IPCW} =& \frac{1}{n}\sum_{i=1}^{n}\Bigg[\hat{S}(t|X_i)^2\frac{\mathbb{I}\{\tilde{T}_i \le t, D_i = 1\}}{\hat{G}(\tilde{T}_i)} \\ &+ (1-\hat{S}(t|X_i))^2\frac{\mathbb{I}\{\tilde{T}_i > t\}}{\hat{G}(\tilde{T}_i)}\Bigg].\end{aligned} \tag{14}$$

3.4.1. Coverage rate

To evaluate the quality of the confidence intervals, the empirical coverage rate is calculated:

$$CR = \frac{1}{N}\sum_{i=1}^{N} I\left(q_{\frac{\theta}{2}}\left(\hat{S}_{i(all)}(t)\right) \le S_i(t) \le q_{1-\frac{\theta}{2}}\left(\hat{S}_{i(all)}(t)\right)\right). \tag{15}$$

The coverage rate is compared to the nominal value of 95%. For the Boot-t method, we compute the coverage rate using the studentized confidence intervals.

3.4.2. Bias

The quality of survival predictions is estimated through the bias, or the mean difference between the predicted ($\hat{S}_k(t)$) and the theoretical ($S_k(t)$) survival probabilities for each individual:

$$\text{Mean Bias} = \frac{1}{N}\sum_{k=1}^{N}\left(\hat{S}_k(t) - S_k(t)\right). \tag{16}$$

The best model should have a bias close to zero.

3.5. Training procedure of the MLP

To evaluate the performances of the different MLPs, we perform a hyperparameter search in the context of a 5-fold cross-validation (CV).

More precisely, the dataset is split into a training and a test set for each model. For the simulation study, 2,000 samples were drawn for the train and the test sets. Regarding applications, the two datasets are divided into 80% for the training set and 20% for the test set. A 5-fold CV is performed on the train set to search for hyperparameters. Moreover, we use the Tree-Parzen algorithm [29] with the package Optuna in Python, which allows us to select hyperparameters iteratively in an informed manner. It is based on a sequential optimization that is more efficient than a simple random search or an exhaustive grid search. We define a search space for each hyperparameter with specific distribution and boundary values. A set of hyperparameters is randomly sampled, and the model is scored on each of the five validation folds. These five validation scores are averaged. A new set of hyperparameters is sampled based on the value of the average score. We repeat the sampling of hyperparameter sets 200 times.

Several hyperparameters are tuned: the number of hidden layers (1 to 4), the number of neurons per layer (between 4 and 128 nodes per layer), the dropout rate (from 0.1 to 0.5), the L_2 regularization penalty (from 0 to 0.1). We try three activation functions (*tanh*, *relu*, or *elu*). We also compare two optimization algorithms (the Adaptive Moment Estimation optimizer and the RMSProp optimizer) with different learning rates (from 0.001 to 0.01) and batch sizes (between 8 and 128). For DeepHit, we also investigate the number of discretization points for the survival times (testing between 10 and 200 points from the smallest to the largest duration in the training set) and the two parameters of the loss function (α between 0 and 1, σ between 0 and 100). A balance has to be found between a coarse discretization grid that may lose information and a dense grid that increases the number of parameters in the neural network. The application of an interpolation scheme enables us to use a coarser discretization grid.

On a given simulation set, an example of a selected set of hyperparameters for the CoxTime model is one hidden layer with 35 neurons per hidden layer, a dropout rate of 0.08, an L_2 penalty of 0.005, the *relu* activation function, RMSProp optimizer with a learning rate of 0.003 and a batch size of 64.

Finally, the entire training set (or its bootstrap replicate) is used to train each network with the previously selected set of hyperparameters. The results are outputted on the test set. Each model of the ensemble is tuned separately, applying this method. All the models are implemented in Python with PyTorch backend.

4. Simulation study

The performance of the different methods was further evaluated and compared using a simulation study. The advantage of simulated data is that we know its true characteristics and underlying assumptions. Here, we simulated data with non-linearity and interactions.

4.1. Data generation

Survival times were simulated according to the CoxPH model, with a log–logistic basis risk distribution [30]. It enables modeling non-monotonic risk rates. Based on the inverse cumulative distribution method [31], survival times are related to covariates as follows:

$$T = \frac{1}{\lambda}\{\exp[-\log(u)\exp(\beta f(X))] - 1\}^{(\frac{1}{\gamma})}. \tag{17}$$

First, we simulated the variable U according to a uniform distribution $\mathcal{U}(0,1)$. Then, $p+1$ variables are drawn independently: $x_1, \ldots, x_p$ following a normal distribution and z_1 that is generated according to a Bernoulli law $\mathcal{B}(0.5)$. X is a sub-sample of 3 variables: $X = (x_1, x_2, x_3)$. These 3 variables are linked to the survival time with $f(X) = \exp(X^T V X)$ and $V = 0.05\times\begin{pmatrix}1 & \rho & \rho^2\\ \rho & 1 & \rho\\ \rho^2 & \rho & 1\end{pmatrix}$. It introduces non-linearity and interactions. The remaining covariates are noise variables and do not contribute to the survival times. Here, $\beta = 0.5$, $\rho = 0.95$, $\lambda = 1.25$ and $\gamma = 0.9$.

Censoring times are generated with an exponential distribution to obtain around 20% of censoring among the data set. Survival times longer than 15 years are censored.

100 simulation data sets are generated, composed of 4,000 individuals each. Each data set is split into a train set (2,000 individuals) and a test set (2,000 individuals). The parameters are estimated on the training data, and the survival prediction and confidence intervals are computed on the test set.

4.2. Oracle survival probabilities

Theoretical survival probabilities for each i at a given time t are obtained using the true value of the log–logistic basis risk function at time t. It is estimated with:

$$h_0(t) = \lambda\gamma(t)^{\gamma-1}(1+(\lambda t)^{\gamma})^{-1}. \tag{18}$$

Using the baseline hazard (Eq. (18)), the baseline cumulative hazard is obtained with the relation $H_0(t) = \int_0^t h_0(s)ds = \log\left(1+(\lambda t)^{\gamma}\right)$. Then the survival function can be retrieved through $H_0(t)$:

$$S(t|X) = \exp\left[-H_0(t)\exp\left(\beta^T f(X_i)\right)\right] \tag{19}$$

These probabilities are used to obtain an oracle C-index and the coverage rate of confidence intervals.

Table 1
Mean of the M C-indices and BS obtained on the 100 simulation test sets at a fixed time ($t = 0.5$).

	C-index				Brier score			
	Boot	DeepEns	MCDrop	BMask	Boot	DeepEns	MCDrop	BMask
Oracle	0.743 (±0.07)	0.743 (±0.07)	0.743 (±0.07)	0.743 (±0.07)	0.149 (±0.004)	0.149 (±0.004)	0.149 (±0.004)	0.149 (±0.004)
CoxCC	**0.712** (±**0.01**)	0.723 (±0.012)	0.718 (±0.017)	0.723 (±0.014)	**0.162** (±**0.007**)	**0.157** (±**0.006**)	0.157 (±0.005)	**0.155** (±**0.005**)
CoxTime	0.709 (±0.015)	**0.721** (±**0.015**)	**0.722** (±**0.009**)	**0.726** (±**0.009**)	0.164 (±0.008)	0.158 (±0.007)	**0.156** (±**0.005**)	**0.155** (±**0.006**)
DeepHit	0.649 (±0.049)	0.707 (±0.022)	0.720 (±0.008)	0.711 (±0.016)	0.208 (±0.045)	0.191 (±0.061)	0.183 (±0.023)	0.183 (±0.017)
PLANN	0.577 (±0.091)	0.711 (±0.021)	0.701 (±0.033)	0.717 (±0.016)	0.203 (±0.028)	0.162 (±0.01)	0.163 (±0.008)	0.158 (±0.006)

Note: The number in brackets is the standard deviation of the M values. The highest value for the C-index and the lowest value for the BS per combination method is in bold.

4.3. Results

In this section, the findings of the simulation study are presented. All measures were computed using a fixed survival time as the horizon ($t = 0.5$).

We first compared the different MLPs in terms of discrimination and calibration. The C-index was estimated M times for each data set, and the mean of the M values was computed. Then, the mean of the 100 C-indices was calculated using all simulation data sets and is reported in Table 1. Regardless of the combination method, both CoxCC and CoxTime obtained the mean C-indices numerically closest to those from the oracle model. The standard deviation of the bootstrap values was much larger for the discrete-time models, DeepHit and PLANN, as compared to CoxCC and CoxTime. Concerning the BS, the results obtained from the comparison were close to those for the C-index.

Next, we compared Boot, DeepEns, MCDrop and BMask applied on MLPs in terms of coverage rate. Table 2 shows the coverage rate of confidence intervals for survival probabilities for all patients in the test set. More precisely, this is the mean value of the coverage rates on all simulation test sets, and the value in brackets corresponds to the standard deviation. The CoxTime model and the Boot method achieved the closest value to 95%, slightly above this pre-defined nominal value. On average, the Boot method with the CoxTime model yielded 95% confidence intervals that contained the true survival probability 96.4% of the time. Overall, the Boot method for the continuous-time framework models and the Boot-t method for the discrete-time framework models achieved the highest coverage rates. The average length of confidence intervals for Boot was larger than for the other combination methods (Table B.13), indicating that the method could be too conservative. The studentized correction increased the coverage rate of the Boot method. On the anti-conservative side, DeepEns, MCDrop, and BMask had similar results with relatively good coverage. MCDrop with CoxTime achieved the closest value to the nominal value of 95%. Note that Table 2 also shows that in a comparison of MLP models, DeepHit and PLANN obtained lower average coverage rates for all the methods considered and much more variability across the M bootstrap resamples. In fact, researchers [32] have demonstrated that DeepHit can reach good discrimination capacities, but at the cost of poorly calibrated survival estimates.

As a benchmark method, we applied the Boot method to a CoxPH model with a LASSO penalization on linear effects. LASSO obtained a low average coverage rate, a low average C-index, and a high average BS, as shown in Table B.8. These results are consistent when we consider that the LASSO model with linear effects is not well suited to a complex dataset that includes multiple interactions and non-linear effects in the data generation process.

Then, we investigated the behavior of the methods in terms of bias. In Fig. 1, we estimated the bias for the MCDrop method. Note that the results are near zero, with the boxplots close to the red dashed line. MCDrop, BMask, and DeepEns obtained results which were close to each other. CoxCC and CoxTime had a negative bias, while DeepHit had the lowest bias. The bias was more variable with the methods applied to PLANN. Finally, we compared the computation times of the methods and MLP models trained in Python. For each method, the M models were trained on a CPU with 2.1 GHz and 4 GB RAM for each simulation set. Fig. 2 displays the running times of the different methods. Boot was the slowest method in terms of computational time. For instance, the median computing time for Boot with CoxCC was twelve times longer than for BMask with the same model. For a given method (here, MCDrop), we did not observe substantial differences between the MLP models.

Table 2
Mean of the M 95% Coverage Rates obtained on the 100 simulation test sets at a fixed time ($t = 0.5$).

	Coverage rate			
	CoxCC	CoxTime	DeepHit	PLANN
Boot	**0.965** (±**0.033**)	**0.964** (±**0.066**)	0.711 (±0.177)	0.702 (±0.169)
Boot-t	0.976 (±0.027)	0.974 (±0.035)	**0.874** (±**0.168**)	**0.856** (±**0.096**)
DeepEns	0.884 (±0.07)	0.882 (±0.091)	0.706 (±0.081)	0.793 (±0.138)
MCDrop	0.895 (±0.044)	0.898 (±0.041)	0.788 (±0.072)	0.681 (±0.128)
BMask	0.879 (±0.068)	0.895 (±0.07)	0.703 (±0.076)	0.761 (±0.102)

Note: The number in brackets is the standard deviation of the M values. The closest value to 95% per model is in bold.

5. Application on real patients cohorts

The methods were applied to 2 high-dimensional cancer cohorts: the METABRIC cohort for patients with breast cancer and the Lung Cancer Explorer cohort for patients with lung cancer.

5.1. Cancer cohorts

The first example data set we used is the METABRIC (Molecular Taxonomy of Breast Cancer International Consortium) cohort, which can be extracted from the MetaGxBreast R package [18]. It comprises clinical features and large-scale gene expression data of breast cancer patients obtained at surgery. We used a nonspecific filter independent from the outcome based on standard deviation to increase the statistical power [33]. We removed genes with an overall standard deviation below the quantile of 95% and retained the 863 genes with the highest standard deviations. The quantile was chosen to obtain approximately 1,000 genes. We selected the probe with the highest variance if multiple probes corresponded to the same gene. 6 clinical covariates were used: age at diagnosis, tumor grade, tumor size, the number of invaded lymph nodes, hormonal therapy, and chemotherapy use. We removed individuals with missing values for the survival time. Since one or more covariate values were missing for 106 patients, we imputed them using predictive mean matching for numerical features and a multinomial logit model for categorical covariates [34]. Then, we standardized the numerical covariates using Z-score normalization and applied one-hot

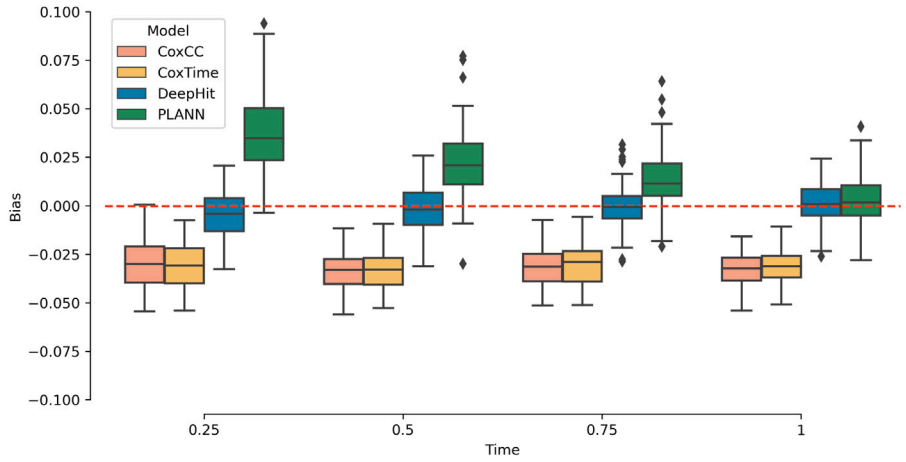


Fig. 1. Boxplot of the M bias values estimated at fixed time points for all synthetic data sets. Each boxplot shows the results for one neural network model applied to all simulation data sets with the MCDrop method. The red horizontal line corresponds to the null value. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

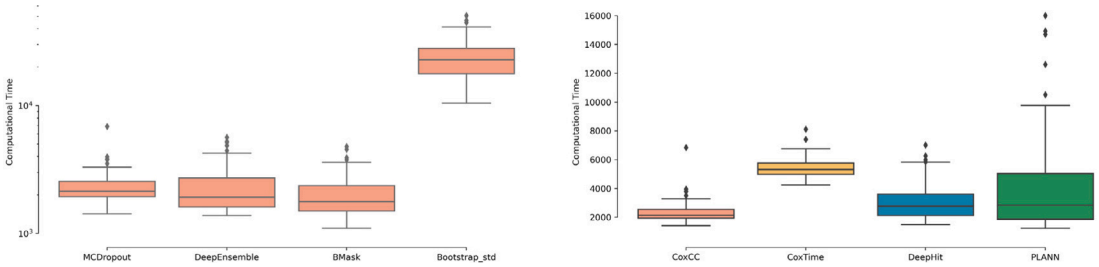


Fig. 2. On the left side, computational time in seconds (logarithmic scale) for CoxCC, using the different methods. On the right side, computational time in seconds of all the MLP models for MCDrop.

encoding to categorical covariates. The final data set represents 1,960 patients and 869 features, with a median survival time of 88 months and a censoring rate of 54.6.

We applied our methods to a second data set: the Lung Cancer Explorer (LCE). It is an online tool created by UT Southwestern Medical Center's Quantitative Biomedical Research Center, which is composed of more than 6,700 patients over 56 data sets, including the Cancer Genomics Atlas (TCGA) and various Gene Expression Omnibus (GEO) data sets. All the data sets provide gene expression and clinical data from lung cancer patients. The expression data stems from 23 genome-wide platforms and is predominantly composed of microarrays. It is reprocessed, normalized, and converted from probe to gene levels. In our work, we selected 2 clinical covariates: cancer stage and the patient's age. Only the genes with less than 5% of missing values were kept for the gene expression data. Then, the 1,000 genes with the highest standard deviations among the patients were conserved. The missing values were replaced with the multiple imputation method using chained equations. We used Bayesian polytomous regression for the stage categories and predictive mean matching for continuous covariates. We standardized the numerical covariates using Z-score normalization and applied one-hot encoding to the categorical covariates. The final dataset includes 4,120 patients and 1,002 features, with a median survival time of 45 months and a censoring rate of 49.0.

5.2. Results

Here, we present the results obtained on the two cancer cohorts. First, the performances of the different methods are compared. Then, examples of confidence intervals computed at the patient level are presented.

On the METABRIC cohort, we compared the results obtained using either all the input covariates or only the clinical covariates. In terms of predictive accuracy (Tables 3 and 4), the methods demonstrated higher mean C-indices and lower BS for neural networks fitted only to the clinical covariates, as compared to those fitted to clinical plus molecular data. This highlights that the clinical covariates already contain a large part of the predictive information for survival.

Tables 5 and 6 report the mean C-indices and mean BS at 2 years using the LCE. In this cohort, adding the molecular data improved discrimination and accuracy, regardless of the chosen neural network architecture or the combination method. Thus, in this lung cancer data set, there was additional prognostic information in the molecular data, that was captured by the neural network beyond the clinical covariates.

We further analyzed expected survival probabilities from the perspective of advising a clinical practitioner. This was done for two patients from the test set of the METABRIC data set: Patient 1 had a survival time of 3 years and 6 months with no censoring; Patient 2 had a survival time of 5 years and 3 months, and his follow-up time was censored. Fig. 3 represents the density of survival probabilities obtained with the MCDrop method applied on CoxTime. With this method, the mean survival probability at 5 years for Patient 1 (orange line) was 0.674 with a confidence interval of $[0.525, 0.849]$. Patient 2 had a 0.910 mean survival probability at 5 years, with a confidence interval of $[0.838, 0.954]$. Even though the event had already happened, the estimated mean survival probability was still relatively high for Patient 1. However, as the graph shows, the corresponding density is more dispersed, and the confidence interval is wider than for Patient 2.

Two patients were also sampled from the LCE test set: Patient 3 was followed for 6 years and 11 months without censoring; Patient 4 had a survival time of 9 years and 4 months before being unavailable for follow-up. Applying the same previous method, we obtained a mean predicted survival probability at 2 years of 0.781 with a confidence interval of $[0.533, 0.956]$ for Patient 3 (pink line on Fig. 3). The estimated

Table 3
Mean of the M values of C-index at 5 years on the METABRIC test set with molecular and clinical covariates.

	All variables				Clinical variables			
	Boot	DeepEns	MCDrop	BMask	Boot	DeepEns	MCDrop	BMask
CoxCC	0.638 (±0.03)	0.663 (±0.023)	0.654 (±0.025)	0.639 (±0.042)	0.655 (±0.027)	0.700 (±0.011)	0.702 (±0.011)	0.606 (±0.045)
CoxTime	0.643 (±0.032)	0.652 (±0.045)	0.657 (±0.026)	0.662 (±0.023)	0.695 (±0.026)	0.736 (±0.01)	0.734 (±0.013)	0.703 (±0.031)
DeepHit	**0.655** (±**0.026**)	**0.670** (±**0.021**)	**0.660** (±**0.021**)	**0.696** (±**0.013**)	**0.717** (±**0.013**)	0.731 (±0.03)	0.735 (±0.007)	**0.718** (±**0.012**)
PLANN	0.654 (±0.025)	0.666 (±0.036)	0.658 (±0.03)	0.633 (±0.058)	0.688 (±0.038)	**0.739** (±**0.009**)	**0.742** (±**0.004**)	0.704 (±0.009)

Note: The number in brackets is the standard deviation of the M values.

Table 4
Mean of the M values of BS at 5 years on the METABRIC test set with molecular and clinical covariates.

	All variables				Clinical variables			
Model	Boot	DeepEns	MCDrop	BMask	Boot	DeepEns	MCDrop	BMask
CoxCC	0.168 (±0.015)	0.156 (±0.005)	0.158 (±0.006)	0.164 (±0.009)	0.149 (±0.005)	0.141 (±0.002)	0.140 (±0.002)	0.161 (±0.009)
CoxTime	0.165 (±0.008)	**0.154** (±**0.008**)	**0.154** (±**0.004**)	0.157 (±0.006)	**0.143** (±**0.004**)	**0.135** (±**0.002**)	0.134 (±0.003)	0.147 (±0.005)
DeepHit	**0.156** (±**0.002**)	**0.154** (±**0.002**)	**0.154** (±**0.003**)	**0.155** (±**0.001**)	0.150 (±0.002)	0.151 (±0.006)	0.149 (±0.001)	0.156 (±0.001)
PLANN	0.176 (±0.011)	0.178 (±0.063)	0.164 (±0.057)	0.180 (±0.065)	0.187 (±0.124)	0.133 (±0.002)	**0.133** (±**0.001**)	**0.145** (±**0.007**)

Note: The number in brackets is the standard deviation of the M values.

Table 5
Mean of the M values of C-index at 2 years on the LCE test set with molecular and clinical covariates.

	All variables				Clinical variables			
	Boot	DeepEns	MCDrop	BMask	Boot	DeepEns	MCDrop	BMask
CoxCC	0.665 (±0.019)	0.672 (±0.022)	0.699 (±0.01)	0.695 (±0.012)	0.633 (±0.013)	**0.646** (±**0.004**)	**0.644** (±**0.005**)	0.578 (±0.034)
CoxTime	0.653 (±0.023)	0.673 (±0.018)	0.683 (±0.019)	0.659 (±0.02)	0.638 (±0.008)	0.645 (±0.005)	0.636 (±0.01)	0.639 (±0.011)
DeepHit	0.673 (±0.017)	0.697 (±0.028)	0.698 (±0.013)	0.647 (±0.007)	**0.654** (±**0.002**)	0.634 (±0.028)	0.634 (±0.013)	0.638 (±0.018)
PLANN	**0.688** (±**0.015**)	**0.703** (±**0.012**)	**0.700** (±**0.014**)	**0.691** (±**0.015**)	0.628 (±0.022)	0.643 (±0.018)	**0.644** (±**0.006**)	0.638 (±0.018)

Note: The number in brackets is the standard deviation of the M values.

Table 6
Mean of the M values of BS at 2 years on the LCE test set with molecular and clinical covariates.

	All variables				Clinical variables			
	Boot	DeepEns	MCDrop	BMask	Boot	DeepEns	MCDrop	BMask
CoxCC	0.176 (±0.005)	0.172 (±0.004)	0.169 (±0.004)	0.169 (±0.005)	0.181 (±0.003)	**0.178** (±**0.001**)	**0.178** (±**0.001**)	0.187 (±0.006)
CoxTime	0.177 (±0.004)	0.171 (±0.004)	**0.171** (±**0.004**)	0.178 (±0.008)	**0.180** (±**0.003**)	0.180 (±0.001)	0.181 (±0.002)	**0.179** (±**0.001**)
DeepHit	0.185 (±0.004)	0.192 (±0.006)	0.182 (±0.004)	0.176 (±0.001)	0.189 (±0.002)	0.186 (±0.001)	0.193 (±0.004)	0.190 (±0.001)
PLANN	**0.172** (±**0.005**)	**0.169** (±**0.004**)	0.172 (±0.005)	**0.170** (±**0.003**)	0.183 (±0.026)	**0.178** (±**0.002**)	**0.178** (±**0.001**)	0.190 (±0.001)

Note: The number in brackets is the standard deviation of the M values.

mean survival probability for Patient 4 was 0.960 with a confidence interval of $[0.885, 0.993]$.

6. Discussion

This paper develops confidence intervals for expected survival probabilities using a combination of artificial neural networks. Our simulation study assessed the performance of different uncertainty measures by comparing various models' survival predictions to the oracle value. These methods were evaluated for two cohorts of cancer patients. We also described expected survival estimates for two patients of each data set, representing uncertainty at the patient level.

Regarding survival analysis, the neural network models defined in a continuous time framework performed the best, with close results between the two variants, CoxCC and CoxTime. In the simulation study, the MLP models CoxCC and CoxTime performed similarly. As the simulated data did not include time-dependent variables, there was no reason for CoxTime to perform better. CoxTime obtained the highest C-indices for the two cohort studies, which confirms results presented in other studies [32]. The discrete-time methods, DeepHit and PLANN, had lower performances when compared to CoxCC and CoxTime. DeepHit and PLANN can reach good discrimination capacities at the cost of poorly calibrated survival estimates.

A comparison of combination methods shows that the Boot method obtained encouraging results for the simulation study but was computationally the most costly; unfortunately, the computational cost could be a drawback when training deep neural networks. Moreover, coverage rates above 95% and the larger average length of confidence intervals for this method indicate that it could be slightly too conservative. DeepEns, MCDrop, and BMask performed well in terms of empirical coverage. MCDrop was the second best method after the Bootstrap one, although it was slightly anti-conservative in the simulations, reaching a coverage rate slightly under 95%. Finally, MCDrop performed well in the simulation study and the applications and was less computer-intensive than Boot. Therefore, MCDrop may represent a reasonable compromise in terms of coverage with regards of computational time,

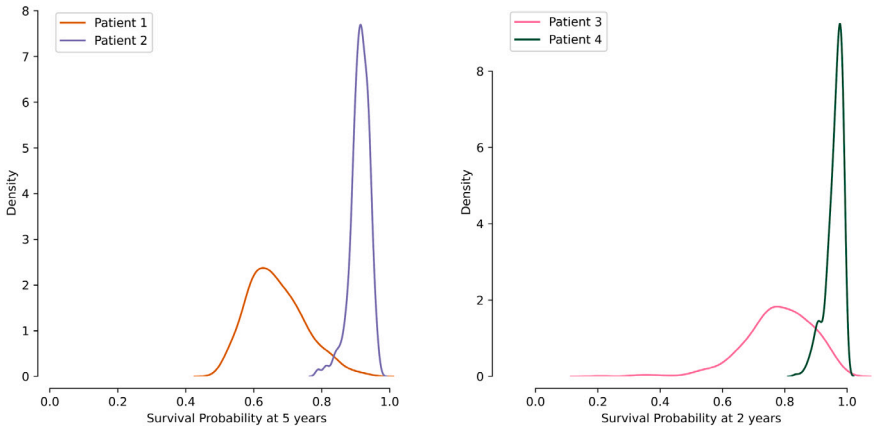


Fig. 3. Density estimate of expected survival probabilities using the MCDrop method with CoxTime, at 5 years for two patients with breast cancer on the left side and at 2 years for two patients with lung cancer on the right side.

especially for deep neural networks, and we recommend to combine it with the CoxTime model for survival analysis.

In the METABRIC breast cancer cohort, the neural networks had difficulty capturing additional prognostic information from the molecular data, suggesting the clinical variables alone are sufficient. In contrast, in the LCE cohort, the models led to substantially stronger discrimination values when adding molecular data to the clinical variables.

One of the limits concerning our combination strategies is that we did not fully investigate the uncertainty resulting from model misspecification. We could account for this type of uncertainty by creating an ensemble of predictions based on different types of models. This amounts to combining weighted predictions from multiple models. Also, we could stack the predictions with the help of a meta-model.

Other sources of uncertainty have the potential to be analyzed in future works. Applying data set shift to a model using adversarial training [15], for example, can reveal variations in the input or the output distributions, especially between the training and test sets.

The different combination methods investigated here could also be applied to other types of neural network architectures, such as Recurrent Neural Networks (RNN). For instance, Dusenberry et al. [35] built an ensemble of RNNs using Deep Ensemble.

Software

The code to obtain the results in this article is available at https://github.com/Oncostat/Unmeasures_nnet.

CRediT authorship contribution statement

Elvire Roblin: Data curation, Formal analysis, Investigation, Visualization, Writing – original draft, Writing – review & editing. **Paul-Henry Cournède:** Conceptualization, Supervision, Validation, Writing – original draft, Writing – review & editing. **Stefan Michiels:** Conceptualization, Supervision, Validation, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Elvire Roblin acknowledges financial support by Foundation Philanthropia-Odier (Ph.D. scholarship).

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.imu.2023.101426.

References

[1] Rosenblatt F. The perceptron: A probabilistic model for information storage and organization in the brain. Psychol Rev 1958;65:386–408.
[2] Faraggi D, Simon R. A neural network model for survival data. Stat Med 1995;14(1):73–82.
[3] Biganzoli E, Boracchi P, Mariani L, Marubini E. Feed forward neural networks for the analysis of censored survival data: a partial logistic regression approach. Stat Med 1998;17(10):1169–86. http://dx.doi.org/10.1002/(SICI)1097-0258(19980530)17:10<1169::AID-SIM796>3.0.CO;2-D.
[4] Kvamme H, Borgan O, Scheel I. Time-to-event prediction with neural networks and cox regression. 2019, arXiv:1907.00825.
[5] Lee C, Zame WR, Yoon J, Schaar MVD. DeepHit: A deep learning approach to survival analysis with competing risks. In: 32nd AAAI conference on artificial intelligence, vol. 32, no. 1. 2018, p. 2314–21.
[6] Roblin E, Cournede P-H, Michiels S. On the use of neural networks with censored time-to-event data. In: Bebis G, Alekseyev M, Cho H, Gevertz J, Rodriguez Martinez M, editors. Mathematical and computational oncology. Cham: Springer International Publishing; 2020, p. 56–67. http://dx.doi.org/10.1007/978-3-030-64511-3_6.
[7] Krzywinski M, Altman N. Points of significance: Importance of being uncertain. Nat Methods 2013;10:809–10. http://dx.doi.org/10.1038/nmeth.2613.
[8] Kompa B, Snoek J, Beam AL. Second opinion needed: communicating uncertainty in medical machine learning. npj Digit Med 2021;4. http://dx.doi.org/10.1038/s41746-020-00367-3.
[9] Ovadia Y, Fertig E, Ren J, Nado Z, Sculley D, Nowozin S, Dillon JV, et al. Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift. 2019.
[10] Leibig C, Allken V, Ayhan MS, Berens P, Wahl S. Leveraging uncertainty information from deep neural networks for disease detection. Sci Rep 2017;7. http://dx.doi.org/10.1038/s41598-017-17876-z.
[11] Kendall A, Gal Y. What uncertainties do we need in Bayesian deep learning for computer vision? In: Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, Garnett R, editors. Advances in neural information processing systems, vol. 30. Curran Associates, Inc.; 2017.
[12] Wager S, Hastie T, Efron B. Confidence intervals for random forests: The jackknife and the infinitesimal jackknife. J Mach Learn Res 2014;15:1625–51.
[13] Gal Y. Uncertainty in deep learning (Ph.D. thesis), University of Cambridge; 2016.
[14] Efron B. Bootstrap methods: Another look at the jackknife. Ann Statist 1979;7(1):1–26.
[15] Lakshminarayanan B, Pritzel A, Blundell C. Simple and scalable predictive uncertainty estimation using deep ensembles. In: Proceedings of the 31st international conference on neural information processing systems. 2016, p. 6405—6416.
[16] Gal Y, Ghahramani Z. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In: 33rd International conference on machine learning, vol. 3. 2016, p. 1651–60.
[17] Mancini T, Calvo-Pardo HF, Olmo J. Prediction intervals for deep neural networks. 2020, arXiv, arXiv:2010.04044.

[18] Gendoo DM, Zon M, Sandhu V, Manem VS, Ratanasirigulchai N, Chen GM, et al. MetaGxData: Clinically annotated breast, ovarian and pancreatic cancer datasets and their use in generating a multi-cancer gene signature. Sci Rep 2019;9:1–14. http://dx.doi.org/10.1038/s41598-019-45165-4.

[19] Cai L, Lin SY, Girard L, Zhou Y, Yang L, Ci B, et al. LCE: an open web portal to explore gene expression and clinical associations in lung cancer. Oncogene 2019;38:2551–64. http://dx.doi.org/10.1038/s41388-018-0588-2.

[20] Cox DR. Regression models and life-tables. J R Stat Soc Ser B Stat Methodol 1972;34(2):187–220.

[21] Tibshirani R. Regression shrinkage and selection via the Lasso. J R Stat Soc Ser B Stat Methodol 1996;58(1):267–88.

[22] Tibshirani R. The Lasso method for variable selection in the Cox model. Stat Med 1997;16(4):385–95. http://dx.doi.org/10.1002/(SICI)1097-0258(19970228)16:4<385::AID-SIM380>3.0.CO;2-3.

[23] Ternès N, Rotolo F, Michiels S. Robust estimation of the expected survival probabilities from high-dimensional cox models with biomarker-by-treatment interactions in randomized clinical trials. BMC Med Res Methodol 2017;17. http://dx.doi.org/10.1186/s12874-017-0354-0.

[24] Sparapani RA, Logan BR, McCulloch RE, Laud PW. Nonparametric survival analysis using Bayesian Additive Regression Trees (BART). Stat Med 2016;35:2741–53. http://dx.doi.org/10.1002/sim.6893.

[25] Feng D, Zhao L. Bdnnsurv: bayesian deep neural networks for survival analysis using pseudo values. Journal of Data Science 2021;19(4):542–54. http://dx.doi.org/10.6339/21-JDS1018.

[26] Efron B, Tibshirani RJ. An introduction to the bootstrap. Monographs on statistics and applied probability, (no. 57). Boca Raton, Florida, USA: Chapman & Hall/CRC; 1993.

[27] Uno H, Cai T, Pencina M, D'Agostino R, Wei L. On the C-statistics for evaluating overall adequacy of risk prediction procedures with censored survival data. Stat Med 2011;30:1105–17. http://dx.doi.org/10.1002/sim.4154.

[28] Graf E, Schmoor C, Sauerbrei W, Schumacher M. Assessment and comparison of prognostic classification schemes for survival data. Stat Med 1999;18:2529–45. http://dx.doi.org/10.1002/(sici)1097-0258(19990915/30)18:17/18<2529::aid-sim274>3.3.co;2-x.

[29] Bergstra J, Bardenet R, Bengio Y, Kégl B. Algorithms for hyper-parameter optimization. In: Advances in neural information processing systems 24: 25th annual conference on neural information processing systems 2011, vol. 24. NIPS 2011, 2011, p. 1–9.

[30] Lee ET, Go OT. Survival analysis in public health research. Annu Rev Public Health 1997;18(1):105–34. http://dx.doi.org/10.1146/annurev.publhealth.18.1.105.

[31] Bender R, Augustin T, Blettner M. Generating survival times to simulate Cox proportional hazards models. Stat Med 2005;24:1713–23. http://dx.doi.org/10.1002/sim.2059.

[32] Kvamme H, Borgan O. Continuous and discrete-time survival prediction with neural networks. Lifetime Data Anal 2021;27:1–27. http://dx.doi.org/10.1007/s10985-021-09532-6.

[33] Bourgon R, Gentleman R, Huber W. Independent filtering increases detection power for high-throughput experiments. Proc Natl Acad Sci USA 2010;107:9546–51. http://dx.doi.org/10.1073/pnas.0914005107.

[34] van Buuren S, Groothuis-Oudshoorn K. Mice: Multivariate imputation by chained equations in R. J Stat Softw 2011;45:1–67. http://dx.doi.org/10.18637/jss.v045.i03.

[35] Dusenberry MW, Tran D, Choi E, Kemp J, Nixon J, Jerfel G, et al. Analyzing the role of model uncertainty for electronic health records. In: Proceedings of the ACM conference on health, inference, and learning. New York, NY, USA: Association for Computing Machinery; 2020, p. 204–13. http://dx.doi.org/10.1145/3368555.3384457.