FastCPH: Efficient Survival Analysis for Neural Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

The Cox proportional hazards model is a canonical method in survival analysis for prediction of the life expectancy of a patient given clinical or genetic covariates – it is a linear model in its original form. In recent years, several methods have been proposed to generalize the Cox model to neural networks, but none of these are both numerically correct and computationally efficient. We propose *FastCPH*, a new method that runs in linear time and supports both the standard Breslow and Efron methods for tied events. We also demonstrate the performance of *FastCPH* combined with *LassoNet*, a neural network that provides interpretability through feature sparsity, on survival datasets. The final procedure is efficient, selects useful covariates and outperforms existing CoxPH approaches.

1 Introduction

Survival analysis studies the dependency of the time to the occurrence of an event on predictor variables. We usually call the estimated period "duration" and the event of interest "death" or "failure." Censored data, where the endpoint of observation is not a failure, is an important component in this field and requires specialized techniques [1].

The Cox proportional hazards model (CoxPH) is a classic semi-parametric method to handle censored data [1]. It was originally used as a linear regression model, supposing the log risk of failure is a linear combination of the clinical or genetic predictor variables. Its core idea is that the dependency of hazard rate on covariates is time-invariant and multiplicative. The formulation of CoxPH likelihood is explained in Section 2. CoxPH is developed under the assumption of continuous data, but there are often tied event times in real datasets. Several ways have been offered to deal with instances where there are ties, including the exact partial likelihood, the Breslow approximation, and the Efron approximation, the latter two being well-tested in theory and widely used in practice [2, 3, 4].

In classical survival studies using CoxPH, a lot of effort is needed in feature engineering to make the model work well. Over the years, several methods were proposed to generalize it to nonlinear situations, allowing more complex formulation for the log-risk function, yet having mixed results [5, 6, 7]. We review the four most popular implementations of the CoxPH model in neural networks as in Table 1: **DeepSurv [8]** implements a deep learning generalization of the Cox proportional hazards model using Theano and Lasagne. It supports tensor operation, runs in $\mathcal{O}(n)$ by vectorized cumulative sums over the entire input, but assumes there are no tied events. **Pycox [9]** computes in $\mathcal{O}(n)$ a cumulative sum of all input hazards but not the true risk sets of the CoxPH function. **PySurvival [10]** is a PyTorch compatible implementation based on Deepsurv [8], while adding an $\mathcal{O}(n^2)$ index matrix for Efron's method to handle ties. **Scikit-survival [11]** (sksurv) is package released in 2020 [12] with Brewslow and Efron approximation in $\mathcal{O}(n)$ using an inner for-loop at each distinct event time when going through all events. However, it does not support deep learning.

[1] We refer censored data as right censoring (the most common type) in this work.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

Table 1: Comparisons for different Cox PH implementations

	Sksurv	PyCox	DeepSurv	PySurvival	***FastCPH* (Ours)**
Time complexity	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$
Deep learning	✗	✓	✓	✗	✓
Handling ties					
Tie-awareness	✓	✗	✗	✓	✓
Efron approximation	✓	✗	✗	✓	✓
Breslow approximation	✓	✗	✗	✗	✓

Simply assuming the absence of ties or ignoring all tied cases is statistically inappropriate. Failure to handle ties and oversimplifications may cause unexpected consequences, especially when the behaviors of neural networks possess randomness and the results can be hard to interpret [4, 13]. However, to the best of our knowledge, none of the existing popular survival analysis packages have CoxPH implemented in both an efficient and correct way for neural networks.

Here we present the Fast Cox Proportional Hazard model (*FastCPH*), a computationally efficient and statistically correct method for survival analysis using neural networks.*FastCPH* is a fully vectorized method that runs in $\mathcal{O}(n)$ and yields both standard Breslow and Efron methods for tied events, which overcomes the limitations of existing CoxPH methods. We implement it with PyTorch and it can be easily used for any other machine learning research requiring the CoxPH model, allowing computationally efficient deep learning training for a larger scale of data. We also combine it with *LassoNet* to present *FastCPH-LassoNet*, a simple neural network that provides interpretability through feature sparsity in survival analysis. We evaluate *FastCPH-LassoNet* on multiple survival datasets, and *FastCPH-LassoNet* outperforms existing CoxPH approaches.

2 Fast Cox Proportional Hazards Model

We propose *FastCPH* as a method for efficient survival analysis for neural networks. As noted in Table 1, *FastCPH* is the first CoxPH method that is both computationally efficient for neural networks and supports Efron and Breslow approximation in handling tied events. We also provide a step-by-step proof verifying the correctness of our tensor-based implementation.

Definition 2.1. The input is given as a feature matrix $\mathbf{x}$ where each row is a sample, and each column is a feature. Each row is associated with an event time t_i (that can produce ties) and an indicator δ_i indicating whether the event is censored or not (1 if uncensored). Given a regression model that gives a real number $g(x_i)$ for each sample, the CoxPH likelihood that is maximized in the absence of ties is:

$$L(g) = \prod_{i|\delta_i=1} \frac{\exp(g(\mathbf{x_i}))}{\sum_{j|t_j \geq t_i} \exp(g(\mathbf{x_j}))} \tag{1}$$

The negative log likelihood (loss function) is

$$LL(g) = \sum_{i \in J} \log \left(\sum_{j \in R_i} \exp\left[g(\mathbf{x_j})\right] \right) - g(\mathbf{x_i}) \tag{2}$$

In implementation, assuming the event times t are sorted in decreasing order (which is a $\mathcal{O}(n \log n)$ preprocessing), we can compute all values of $\log\left(\sum_{j \in R_i} \exp\left[g(\mathbf{x_j})\right]\right)$ in $\mathcal{O}(n)$ by using the `logcumsumexp` function[2] (as implemented in PyTorch [14]):

$$LL(g) = \sum_{i \in J} \texttt{logcumsumexp}(g(\mathbf{x}))_i - g(\mathbf{x_i}) \tag{3}$$

Breslow's method When there are ties, the theoretical best solution would assume that the events still happened in some order and sum the formula without ties over all orders. This is not efficient because there are $d_i!$ possible orders for each tie, thus rarely used in applications. Breslow approximation assumes that all d_i elements were selected from the same risk set. Thus, the above formula stays unchanged. The implementation simply indexes the `logcumsumexp` terms so that events with the same failure time have the same denominator.

[2] $\texttt{logcumsumexp}(g(\mathbf{x}))_i$ is a slight abuse of notation as `logcumsumexp` is applied on all events, not just those from J, then indexed on the elements of J.

Table 2: Performance of different CoxPH models on survival datasets (in percentage, 95% CI if indicated). ♢: Models with simple linear architecture. ♣: Models with more complex structures, best results from (16,16), (32), (32, 16), (64).

	Breast cancer	WHAS500	Veteran's lung cancer	HNSCC
CoxPH Linear	51.4	71.3	66.4	59.1
CoxNet	57.0	71.4	72.6	74.3
GlmNet	60.3 (±4.67)	70.1 (±0.68)	70.7 (±1.34)	**74.8** (±2.18)
DeepSurv♢	57.8 (±1.52)	70.0 (±2.05)	66.1 (±3.14)	73.1 (±1.53)
DeepSurv♣	68.7 (±1.20)	66.1 (±1.07)	69.3 (±0.72)	61.6 (±3.05)
FastCPH-LassoNet ♢	67.0 (±5.39)	76.6 (±1.21)	71.9 (±1.90)	70.1 (±3.96)
FastCPH-LassoNet ♣	**69.7** (±5.35)	**76.8** (±1.40)	**73.0** (±2.49)	69.3 (±4.68)

Table 3: Hyperparameters and run time of *FastCPH-LassoNet* (linear structure, 95% CI if indicated). Run time is in seconds for per run per CPU. Training performed on 2.8 GHz Quad-Core Intel Core i7 with 16 GB memory.

	Breast cancer	WHAS500	Veteran's lung cancer	HNSCC
# selected features	24.6 (±2.95)	14.0 (±0.00)	10.6 (±0.78)	11.0 (±3.95)
# total features	80	14	11	107
run time	261s	225s	201s	283s

Efron's method observes that the denominator is too big in Breslow's approximation, as when multiples elements are selected from the same risk set, that risk set gets smaller.

Definition 2.2. Efron's approximation results in the following likelihood:

$$L(g) = \prod_{i \in J'} \frac{\prod_{j \in D_i} \exp(g(\mathbf{x_i}))}{\prod_{k=0}^{d_i - 1} \left(\sum_{j \in R_i} \exp(g(\mathbf{x_j})) - \frac{k}{d_i} \sum_{j \in D_i} \exp(g(\mathbf{x_j})) \right)} \tag{4}$$

where J' is a subset of J with unique event times. The idea is to discount the denominator over the whole risk set $\sum_{j \in R_i} \exp(g(\mathbf{x_j}))$ by the average risk over D_i: $\frac{1}{d_i} \sum_{j \in D_i} \exp(g(\mathbf{x_j}))$. k indexes the set D_i.

Compared with Equation 2, the numerator did not change (it is still the product over all uncensored events). The denominator has three terms:

- $\sum_{j \in R_i} \exp(g(\mathbf{x_j}))$ can be computed for all j in linear time as before.
- $\frac{k}{d_i}$ is trivial to compute for all elements of J (which is the union of all D_i for $i \in J'$).
- $\sum_{j \in D_i} \exp(g(\mathbf{x_i}))$ can also be computed in linear time with a vectorized scatter operation.

Finally, those terms are easy to combine in linear time with vectorized operations. All computations are realized in the log space to avoid numerical errors, using tricks similar to those used to implement the log-sum-exp operation. To summarize, we provide *FastCPH* as an efficient, vectorized and linear-time function implemented in PyTorch that can be conveniently used by any neural network.

FastCPH-LassoNet is proposed as a survival prediction method with feature sparsity. We use *LassoNet* as the backbone [15], which is a neural network that achieves feature sparsity using a LASSO-style regularization [16]. Originally, it was applied with mean squared error and cross-entropy losses for regression and classification problems. We apply *FastCPH* as the loss function for survival analysis. The implementation detail is in Section A.2 in the supplementary materials.

3 Experiments

In the experiments, we want to evaluate the following three aspects of *FastCPH*: Is the implementation of *FastCPH* computationally efficient for neural networks? Can *FastCPH-LassoNet* obtain

feature sparsity along the regularization path? Does *FastCPH-LassoNet* have promising performance compared to other CoxPH-based models on real-world survival datasets?

3.1 Runtime analysis

To analyze the computational efficiency of *FastCPH*, we compare its runtime with 3 other popular vectorized implementations including PyCox, DeepSurv, PySurvival. PySurvival use risk / fail matrices to compute Efron method. PyCox and DeepSurv are for deep learning purposes and do not support tie approximations.
Among these baselines, *FastCPH* is the most computationally efficient CoxPH implementation that supports Breslow and Efron approximations. Specifically, as shown in Fig 1, the curve of *FastCPH* with Breslow method is very close to the ones of PyCox and DeepSurv, which don't have tie awareness. As the size of data increases, the difference among *FastCPH* with Breslow, PyCox, and DeepSurv are getting smaller. For the two baselines both using Efron approximation, *FastCPH* with Efron method is significantly faster than PySurvival. The gap between *FastCPH* with Efron and Breslow can be caused by the computation of the weighted terms in denominator. The experimental results align with our claims on the linearity of *FastCPH*'s runtime.

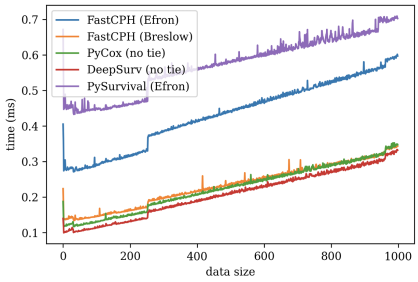

Figure 1: Runtime comparisons between different CoxPH implementations. The x-axis is the size of data, and the y-axis is the time for one-time CoxPH calculation in milliseconds

3.2 Evaluating *FastCPH-LassoNet* on real world's datasets

We compare *FastCPH-LassoNet* with other CoxPH-based methods on 4 datasets: breast cancer [17], WHAS500 [18], veteran's lung cancer [19], and HNSCC [20]. We use CoxPH linear model [1], CoxNet [21], GlmNet [22] and DeepSurv [8] as baselines. The first three are classical CoxPH-based models with different regularizations. DeepSurv is considered the most advanced deep learning CoxPH-based method [8, 23]. We use Harrell's concordance index (C-index) as a metric. We also provide a $\mathcal{O}(n \log n)$ implementation of the C-index using ordered data structures.
As in Table 2, *FastCPH-LassoNet* is the best CoxPH-based method on 3 out of 4 survival datasets we use, showing its discrimination ability to provide reliable ranking of survival times based on risk scores. Notice that for the breast cancer and WHAS500 datasets, *FastCPH-LassoNet* yields a significantly better C-index than DeepSurv and other existing methods in Table 2. Specifically, comparing *FastCPH-LassoNet* to DeepSurv, *FastCPH-LassoNet* performs better in 4 datasets under the same architecture. We supplement further comparisons between *FastCPH-LassoNet* and DeepSurv in Section A.2.
Moreover, *FastCPH-LassoNet* is able to attain an effective recovery of signals given a subset of features. As shown in Table 2 and 3, the model gives excellent performance with sparsity in covariates. However, when the number of total features is small, the model may not be able to obtain sparsity over covariates, as shown by its result of WHAS500. The sparsity demonstrated in the experiments implies its potential on large-scale, more complicated real world datasets.

4 Discussion

In this paper, we have proposed *FastCPH*, an efficient CoxPH method for survival analysis in neural networks that follows the exact formula of well-tested methods to handle tied events. *FastCPH* is an efficient and numerically correct solution for neural networks in survival analysis. It can be quickly adapted to other deep learning methods and used in more real-world scenarios with censored data such as [24, 25, 26]. We have shown *FastCPH-LassoNet* outperformed other CoxPH-based methods in various survival datasets. More study can be done to provide applications of *FastCPH* as an objective function in more complex neural network architectures. It will be interesting to see the effect of tied events on the behavior of neural networks. In addition, it is worth pointing out the importance of following the exact formulae of classic statistical methods in implementation and avoiding oversimplifications in the machine learning community.

A Appendix

This supplementary document is organized as the following. Firstly, we discuss the related work on CoxPH models, deep learning in survival analysis, and tie handling. Next, we discuss implementation details on the metric, settings, and baselines. We then provide additional information on datasets we used in the experiments, including an illustration of pipeline, covariate breakdown, and the correlation matrix of HNSCC dataset. Lastly, we provide additional experiment comparing *FastCPH-LassoNet* with DeepSurv in NN context. We also supplement the code for *FastCPH*.

A.1 Related Work

CoxPH models A major advantage of CoxPH models over methods like Kaplan-Meier curves and the log-rank test is their ability to work easily with quantitative predictor variables and ability to generalize patterns from censored data [27]. Therefore, they are particularly suitable for survival analysis and predictions and are applied extensively in the biomedical field including in analyzing gene expressions and the likelihood of various diseases including liver diseases, coronary heart disease, diabetes, etc [28, 29, 30, 31]. Beyond that, CoxPH models also have a variety of applications. When compared with the results of the multiple discriminant analysis methods, the CoxPH model gives lower type I errors [24].

Deep learning in survival analysis With the rise of deep learning applications in many scientific fields, some studies have tried to combine CoxPH functions with deep neural networks for better time-to-event predictions for larger datasets [8, 23, 32]. The inclusion of a neural network can simplify the *a priori* covariates selection and make the model adaptively learn them while preserving the effectiveness of CoxPH functions in survival analysis. Our work focuses on the deep learning method to model the survival hazards using CoxPH.

Handling ties When handling ties, the most commonly used approach is the Breslow method which simply uses the number of tied events as the exponent in the denominator of the relative risk [2]. The Efron method is thought to produce superior outcomes, yet the formulation is more complicated to implement efficiently [3]. The difference with the Breslow approach is minor when the number of ties is not large [33]. Our method supports both Efron's and Breslow's method.

A.2 Implementation Details

Metric Harrell's concordance index (C-index) is a generalization of Area under ROC curve (AUC) regarding censored data, reflecting the accuracy of pairwise orders of the risk function as the output of the model [34, 35, 36]. For input $\mathbf{x}_i$, duration t_i and events δ_i (1 if uncensored, 0 otherwise), the C-index is computed as:

$$C = \frac{\sum_{i,j:t_j<t_i} 1_{g(\mathbf{x}_j)>g(\mathbf{x}_i)}\delta_j}{\sum_{i,j:t_j<t_i} \delta_j}. \tag{5}$$

FastCPH-LassoNet is *LassoNet* combined with *FastCPH* as the objective function. Like LASSO, *LassoNet* penalizes the L^1 norm of coefficients applied to features. The Lagrange multiplier associated with that penalty is noted λ. The model is trained with increasing values of λ, on a dense-to-sparse path where the values of λ follow a geometric scale. The starting value of λ is a hyperparameter that should be carefully selected: if it is too small, the model will train on a lot of useless configurations; if it is too large, the optimizer will ignore features too fast. Another hyperparameter is M, the hierarchy coefficient that balances the linear and non-linear parts of the model. Fig 2 gives an example of the training curve of *FastCPH-LassoNet* on the breast cancer dataset. We can see *LassoNet* optimized properly with *FastCPH* as the loss function.

Runtime Analysis We assume all events are uncensored and the data is sorted by duration. We first randomly generate data from size 1 to 10^3. We note down the runtime as calculating the negative log likelihood once and report the mean of 5 random trails.

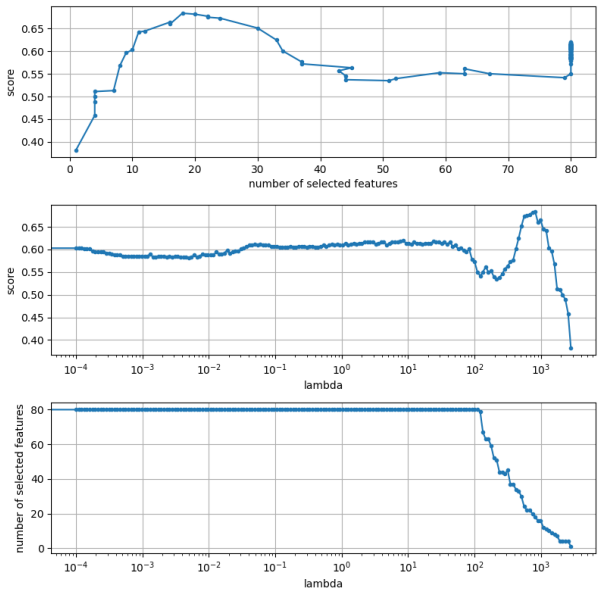

Figure 2: Demonstration of the training behavior of *FastCPH-LassoNet* on the breast cancer dataset. The y-axis of the first two rows represents the test score, which is calculated as the C-index. The model starts from a given λ. At each iteration, λ increases at a fixed geometric rate, the model is retrained based on the model selected at the last λ. At each λ, the iterated model can select a subset of features in the input, that is supposed to decrease. The first figure is the number of selected features versus the test C-index. The second figure represents the change in test score when λ increases during training. The third figure is the number of features selected when λ increases during training. We can see that the test score reaches a peak when λ is between 10^2 and 10^3. The test C-index fluctuates when the number of selected features decreases, and it reaches the global optimal when the number of features is 19.

Baseline implementation The implementations of the Cox linear model and CoxNet are from Scikit-survival [11]. For the CoxPH linear model, we set $\alpha = 10^{-6}$ as the regularization parameter in the ridge regression penalty. CoxNet is the CoxPH model with an elastic net penalty. We use cross-validation for choosing the best α of the regularization path from 10^{-1} to 10^{-5}. For GlmNet, we use the R built-in cross-validation `cv.glmnet` with the Breslow method to select the model for testing. The number of folds in cross-validation (if used) is 5 for breast cancer and veteran's lung cancer dataset and 10 for WHAS500 and HNSCC dataset. For *FastCPH-LassoNet* and DeepSurv, we fix the number of hidden dimensions to 1 and the number of hidden layers to 1. They share the same architecture and setting (ReLU, Adam, lr=10^{-3}). For *FastCPH-LassoNet*, we set $M = 10$ and start at $\lambda = 10^{-6}$. The `prox` method of *LassoNet* is called on the dense model following on a geometric path until the model becomes sparse. That value is divided by 10 to give `lambda_start`. 5-fold cross validation is used to select the best λ value from multiple runs. We use Efron's method to break ties. We use stratified sampling w.r.t uncensored/censored events to split the training set (80%) and test set (20%) for each of the datasets. For models with randomness in training, we run 5 trails for each set of hyperparameters and obtain the average performance.

A.3 Datasets statistics

We use the following four datasets in our experiments:

- **Breast cancer dataset [17]:** This dataset comes from experiments set up to validate a certain gene signature in primary breast tumors. It contains data points from 198 patients,

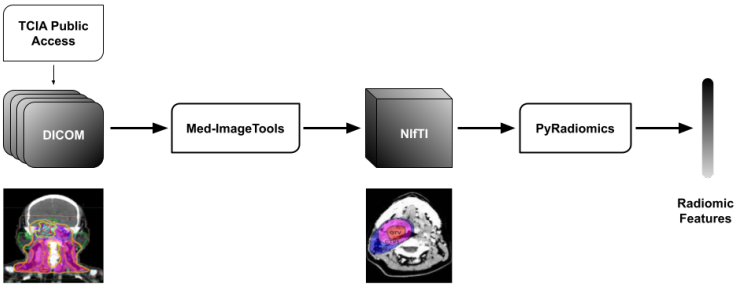

Figure 3: The pipeline of obtaining HNSCC features for survival analysis. HNSCC dataset was curated by the University of Texas MD Anderson Cancer Center, approved by the institutional review board, and written informed consent was obtained from all study participants. This dataset is available publicly upon submission of a limited data access agreement to safeguard patient privacy. While the dataset has been thoroughly de-identified in accordance with HIPAA, it is theoretically possible to reconstruct the face, head, or body using volumetric images. For our analysis, we ensured the data was processed on a HIPAA-compliant encrypted server and the radiomics features extracted for our analysis do not contain any identifiable information or offensive content.

Table 4: Event and clinical variable distribution of HNSCC.

# of patients	**451**
Outcome	
Alive / Censored	395 (88%)
Death	56 (12%)
Sex	
Male	387 (86%)
Female	64 (14%)
Disease subsite	
Base of Tongue	238 *(53%)*
Tonsil	174 *(39%)*
Glossopharyngeal sulcus	10 *(2%)*
Other	29 *(6%)*
HPV Status	
Positive	232 *(51%)*
Negative / Unknown	219 *(49%)*
Stage	
I	3 *(1%)*
II	14 *(3%)*
III	63 *(14%)*
IV	371 *(83%)*
Tumor Laterality	
R	222 *(49%)*
L	215 *(48%)*
Other	14 *(3%)*

with 80 features each. The endpoint of this dataset is distant metastases. Of all patients, 51 of them (25.8%) exhibited the symptom.

Table 5: Statistics on tied events in different datasets

	Breast cancer	WHAS500	Veteran's lung cancer	HNSCC
# total observations	198	500	137	451
# total tied events	6	178	64	232
# uncensored events	51	215	128	56
# uncensored tied events	0	80	55	4

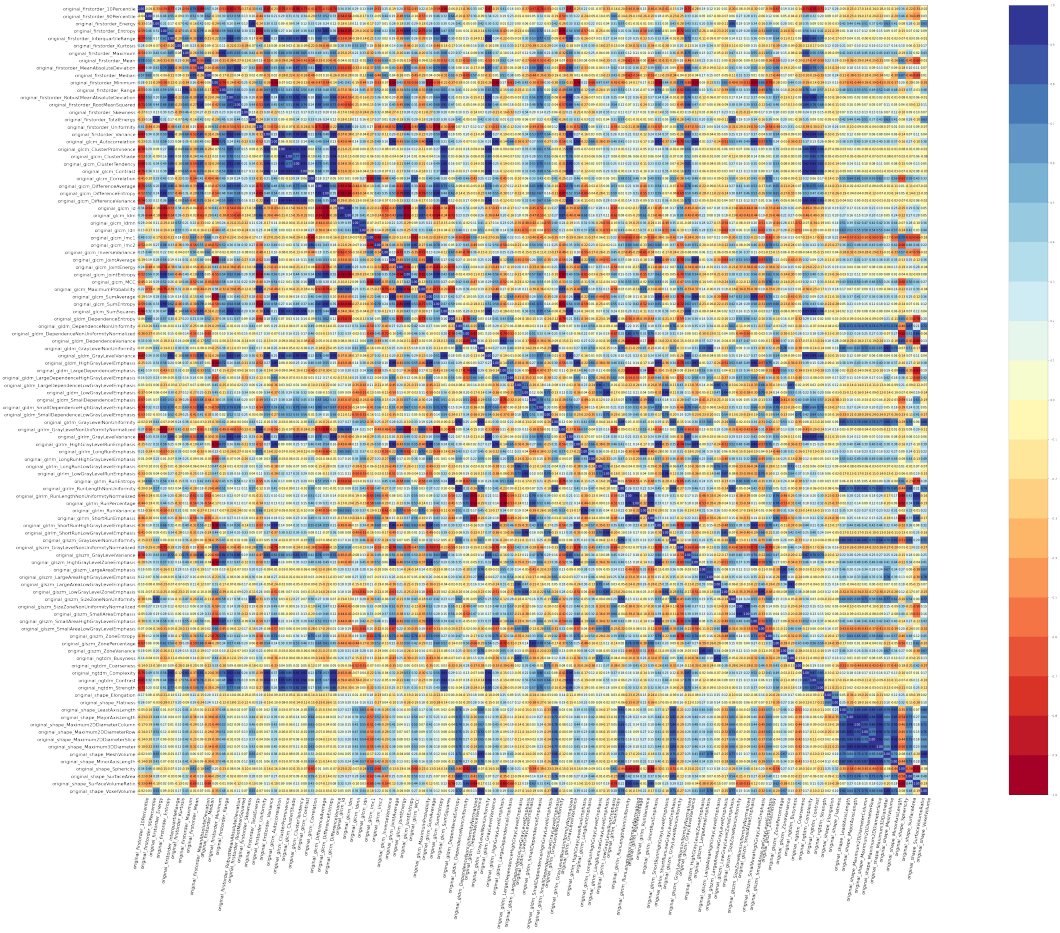

Figure 4: Correlation matrix of covariates in HNSCC. The red color at the bottom of the spectrum indicates -1 and blue color at the top means 1.

- **WHAS500 dataset [18]:** The Worcester Heart Attack Study dataset is an observational dataset set up to track trends in acute myocardial infarction and out-of-hospital coronary heart disease deaths in Worcester, Massachusetts. The endpoint of this dataset is death. Out of 500 patients in the dataset with 14 features each, the endpoint occurred for 215 patients (43.0%).
- **Veteran's lung cancer dataset [19]:** This dataset comes from a lung cancer trial by the Department of Veterans Affairs. The endpoint of this dataset is death. Out of 137 patients in the dataset with 6 features each, the endpoint occurred for 128 patients (93.4%).
- **HNSCC [20]:** This dataset is composed of 451 head and neck squamous cell carcinoma (HNSCC) patients treated with curative-intent intensity modulated radiotherapy (IMRT).

Table 6: Results of *FastCPH-LassoNet* and DeepSurv with more complex architectures (in percentage, 95% CI)

	Breast cancer	WHAS500	Veteran's lung cancer	HNSCC
(16, 16)				
FastCPH-LassoNet	**69.7** (±5.35)	76.6 (±1.35)	**73.0** (±2.49)	63.7 (±5.89)
DeepSurv	68.2 (±2.47)	64.5 (±1.06)	66.3 (±1.47)	61.6 (±3.05)
(32)				
FastCPH-LassoNet	67.4 (±4.90)	76.8 (±1.29)	72.6 (±2.12)	**69.3** (±4.68)
DeepSurv	66.6 (±1.27)	65.8 (±0.68)	69.3 (±0.72)	58.9 (±2.56)
(32, 16)				
FastCPH-LassoNet	69.0 (±3.47)	75.4 (±2.49)	71.9 (±3.23)	65.3 (±5.38)
DeepSurv	68.7 (±1.20)	66.1 (±1.07)	66.2 (±1.46)	57.1 (±2.39)
(64)				
FastCPH-LassoNet	68.6 (±5.11)	**76.8** (±1.40)	72.9 (±2.50)	69.0 (±4.22)
DeepSurv	67.0 (±1.24)	64.8 (±0.63)	67.3 (±1.13)	58.5 (±2.00)

This dataset was previously used to predict local recurrence and HPV status [37]. Survival analysis is done in data exploration, but nothing predictive. We include it as a showcase of our method. The endpoint used for our analysis is death. This is a challenging prediction dataset when adpating it for survival analysis. Fig 4 is a visualization of the correlation matrix of clinical covariates in HNSCC dataset generated by PySurvival [10]. As we can see in the figure, many of the covariates are heavily correlated with each other, posing a need of selecting useful features for constructing an efficient solution. According to a single value decomposition computation, the matrix is rank deficient. Despite that, *FastCPH-LassoNet* can successfully select a subset of covariates (11.04 out of 107) and attain a good and stable performance as shown in Table 2.

Ties are common in the datasets we use, listed as in Table 5. Breslow and Efron methods give the same log likelihood when no ties are present in the dataset. Our method in Table 2 using Breslow approximation is capable of handling datasets with and without ties.

Data preprocessing For the breast cancer dataset, WHAS500, and veteran's lung cancer dataset, we retrieve the data from Scikit-survival package [38] and obtain one-hot encodings to quantify entries such as treatment received, cell types, prior therapy, etc. The number of final entries is shown in the last row of Table 3. The HNSCC dataset is publicly available via the Cancer Imaging Archive with TCIA Limited Access License [20]. The DICOM imaging data is processed using the Med-ImageTools pipeline [39] to extract the computed tomography (CT) images and gross tumor volume (GTV) segmentation masks with uniform voxel spacing for consistent feature extraction. These images and masks are processed into the NIfTI file format, which is a common standard for 3D medical images, and compatible with PyRadiomics. The processed image and GTV masks into PyRadiomics to extract shape, texture, and statistics features [40].

A.4 *FastCPH-LassoNet* with more complex architectures

In the context of NN methods, we further analyze the performance of *FastCPH-LassoNet* using more complex architectures. We use DeepSurv as the baseline because it is commonly recognized as the most advanced CoxPH-based deep learning method. We let *FastCPH-LassoNet* and DeepSurv share the same architecture (as indicated in parentheses in Table 6) and setting (ReLU, Adam, `lr`=10^{-3}). For both methods, we run 15 trails to give 95% CI.
As shown in Table 6, *FastCPH-LassoNet* outperforms Deepsurv with the same architecture on all datasets in our experiments. It is a more robust deep learning method with promising results in survival analysis.

A.5 Code

```
"""
implement CoxPHLoss
```

```python
"""

class CoxPHLoss(torch.nn.Module):
    """Loss for CoxPH model. """

    allowed = ("breslow", "efron")

    def __init__(self, method):
        super().__init__()
        assert method in self.allowed, f"Method must be one of {self.allowed}"
        self.method = method

    def forward(self, log_h, y):
        log_h = log_h.flatten()

        durations, events = y.T

        # sort input
        durations, idx = durations.sort(descending=True)
        log_h = log_h[idx]
        events = events[idx]

        event_ind = events.nonzero().flatten()

        # numerator
        log_num = log_h[event_ind].mean()

        # logcumsumexp of events
        event_lcse = torch.logcumsumexp(log_h, dim=0)[event_ind]

        # number of events for each unique risk set
        _, tie_inverses, tie_count = torch.unique_consecutive(
            durations[event_ind], return_counts=True, return_inverse=True
        )

        # position of last event (lowest duration) of each unique risk set
        tie_pos = tie_count.cumsum(axis=0) - 1

        # logcumsumexp by tie for each event
        event_tie_lcse = event_lcse[tie_pos][tie_inverses]

        if self.method == "breslow":
            log_den = event_tie_lcse.mean()

        elif self.method == "efron":
            # based on https://bydmitry.github.io/efron-tensorflow.html

            # logsumexp of ties, duplicated within tie set
            tie_lse = scatter_logsumexp(log_h[event_ind], tie_inverses, dim=0)[
                tie_inverses
            ]
            # multiply (add in log space) with corrective factor
            aux = torch.ones_like(tie_inverses)
            aux[tie_pos[:-1] + 1] -= tie_count[:-1]
            event_id_in_tie = torch.cumsum(aux, dim=0) - 1
            discounted_tie_lse = (
                tie_lse
                + torch.log(event_id_in_tie)
                - torch.log(tie_count[tie_inverses])
```

```
            )

            # denominator
            log_den = log_substract(event_tie_lcse, discounted_tie_lse).mean()

        # loss is negative log likelihood
        return log_den - log_num


def concordance_index(risk, time, event):
    """
    O(n log n) implementation of https://square.github.io/pysurvival/metrics/c_index.html
    """
    assert len(risk) == len(time) == len(event)
    n = len(risk)
    order = sorted(range(n), key=time.__getitem__)
    past = SortedList()
    num = 0
    den = 0
    for i in order:
        num += len(past) - past.bisect_right(risk[i])
        den += len(past)
        if event[i]:
            past.add(risk[i])
    return num / den

"""
Cox Experiments
"""

from pathlib import Path
from time import time
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
from tqdm import tqdm

from sklearn import preprocessing
from sklearn.model_selection import train_test_split, StratifiedKFold

import sksurv.datasets

from lassonet import LassoNetCoxRegressorCV

from joblib import Parallel, delayed
from tqdm_joblib import tqdm_joblib

from lassonet import plot_cv
from lassonet.utils import confidence_interval

DATA_PATH = Path(__file__).parent / "data"
DATA_PATH.mkdir(exist_ok=True)

FIGURES_PATH = Path() / "cox_figures"
FIGURES_PATH.mkdir(exist_ok=True)


def transform_one_hot(input_matrix, col_name):
    one_hot_col = pd.get_dummies(input_matrix[col_name], prefix=col_name)
```

```
    input_matrix = input_matrix.drop([col_name], axis=1)
    input_matrix = input_matrix.join(one_hot_col)
    return input_matrix


def dump(array, name):
    pd.DataFrame(array).to_csv(DATA_PATH / name, index=False)


def gen_data(dataset):
    if dataset == "breast":
        X, y = sksurv.datasets.load_breast_cancer()
        di_er = {"negative": 0, "positive": 1}
        di_grade = {
            "poorly differentiated": -1,
            "intermediate": 0,
            "well differentiated": 1,
            "unkown": 0,
        }
        X = X.replace({"er": di_er, "grade": di_grade})
        y_temp = pd.DataFrame(y, columns=["t.tdm", "e.tdm"])
        di_event = {True: 1, False: 0}
        y_temp = y_temp.replace({"e.tdm": di_event})
        y = y_temp

    elif dataset == "fl_chain":
        X, y = sksurv.datasets.load_flchain()
        di_mgus = {"no": 0, "yes": 1}
        X = X.replace({"mgus": di_mgus, "creatinine": {np.nan: 0}})
        col_names = ["chapter", "sex", "sample.yr", "flc.grp"]
        for col_name in col_names:
            X = transform_one_hot(X, col_name)
        y_temp = pd.DataFrame(y, columns=["futime", "death"])
        di_event = {True: 0, False: 1}
        y_temp = y_temp.replace({"death": di_event})
        y = y_temp

    elif dataset == "whas500":
        X, y = sksurv.datasets.load_whas500()
        y_temp = pd.DataFrame(y, columns=["lenfol", "fstat"])
        di_event = {True: 1, False: 0}
        y_temp = y_temp.replace({"fstat": di_event})
        y = y_temp

    elif dataset == "veterans":
        X, y = sksurv.datasets.load_veterans_lung_cancer()
        col_names = ["Celltype", "Prior_therapy", "Treatment"]
        for col_name in col_names:
            X = transform_one_hot(X, col_name)
        y_temp = pd.DataFrame(y, columns=["Survival_in_days", "Status"])
        di_event = {False: 0, True: 1}
        y_temp = y_temp.replace({"Status": di_event})
        y = y_temp
    elif dataset == "hnscc":
        raise ValueError("Dataset exists")
    else:
        raise ValueError("Dataset unknown")

    dump(X, f"{dataset}_x.csv")
```

```
    dump(y, f"{dataset}_y.csv")


def load_data(dataset):
    path_x = DATA_PATH / f"{dataset}_x.csv"
    path_y = DATA_PATH / f"{dataset}_y.csv"
    if not (path_x.exists() and path_y.exists()):
        gen_data(dataset)
    X = np.genfromtxt(path_x, delimiter=",", skip_header=1)
    y = np.genfromtxt(path_y, delimiter=",", skip_header=1)
    X = preprocessing.StandardScaler().fit(X).transform(X)
    return X, y


def run(
    dataset,
    hidden_dims,
    path_multiplier,
    *,
    random_state,
    tie_approximation="breslow",
    dump_splits=False,
    verbose=False,
):
    X, y = load_data(dataset)
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, random_state=random_state, stratify=y[:, 1], test_size=0.20
    )

    if dump_splits:
        for array, name in [
            (X_train, "x_train"),
            (y_train, "y_train"),
            (X_test, "x_test"),
            (y_test, "y_test"),
        ]:
            dump(array, f"{dataset}_{name}_{random_state}.csv")

    cv = list(
        StratifiedKFold(n_splits=5, shuffle=True, random_state=random_state).split(
            X_train, y_train[:, 1]
        )
    )

    model = LassoNetCoxRegressorCV(
        tie_approximation=tie_approximation,
        hidden_dims=hidden_dims,
        path_multiplier=path_multiplier,
        cv=cv,
        torch_seed=random_state,
        verbose=verbose,
    )
    model.path(X_train, y_train)
    plot_cv(model, X_test, y_test)
    plt.savefig(
        FIGURES_PATH
        / (
            f"cox-cv-{dataset}-{random_state}"
            f"-{model.hidden_dims}-{model.path_multiplier}.jpg"
```

```
        ),
        dpi=300,
    )

    test_score = model.score(X_test, y_test)

    return test_score


def experiment(dataset, hidden_dims, path_multiplier, n_runs, n_jobs):
    start = time()
    with tqdm_joblib(desc=f"Running on {dataset}", total=n_runs):
        scores = np.array(
            Parallel(n_jobs=n_jobs)(
                delayed(run)(
                    dataset,
                    hidden_dims=hidden_dims,
                    path_multiplier=path_multiplier,
                    tie_approximation=tie_approximation,
                    random_state=random_state,
                )
                for random_state in range(n_runs)
            )
        )
    scores_str = np.array2string(
        scores, separator=", ", formatter={"float_kind": "{:.2f}".format}
    )
    log = (
        f"Dataset: {dataset}\n"
        f"Arch: {hidden_dims}\n"
        f"Path multiplier: {path_multiplier}\n"
        f"Score: {scores.mean():.04f} "
        f"± {confidence_interval(scores):.04f} "
        f"with {n_runs} runs\n"
        f"Raw scores: {scores_str}\n"
        f"Running time: {time() - start:.00f}s with {n_jobs} processors\n"
        f"Running time per run per cpu: {(time() - start) * n_jobs / n_runs:.00f}s\n"
        f"{'-' * 50}\n"
    )

    tqdm.write(log)
    with open("cox_experiments.log", "a") as f:
        print(log, file=f)


if __name__ == "__main__":
    """
    run with python3 script.py dataset [method]

    dataset=all runs all experiments

    method can be "breslow" or "efron" (default "efron")
    """

    import sys

    dataset = sys.argv[1]
    tie_approximation = sys.argv[2] if len(sys.argv) > 2 else "efron"
    if dataset == "all":
```

```
        datasets = ["breast", "whas500", "veterans", "hnscc"]
        verbose = False
    else:
        datasets = [dataset]
        verbose = 1

    N_RUNS = 15
    N_JOBS = 5  # set to a divisor of ‘n_runs‘ for maximal efficiency

    for hidden_dims in [(16, 16), (32,), (32, 16), (64,)]:
        for path_multiplier in [1.01, 1.02]:
            for dataset in datasets:
                experiment(
                    dataset=dataset,
                    hidden_dims=hidden_dims,
                    path_multiplier=path_multiplier,
                    n_runs=N_RUNS,
                    n_jobs=N_JOBS,
                )

```

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] The codes are in supplemental material
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

References

[1] David R Cox. Regression models and life-tables. *Journal of the Royal Statistical Society: Series B (Methodological)*, 34(2):187–202, 1972.

[2] Norman E Breslow. Analysis of survival data under the proportional hazards model. *International Statistical Review/Revue Internationale de Statistique*, pages 45–57, 1975.

[3] Bradley Efron. The efficiency of cox's likelihood function for censored data. *Journal of the American statistical Association*, 72(359):557–565, 1977.

[4] Terry M Therneau and Patricia M Grambsch. The cox model. In *Modeling survival data: extending the Cox model*, pages 39–77. Springer, 2000.

[5] L Mariani, D Coradini, E Biganzoli, P Boracchi, E Marubini, S Pilotti, B Salvadori, R Silvestrini, U Veronesi, R Zucali, et al. Prognostic factors for metachronous contralateral breast cancer: a comparison of the linear cox regression model and its artificial neural network extension. *Breast cancer research and treatment*, 44(2):167–178, 1997.

[6] Anny Xiang, Pablo Lapuerta, Alex Ryutov, Jonathan Buckley, and Stanley Azen. Comparison of the performance of neural network methods and cox regression for censored survival data. *Computational statistics & data analysis*, 34(2):243–257, 2000.

[7] Daniel J Sargent. Comparison of artificial neural networks with other statistical approaches: results from medical data sets. *Cancer: Interdisciplinary International Journal of the American Cancer Society*, 91(S8):1636–1642, 2001.

[8] Jared L Katzman, Uri Shaham, Alexander Cloninger, Jonathan Bates, Tingting Jiang, and Yuval Kluger. Deepsurv: personalized treatment recommender system using a cox proportional hazards deep neural network. *BMC medical research methodology*, 18(1):1–12, 2018.

[9] Håvard Kvamme, Ørnulf Borgan, and Ida Scheel. Time-to-event prediction with neural networks and cox regression. *arXiv preprint arXiv:1907.00825*, 2019.

[10] Stephane Fotso et al. PySurvival: Open source package for survival analysis modeling, 2019–.

[11] Sebastian Pölsterl. scikit-survival: A library for time-to-event analysis built on top of scikit-learn. *J. Mach. Learn. Res.*, 21(212):1–6, 2020.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[13] Jadwiga Borucka. Methods of handling tied events in the cox proportional hazard model. *Studia Oeconomica Posnaniensia*, 2(2):91–106, 2014.

[14] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[15] Ismael Lemhadri, Feng Ruan, and Rob Tibshirani. Lassonet: Neural networks with feature sparsity. In *International Conference on Artificial Intelligence and Statistics*, pages 10–18. PMLR, 2021.

[16] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[17] Christine Desmedt, Fanny Piette, Sherene Loi, Yixin Wang, Françoise Lallemand, Benjamin Haibe-Kains, Giuseppe Viale, Mauro Delorenzi, Yi Zhang, Mahasti Saghatchian d'Assignies, et al. Strong time dependence of the 76-gene prognostic signature for node-negative breast cancer patients in the transbig multicenter independent validation series. *Clinical cancer research*, 13(11):3207–3214, 2007.

[18] Rainer Muche. Applied survival analysis: Regression modeling of time to event data. dw hosmer, jr., s lemeshow. new york: John wiley, 1999, pp. 386, us $89.95. isbn: 0-471-15410-5., 2001.

[19] John D Kalbfleisch and Ross L Prentice. *The statistical analysis of failure time data*. John Wiley & Sons, 2011.

[20] Aaron Grossberg, Hesham Elhalawani, Abdallah Mohamed, Sam Mulder, Bowman Williams, Aubrey L. White, James Zafereo, Andrew J. Wong, Joel E. Berends, Shady AboHashem, Jeremy M. Aymard, Aasheesh Kanwar, Subha Perni, Crosby D. Rock, Sasikarn Chamchod, Michael E. Kantor, Theodora Browne, Katherine A. Hutcheson, G. Brandon Gunn, Steven J. Frank, David Rosenthal, Adam S. Garden, Clifton Fuller, M.D. Anderson Cancer Center Head, and Neck Quantitative Imaging Working Group. Hnscc, 2020.

[21] Noah Simon, Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for cox's proportional hazards model via coordinate descent. *Journal of statistical software*, 39(5):1, 2011.

[22] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

[23] Annette Spooner, Emily Chen, Arcot Sowmya, Perminder Sachdev, Nicole A Kochan, Julian Trollor, and Henry Brodaty. A comparison of machine learning methods for survival analysis of high-dimensional clinical data for dementia prediction. *Scientific reports*, 10(1):1–10, 2020.

[24] William R Lane, Stephen W Looney, and James W Wansley. An application of the cox proportional hazards model to bank failure. *Journal of Banking & Finance*, 10(4):511–531, 1986.

[25] Kung-Yee Liang, Steven G Self, and Xinhua Liu. The cox proportional hazards model with change point: An epidemiologic application. *Biometrics*, pages 783–793, 1990.

[26] A Bendell, DW Wightman, and EV Walker. Applying proportional hazards modelling in reliability. *Reliability Engineering & System Safety*, 34(1):35–53, 1991.

[27] Bradley Efron. Logistic regression, survival analysis, and the kaplan-meier curve. *Journal of the American statistical Association*, 83(402):414–425, 1988.

[28] Hwa-Gan H Chang, Lloyd L Lininger, Joseph T Doyle, Patricia A Maccubbin, and Richard B Rothenberg. Application of the cox model as a predictor of relative risk of coronary heart disease in the albany study. *Statistics in Medicine*, 9(3):287–292, 1990.

[29] Cong Li and Jianguo Sun. Variable selection for high-dimensional quadratic cox model with application to alzheimer's disease. *The International Journal of Biostatistics*, 16(2), 2020.

[30] Philippe Bastien. Pls-cox model: application to gene expression. *Proceedings in Computational Statistics*, pages 655–662, 2004.

[31] Lynn A Sleeper and David P Harrington. Regression splines in the cox model with application to covariate effects in liver disease. *Journal of the American Statistical Association*, 85(412):941–949, 1990.

[32] Travers Ching, Xun Zhu, and Lana X Garmire. Cox-nnet: an artificial neural network method for prognosis prediction of high-throughput omics data. *PLoS computational biology*, 14(4):e1006076, 2018.

[33] Terry M Therneau. Extending the cox model. In *Proceedings of the First Seattle symposium in biostatistics*, pages 51–84. Springer, 1997.

[34] Frank E Harrell, Robert M Califf, David B Pryor, Kerry L Lee, and Robert A Rosati. Evaluating the yield of medical tests. *Jama*, 247(18):2543–2546, 1982.

[35] Hajime Uno, Tianxi Cai, Michael J Pencina, Ralph B D'Agostino, and Lee-Jen Wei. On the c-statistics for evaluating overall adequacy of risk prediction procedures with censored survival data. *Statistics in medicine*, 30(10):1105–1117, 2011.

[36] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[37] MD Anderson Cancer Center Head, Neck Quantitative Imaging Working Group, et al. Investigation of radiomic signatures for local recurrence using primary tumor texture analysis in oropharyngeal head and neck cancer patients. *Scientific reports*, 8, 2018.

[38] Sebastian Pölsterl. scikit-survival: A library for time-to-event analysis built on top of scikit-learn. *Journal of Machine Learning Research*, 21(212):1–6, 2020.

[39] Kim Sejin. Med-imagetools:transparent and reproducible medical image processing pipelines in python. `https://github.com/bhklab/med-imagetools`, 2022.

[40] Joost JM Van Griethuysen, Andriy Fedorov, Chintan Parmar, Ahmed Hosny, Nicole Aucoin, Vivek Narayan, Regina GH Beets-Tan, Jean-Christophe Fillion-Robin, Steve Pieper, and Hugo JWL Aerts. Computational radiomics system to decode the radiographic phenotype. *Cancer research*, 77(21):e104–e107, 2017.