

Cox regression (recap)

We have a sample of n individuals, and let $N_i(t)$ count the **observed** occurrences of the event of interest for individual i as a function of (study) time t

Assume that the intensity process N_i may be given as

$$\lambda_i(t) = \underbrace{Y_i(t)}_{\text{at risk indicator}} \cdot \underbrace{\alpha(t | \mathbf{x}_i)}_{\text{hazard rate}}$$

where $\alpha(t|\mathbf{x}_i) = \alpha_0(t) \exp(\boldsymbol{\beta}^T \mathbf{x}_i)$

Here $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{ip})^T$ is a vector of fixed covariates and $\boldsymbol{\beta} = (\beta_1, \beta_2, \dots, \beta_p)^T$ is a vector of regression coefficients

Note that $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{ip})^T$ may contain interactions between and/or transformations of the original covariates

We estimate $\boldsymbol{\beta}$ by maximizing Cox's partial likelihood

$$L(\boldsymbol{\beta}) = \prod_{T_j} \frac{\exp(\boldsymbol{\beta}^T \mathbf{x}_{i_j})}{\sum_{l \in \mathcal{R}_j} \exp(\boldsymbol{\beta}^T \mathbf{x}_l)}$$

where i_j is the index of the individual who experiences an event at time T_j and $\mathcal{R}_j = \{l \mid Y_l(T_j) = 1\}$ is the **risk set**

We estimate $A_0(t) = \int_0^t \alpha_0(u) du$ by the Breslow estimator

$$\hat{A}_0(t) = \sum_{T_j \leq t} \frac{1}{\sum_{l \in \mathcal{R}_j} \exp(\hat{\boldsymbol{\beta}}^T \mathbf{x}_l)}$$

where $T_1 < T_2 < \dots$ are the event times

The cumulative hazard corresponding to an individual with a given covariate vector $\mathbf{x}_0$ is

$$A(t | \mathbf{x}_0) = \int_0^t \alpha(u | \mathbf{x}_0) du = \int_0^t \exp(\boldsymbol{\beta}^T \mathbf{x}_0) \alpha_0(u) du$$

and it may be estimated by

$$\hat{A}(t | \mathbf{x}_0) = \int_0^t \exp(\hat{\boldsymbol{\beta}}^T \mathbf{x}_0) d\hat{A}_0(u) = \exp(\hat{\boldsymbol{\beta}}^T \mathbf{x}_0) \hat{A}_0(t)$$

The survival function corresponding to an individual with a given covariate vector $\mathbf{x}_0$ is

$$S(t | \mathbf{x}_0) = \exp\{-A(t | \mathbf{x}_0)\}$$

It may be estimated by a Kaplan-Meier type estimator, or by

$$\tilde{S}(t | \mathbf{x}_0) = \exp\{-\hat{A}(t | \mathbf{x}_0)\}$$

Time-to-event prediction

The aim of a Cox analysis may be

- **to understand** how the various covariates affect the survival distribution
- **to predict** the time of a future event for an individual with given covariates

In medical applications of the Cox model, the focus is often on the first of these aims

In this lecture, we will focus on the second aim, that is on **time-to-event prediction**

To this end, we will consider methods for estimation of the survival distribution $S(t|\mathbf{x}_0)$ for an individual with given values of the (original) covariates $\mathbf{x}_0 = (x_{01}, \dots, x_{0p})^T$

A number of methods for time-to-event prediction has been proposed in the literature (an overview of some of them is given in the paper by Kvamme et al., 2019)

We will mainly consider methods that are based on Cox's regression model

For the Cox model we have $\alpha(t|\mathbf{x}) = \alpha_0(t) \exp(\boldsymbol{\beta}^T \mathbf{x})$

We obtain a more flexible model if we replace the linear predictor $\boldsymbol{\beta}^T \mathbf{x}$ by a **neural net** $g(\mathbf{x})$

This leads to a method known as **DeepSurv**

Before we discuss this (and other methods), we will have a brief look at neural nets

Neural networks

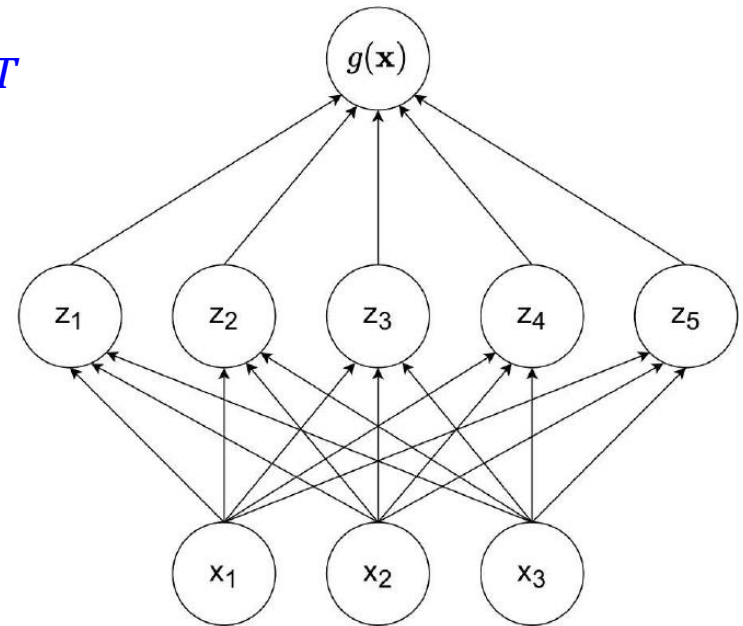
We restrict attention to the simple case of multilayer perceptrons (MLPs), which consist of one or more fully connected hidden layers

Input covariates $\mathbf{x} = (x_1, \dots, x_p)^T$

First hidden layer is $\mathbf{z} = (z_1, \dots, z_q)^T$
with $z_j = h_1(\boldsymbol{\theta}_{1j}^T \mathbf{x})$ where h_1 is an
activation function, for example,
 $h_1(x) = \max(0, x)$

(Similarly for other hidden layers)

Output $g(\mathbf{x}) = h_2(\boldsymbol{\theta}_2^T \mathbf{z})$



MLP with one hidden layer

The assess the fit of a neural network, we consider a **loss function** that compares the model with data; a common choice being **the negative of the log-likelihood function**

Neural networks typically contain a **large number of parameters**, so we need to be careful to avoid overfitting

To this end we may, for large data sets, split the data in three parts: a **training set**, a **validation set** and a **test set**

We then «minimize» the loss function on the training data by some version of **stochastic gradient descent**, and stop when the loss function takes its minimum value for the validation data

Finally we use the held-out test data to quantify the performance of the fitted model

For data sets that are not large, we may use (e.g.) five fold cross-validation to train and validate the network, before we quantify its performance on a held-out test set

DeepSurv

The DeepSurv method assumes a model of the form

$$\alpha(t|\mathbf{x}) = \alpha_0(t) \exp(g(\mathbf{x}))$$

where $g(\mathbf{x})$ is a MLP

Remember the assumptions underlying Cox regression:

1) **Log-linearity:** $\log\{\alpha(t|\mathbf{x})\} = \log\{\alpha_0(t)\} + \boldsymbol{\beta}^T \mathbf{x}$

2) **Proportional hazards:**

$$\frac{\alpha(t|\mathbf{x}_2)}{\alpha(t|\mathbf{x}_1)} = \exp\{\boldsymbol{\beta}^T (\mathbf{x}_2 - \mathbf{x}_1)\} \quad (\text{independent of time})$$

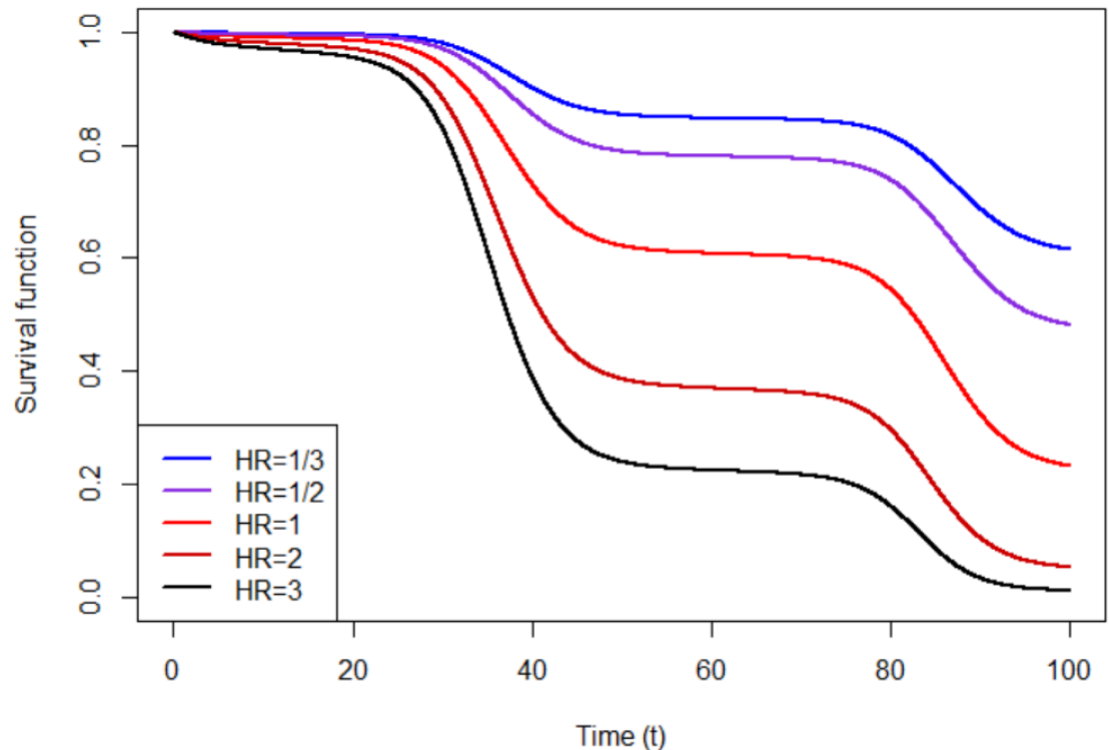
DeepSurv addresses the first of these assumptions, but not the second one

Let $S_0(t) = \exp\{-A_0(t)\}$ be the survival function corresponding to the baseline hazard, and let $HR = \exp(g(\mathbf{x}))$, then the survival functions we may obtain using DeepSurv are given by

$$S(t|\mathbf{x}) = \exp\{-A_0(t)HR\} = S_0(t)^{HR}$$

The DeepSurv
model is not
very flexible

To obtain a more
flexible model,
we need to also
address the
assumption of
proportional
hazard



Non-proportional hazards in Cox regression

For the classical Cox model, we may obtain non-proportional hazards by considering **time-varying covariates**

We then assume that the hazard takes the form

$$\alpha(t|\mathbf{x}(t)) = \alpha_0(t) \exp\{\boldsymbol{\beta}^T \mathbf{x}(t)\}$$

where $\mathbf{x}(t) = (x_1(t), x_2(t), \dots, x_p(t))^T$ is a vector of time-varying covariates

For example, we may have $x_j(t) = x \cdot t$ or $x_j(t) = x \cdot \log t$, where x is one of the original covariates

We obtain a more flexible non-proportional model if we replace the time-dependent linear predictor $\boldsymbol{\beta}^T \mathbf{x}(t)$ in by a **neural net** $g(t, \mathbf{x})$ that also includes time

For such a model, the partial likelihood takes the form

$$L = \prod_{T_j} \frac{\exp(g(T_j, \mathbf{x}_{i_j}))}{\sum_{l \in \mathcal{R}_j} \exp(g(T_j, \mathbf{x}_l))}$$

Note that we here have to evaluate the sum in the denominator for each of the event times $T_1 < T_2 < \dots$ so this is computationally quite heavy for large data sets

It is computationally more efficient to use a **case-control** type partial likelihood

For each event time T_j we then sample at random m controls among those at risk at time T_j (excluding the case i_j) and let the **sampled risk set** $\tilde{\mathcal{R}}_j$ consist of the case i_j and the m sampled control (m is small)

We may then use the case-control partial likelihood

$$L = \prod_{T_j} \frac{\exp(g(T_j, \mathbf{x}_{i_j}))}{\sum_{l \in \tilde{\mathcal{R}}_j} \exp(g(T_j, \mathbf{x}_l))}$$

which requires much less computations , and also allows the partial likelihood to be computed in batches (by restricting the sampling of controls to the same batch as the case)

We may then fit a MLP $g(t, \mathbf{x})$ using the loss function

$$\text{loss} = -\log L + \lambda \sum_{T_j} \sum_{l \in \tilde{\mathcal{R}}_j} |g(T_j, \mathbf{x}_l)|$$

Unlike case-control studies in epidemiology, we here sample new controls at each iteration

The survival function corresponding to an individual with a given covariate vector $\mathbf{x}_0$ may now be estimated as

$$\tilde{S}(t | \mathbf{x}_0) = \exp \left\{ -\hat{A}(t | \mathbf{x}_0) \right\}$$

where

$$\hat{A}(t | \mathbf{x}_0) = \int_0^t \exp(\hat{g}(u, \mathbf{x}_0)) d\hat{A}_0(u)$$

and

$$\hat{A}_0(t) = \sum_{T_j \leq t} \frac{1}{\sum_{l \in \mathcal{R}_j} \exp(\hat{g}(T_j, \mathbf{x}_0))}$$

is a Breslow type estimator

The resulting method is denoted **Cox-Time**

Comparison of prediction methods

We will compare the performance a number of methods for time-to-event prediction using four published data set

The methods we consider are **Classical Cox regression**, **DeepSurv**, **Cox-Time**, **DeepHit** and **Random Survival Forest (RSF)**

DeepHit is a machine learning method (described in the paper by Kvamme et al 2019) which is fitted using a loss function that combines a negative log-likelihood and a penalty for discordant predictions (cf. later slides)

RSF is a much used method for time-to-event prediction. The method is based on an ensemble of survival trees grown on bootstrapped censored survival data

Evaluation criteria

For comparing the methods, we will use the **concordance index (C-index)** and the **Brier score**

The **C-index** is an estimate of the probability that, for a random pair of individuals, the predicted survival times of the two individuals have the same ordering as the true survival times

Using the notation of the first lecture, we will use a version of the C-index that estimates the probability

$$C^{\text{td}} = P\{\tilde{S}(\tilde{T}_i|\mathbf{x}_i) < \tilde{S}(\tilde{T}_i|\mathbf{x}_j) \mid \tilde{T}_i < \tilde{T}_j, D_i = 1\}$$

For the **Brier score** we will first consider the situation without censoring

We then have survival times $T_1^*, T_2^*, \dots, T_n^*$ for n individuals, and for each individual i we have a predicted survival function $\tilde{S}(t|\mathbf{x}_i)$

Then the Brier score at time t is given by

$$\begin{aligned} \text{BS}(t) &= \frac{1}{n} \sum_{i=1}^n \left[I(T_i^* > t) - \tilde{S}(t|\mathbf{x}_i) \right]^2 \\ &= \frac{1}{n} \sum_{i=1}^n \left[\tilde{S}(t|\mathbf{x}_i)^2 I(T_i^* \leq t) - \{1 - \tilde{S}(t|\mathbf{x}_i)\}^2 I(T_i^* > t) \right] \end{aligned}$$

To handle censoring, we use an inverse probability of censoring weighting (IPCW)

To this end let $\hat{G}(t)$ be the Kaplan-Meier estimate of the censoring survival function

Then the IPCW version of the Brier score is given by

$$BS(t) = \frac{1}{n} \sum_{i=1}^n \left[\frac{\tilde{S}(t|\mathbf{x}_i)^2 I(\tilde{T}_i \leq t, D_i = 1)}{\hat{G}(\tilde{T}_i)} - \frac{\{1 - \tilde{S}(t|\mathbf{x}_i)\}^2 I(\tilde{T}_i > t)}{\hat{G}(t)} \right]$$

The Brier score is defined for a given time t

To obtain a criterion that is relevant over a time interval $[t_1, t_2]$, we may consider the integrated Brier score

$$IBS = \int_{t_1}^{t_2} BS(t) dt$$

Four published data sets

We will study the performance of the methods of four published data sets:

Data set	Size	Covariates	Prop. Censored
SUPPORT	8,873	14	0.32
METABRIC	1,904	9	0.42
Rot. & GBSG	2,232	7	0.43
FLCHAIN	6,524	8	0.70

The methods depend on a number of **hyperparameters** (not the same for all methods). For example, number of hidden layers and number of nodes per layer

See Kvamme et al (2019) for a description on how the hyperparameters have been tuned

Results for the C-index (large values are good)

Method	SUPPORT	METABRIC	Rot. & GBSG	FLCHAIN
Classical Cox	0.598	0.628	0.666	0.790
Cox-MLP (DeepSurv)	0.611	0.636	0.674	0.790
Cox-MLP (CC)	0.613	0.643	0.669	0.793
Cox-Time	0.629	0.662	0.677	0.790
DeepHit	0.642	0.675	0.670	0.792
RSF	0.628	0.649	0.667	0.784

Results for the integrated Brier score (low values is good)

Method	SUPPORT	METABRIC	Rot. & GBSG	FLCHAIN
Classical Cox	0.217	0.183	0.180	0.096
Cox-MLP (DeepSurv)	0.214	0.176	0.170	0.092
Cox-MLP (CC)	0.213	0.174	0.171	0.093
Cox-Time	0.212	0.172	0.169	0.102
DeepHit	0.223	0.184	0.178	0.120
RSF	0.215	0.175	0.171	0.093

KKBox data

KKBox is a music streaming service (similar to Spotify) that is popular in Southeast Asia

We use a data set with information on more than 2 million customers

Information on the customers (covariates) include age, subscription price, age, gender, city, number of previous churns (in all we use 15 covariates)

Our aim is to predict the time to cancelation of a subscription (churn)

We split the data set into training, validation and test sets as follows:

Data set	Size	Churned	Censored	Prop. Censor	Unique users
Train	1,786,333	1,279,358	506,975	0.28	1,582,202
Test	661,748	473,489	188,259	0.28	586,001
Validation	198,665	142,104	56,561	0.28	175,801

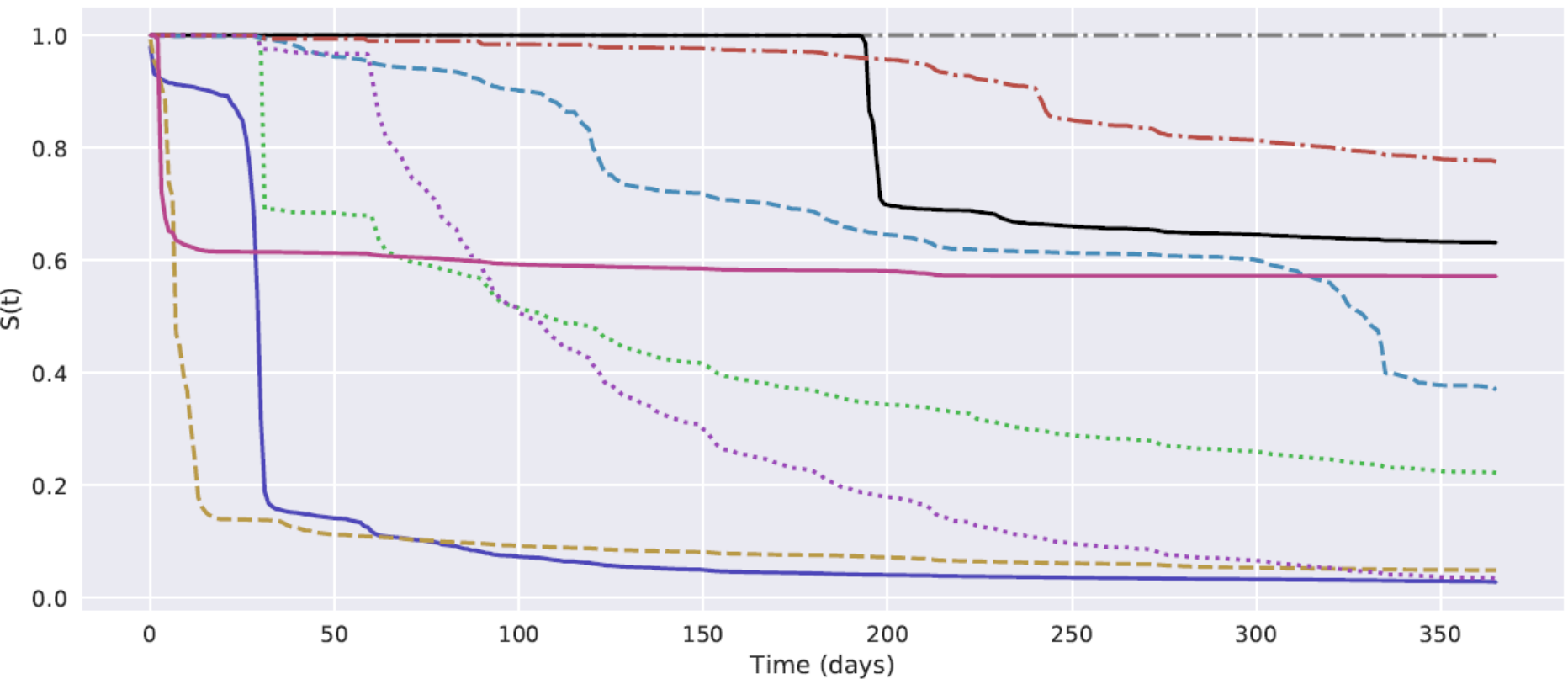
Best number of hidden layers and nodes per layer for the MLP-based methods

Method	Layers	Nodes
Cox-MLP (DeepSurv)	6	256
Cox-MLP (CC)	6	128
Cox-Time	8	256
DeepHit	6	512

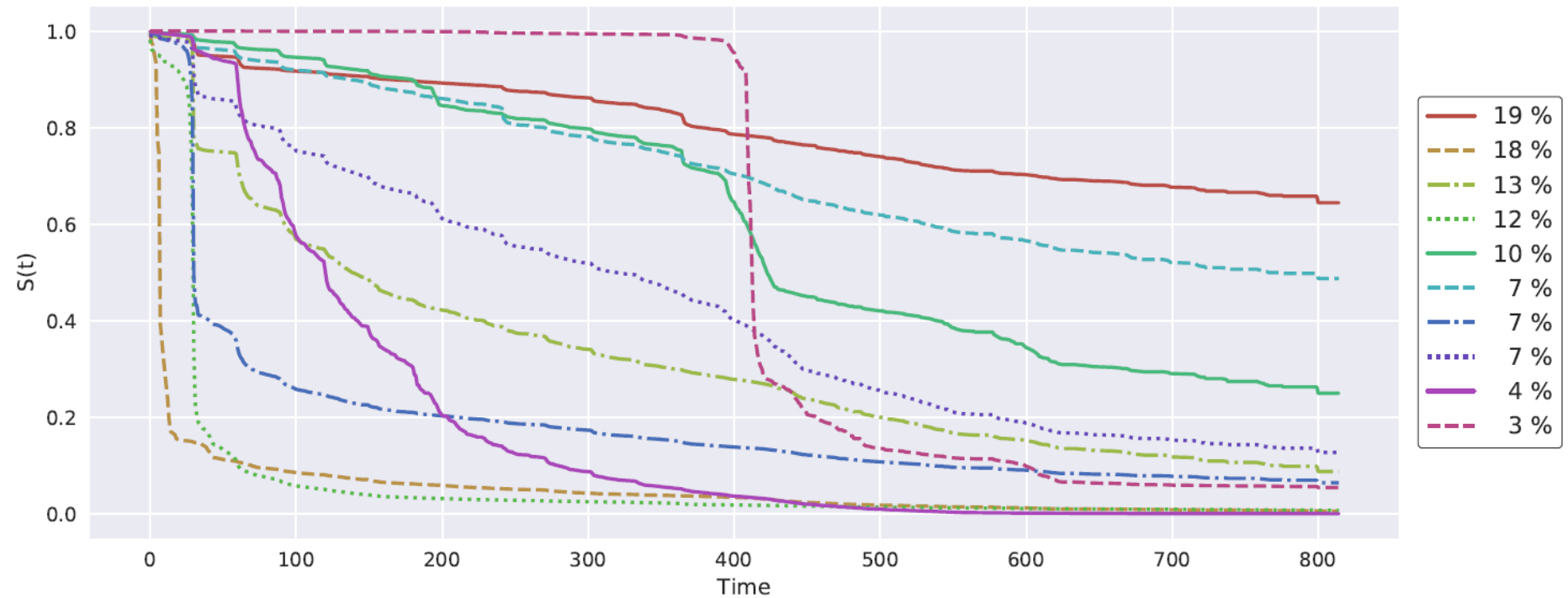
Performance of the methods

Method	C^{td}	IBS
Cox-SGD (Linear)	0.816	0.127
Cox-MLP (DeepSurv)	0.841	0.111
Cox-MLP (CC)	0.844	0.119
Cox-Time	0.861	0.107
DeepHit	0.888	0.147
RSF	0.855	0.112

Nine examples of survival predictions with Cox-Time



Grouped survival predictions with Cox-Time (grouped by K-means clustering)



<https://github.com/havakv/pycox>



Time-to-event prediction with PyTorch



[GitHub Actions status](#)

pypi

v0.3.0

conda-forge

v0.3.0

python

missing

License

BSD 2-Clause

[Get Started](#) • [Methods](#) • [Evaluation Criteria](#) • [Datasets](#) • [Installation](#) • [References](#)

pycox is a python package for survival analysis and time-to-event prediction with [PyTorch](#), built on the [torch tuples](#) package for training PyTorch models. An R version of this package is available at [survivalmodels](#).

The package contains implementations of various [survival models](#), some useful [evaluation metrics](#), and a collection of [event-time datasets](#). In addition, some useful preprocessing tools are available in the `pycox.preprocessing` module.