High Dimensional Feature Selection Algorithm with Interactions on Time-to-Event Outcome

by

Lin Yu

Supervisor

Dr. Wei Xu

Dalla Lana School of Public Health

University of Toronto

Toronto, Ontario, Canada

Abstract (word count: 250)

Background: Cancer survival analysis often confronts high-dimensional data. To allow the implementation of classical statistical modelling such as Cox regression in such setting, feature selection (FS) is commonly conducted before statistical modelling. However, existing algorithms do not often consider the interaction term and ignore the statistical significance of it, even though the interactive effect is of clinical interest in helping understand the complex relationships among variables and reveal potential heterogenous effect among individuals.

Objectives: This study aimed to develop high-dimensional feature selection with interactions (HDSI) algorithms for time-to-event outcome that incorporates interactions in the high dimensional setting.

Methods: Two algorithms, HDSI-LASSO and HDSI-Ridge were developed by incorporating Cox-LASSO and Cox-Ridge into the HDSI framework. Bootstrap resampling algorithm was developed by incorporating the two-way interaction terms into the model, FS criteria were determined based on pooled significance and model performance of each feature. The performances of the proposed algorithm were evaluated in simulation studies and implemented into a real-world dataset.

Results: In the simulations, the HDSI-LASSO and HDSI-Ridge, on average, selected 3 out of 5 true marginal effects, and 1 out of 2 true interactive features. In the real-world study, HDSI-LASSO and HDSI-Ridge selected 7 and 66 out of 4950 interaction terms, respectively. Both algorithms showed good performance by selecting very few noisy features.

Conclusions: The proposed HDSI algorithms show potential in identifying interactive effects, which consequently, increase the model performance. The HDSI-LASSO offers more robust performance regarding changes in the number of features, while HDSI-Ridge is preferable for lower dimension settings.

1. Introduction

With the advent of the big data era, high-dimensional feature become more and more common in cancer survival analysis. The literature has seen successful applications of large-scale datasets in prediction of disease outcomes(X. Wang & Wen, 2022)(Bazgir & Lu, 2023). On the other hand, it brings several challenges for developing prediction models using classical survival regression models given the data complexity and limited sample size of cancer patients. For example, the classical survival model is usually infeasible to fit, or poorly fit as most statistical methods were developed under the assumption that the number of features is much smaller than the number of study samples. Additionally, it is well-known that high-dimensional features could cause overfitting issues(Pires & Branco, 2019). Finally, the modelling process for high-dimensional data usually is computationally intensive and inefficient.

To address the problems of high dimensionality, feature selection (FS) is usually performed before statistical modelling. FS is a process that aims to select a subset of features from the original dataset that are important for outcome prediction(Walter & Tiemeier, 2009). Several FS methods exist in the literature and can be classified into filter (e.g., correlation coefficient), wrapper (e.g., random forest, support vector machine, etc.), and embedded (e.g., Lasso, Ridge, Elastic net etc.) methods(Theng & Bhoyar, 2023). However, these FS methods usually only consider the marginal effects, and less attention is paid to the interactive effects which are clinically important as they allow us to study how the effects differ among subpopulations with different clinical characteristics, and thus could provide valuable evidence for personalized prediction. Moreover, the performance measurement used for FS in the existing methods is less interpretable in public health domains. In most cases, the model performance is usually the only criterion for FS, and the significance of features is not considered (Jain & Xu, 2021).

Therefore, to account for interactive effects and the statistical interpretation, this study proposed a high-dimensional feature selection algorithm for time-to-event data. The proposed algorithm considered the interactive effects and statistical significance in the feature selection techniques for time-to-event outcomes. The benefit of examining interactive terms is two-fold: first, it allows us to examine the complex relationships among the features. Second, it could potentially improve the overall model performance on outcome prediction.

This report is structured as follows. In *Section 2*, we introduce the high dimensional feature selection algorithm with interactions on time-to-event outcome; subsequently, In *Section 3*, we assess the proposed algorithm in a simulation study. In *Section 4,* We implement the proposed algorithm to a real-world dataset and present the results. Finally, in *section 5*, we discuss the findings of this study.

2. Method

2.1 HDSI algorithms for continuous and binary outcome

Before we introduce the proposed algorithm, we briefly introduce the HDSI algorithm for continuous outcome proposed by (Jain & Xu, 2021) and HDSI-BO algorithm for binary outcome proposed by (Zhuang & Xu, 2021).

HDSI algorithms for continuous outcome was motivated by the random LASSO algorithm (S. Wang et al., 2011). The features are selected based on the statistical significance and its contribution to the model performance. Specifically, multiple samples of the same size as the original dataset are generated using bootstrap sampling with replacement. For each sample, features are randomly selected using bootstrap sampling without replacement. Then different modelling techniques (for example, linear regression) are applied to the bootstrap samples to obtain the coefficient estimates and model coefficient of determination (R^2). Then, the significance of each feature was determined by the quantile of the coefficient estimate. Finally, the feature selection is done based on the statistical significance and model performance. The

HDSI algorithm for binary case has a similar structure except that it uses the area under the ROC curve (AUC) as the model performance metric.

However, given the nature of time-to-event outcome, the HDSI algorithms for continuous and binary cases cannot be directly applied to time-to-event outcome. Therefore, the following sections described the main methodology that extends the HDSI algorithm to time-to-event outcome.

2.2 HDSI algorithm for time-to-event outcome

The following steps described the detailed HDSI algorithm on time-to-event outcome. The pipeline of the overall algorithm is summarized in **Box 1**.

Step 1: Prepare B bootstrap samples with interactive effects. Suppose the original data set D ($D \in n \times p$) has n observations, and p features denoted by $X_1, X_2, \dots, X_p$. We randomly sampled *n* observations with replacement, and sampled *k* (*k* < *p*) features without replacement from *D* to form a bootstrap sample B_j ($j \in 1,2,3 \dots, B$). For B_j, we further added all possible two-way interaction terms, which gives B_j of $\left(k + \binom{k}{2}\right)$ total features, with k main effects and $\binom{k}{2}$ interactive effects. This process was repeated for *B* times, generating *B* bootstrap samples with interactive effects.

Step 2: Build Cox regression model on B bootstrap samples. For B_j, a Cox regression model was fitted by regressing the survival outcome on both main and interactive features in the bootstrap sample. The estimated coefficient for feature X_i ,denoted by $\hat{\beta}_i^j$, was obtained from the model, and the model performance C_i^j, quantified by C-index, was also obtained. To summarize the model output from *B* bootstrap samples, the pooled estimated coefficient for feature X_i is calculated by:

$$\hat{\beta}_i = \sum_{j=1}^{b_i} \hat{\beta}_i^j / b_i \qquad (1)$$

where b_i is the total number of bootstrap samples that contains X_i.

The C-index for X_i is defined as the minimum of all C-index values of models that contain X_i, that is,

$$C_i = \min\,(C_i^1, C_i^2, C_i^3, \dots, C_i^{b_i}) \tag{2}$$

Step 3: Select features based on the coefficient estimates and C-index. Two aspects, namely the statistical significance and model performance, were considered in feature selection. The pooled coefficient estimates, and C-indexes were used as the criteria for feature selection, where the coefficient estimate of X_i captures its significance, and the C-index captures $X_i's$ contribution to the model performance.

Firstly, the lower quantile $\frac{Q_i}{2}$ and upper quantile (100-$\frac{Q_i}{2}$) percentile of feature X_i were computed from $\hat{\beta}_i^1, \hat{\beta}_i^2, \dots, \hat{\beta}_i^j, \hat{\beta}_i^{b_i}$ to determine $X_i's$ statistical significance. A quantile range $\left(\frac{Q_i}{2}, 100 - \frac{Q_i}{2}\right)$ include zero was considered as statistically significant, otherwise, X_i is not of statistical significance. This can be formulated as follows:

$$P_i = I\{0 \notin \left(\frac{Q_i}{2}, 100 - \frac{Q_i}{2}\right)\} \tag{3}$$

Where I is an indicator function, $P_i = 1$ and $P_i = 0$ indicate X_i is statistically significant and not significant, respectively.

Secondly, a C-index threshold $C_0$was calculated as follows:

$$C_0 = \mu_c + R_f \sigma \tag{4}$$

where μ_c is the average value of k+$\binom{k}{2}$ features, and can be calculated as:

$$\mu_c = \sum_{i=1}^{k+\binom{k}{2}} C_i \tag{5}$$

σ is the standard deviation of $C_1, C_2, \dots, C_p$, and R_f is a hype-parameter that takes values in the real line. A larger R_f indicates a higher threshold for feature selection, and features with C-indexes larger than the threshold were considered as candidate features.

$$CF_i = I\{C_i \geq C_0\} \tag{6}$$

Where I is an indicator function, $CF_i = 1$ if C_i is above the threshold; otherwise, $CF_i = 0$.

In combination of these two criteria, features were only selected when both P_i and CF_i equal to 1.

Box 1 A summary of the HDSI algorithm for time-to-event data

1. **Prepare B bootstrap samples with size n:**

   1(a) For sample B_j ($j \in (1,2, \dots B)$), use bootstrap with replacement to sample n observations, and bootstrap without replacement to sample k ($k < p$) features from the original dataset ($D \in n \times p$)

   1(b) For sample B_j, add all the possible two-way interaction terms from the k features, the total number of features in B_j is $k+\binom{k}{2}$.

2. **Build the model:**

   2(a) For the j^{th} bootstrap set, apply feature selection algorithms (e.g., lasso/ridge) to fit a Cox regression model, the estimated coefficients and C-index are obtained from the model output.

   2(b) Repeat for all B bootstrap sets and pool the results from B bootstrap sets. The pooled estimated coefficient is calculated by **Equation (1)**.

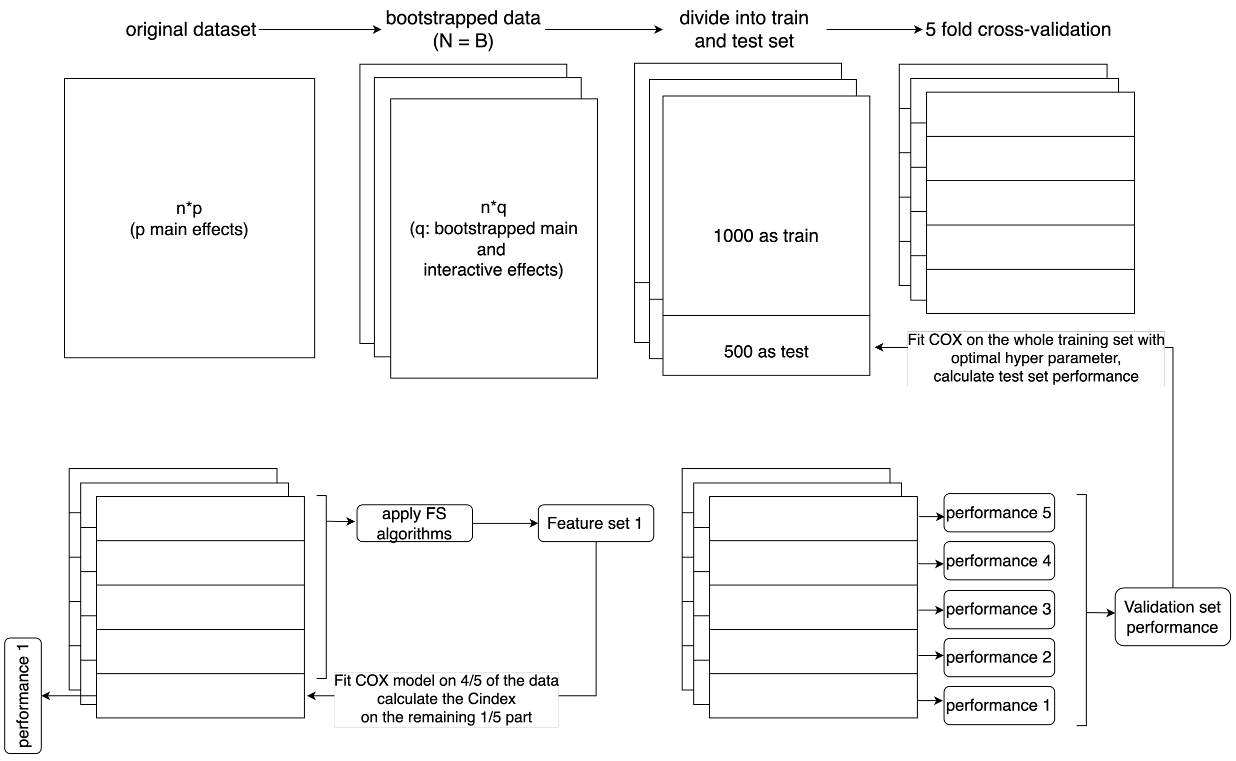
3. **Select Features**

   3(a) For feature X_i , the $\frac{Q_i}{2}$ quantile and (100- $\frac{Q_i}{2}$) quantile of its estimated effect are obtained.

3(b) Feature is selected if the quantile ($\frac{Q_i}{2}$, 100- $\frac{Q_i}{2}$) does not include zero and C-index is greater than the C-index threshold calculated by **Equation (4)** and **Equation (5)**.

2.3 Hyper-parameter Tuning

We considered two hyper-parameters: Q_i , the quantile of coefficient estimate, it takes values from .90 with a step size of 0.1 and R_f range from -4 to -4 with a step size of 1. The hyper-parameter tuning process is summarized in **Figure 1.**



original dataset
bootstrapped data (N = B)
divide into train and test set
5 fold cross-validation
n*p (p main effects)
n*q (q: bootstrapped main and interactive effects)
1000 as train
500 as test
Fit COX on the whole training set with optimal hyper parameter, calculate test set performance
apply FS algorithms
Feature set 1
Fit COX model on 4/5 of the data calculate the Cindex on the remaining 1/5 part
performance 1
performance 5
performance 4
performance 3
performance 2
performance 1
Validation set performance

Figure 1 The workflow for the hyperparameter tuning process.

B bootstrapped sets were sampled using Step 1 we mentioned earlier. The K-fold cross-validation (K =5) was used. The training data was randomly divided into five roughly equal-sized parts. For each combination of hyper-parameters, each time one part of the data was left out as the validation set. The remaining four parts were combined as the training set. The HDSI feature selection algorithm was then applied to the four parts that form the training set to fit a Cox regression model, and the C-index was calculated by applying the model fitted to the validation set to get the validation performance. This process was done recursively on each part of the training set until each part was used as the validation set. The validation performance of the model based on the current hyper-parameters was calculated as the average of the internal validation performance. The pseudo codes for this process are illustrated in **Box 2**.

Box 2 Pseudo codes for hyper-parameter tuning

```
## Loop over quantile
for{qtl in seq(0,0.5,0.01)}{
  ## Loop over Rf
  for{Rf in seq(-4,4,0.5)}{
    ## loop over B bootstrap samples
    for (B in c(1:B)) {
      divide B into 5 parts
      for (K in c(1:5)){
        apply HDSI algorithm on Data[-K,]
        cv_performance[K] = calculate the internal performance on Data[K,]
      }
      ## calculate averaged internal performance for the $B^{th}$ bootstrap sample
      internal_perforamnce[B] = mean(performance [1:5])
    }
    ## combine the averaged internal performance of B bootstrap samples to
    ## get the performance under the current value of hyperparameters
    performance[qtl,Rf] = bind_rows(internal_perforamnce[1],...,internal_perforamnce[B])

  }

}
```

The analysis was done with R (version 4.3.1). The GitHub repository for this work can be accessed at: thisislinyu/CHL5208: practicum project (github.com).

3 Simulation study

3.1 simulation design

In the simulation, we assumed there are 5 true continuous features namely X_1, X_2, X_3, X_4 and X_5, along with two interactive terms X_1X_2 and X_3X_4 are associated with the outcome, the true model thus can be specified as:

$$h(t) = h_0(t)\exp(Z) \tag{7}$$

where the linear predictor is:

$$Z = X_1 + X_2 + 0.75X_3 - 0.75X_4 + 0.75X_5 + X_1X_2 - X_3X_4 \tag{8}$$

The survival time $\tilde{T}$ was generated using inverse transform sampling method(Bender et al., 2005). The detailed derivation can be found in **Appendix 1**. Briefly, in the Cox regression model, the survival function can be written as follows:

$$S(\tilde{T}) = \exp[-H_0(\tilde{T})\exp(Z)] \tag{9}$$

Where $S(\tilde{T})$ follows a uniform (0,1) distribution. Therefore, the survival time $\tilde{T}$ can be expressed as

$$\tilde{T} = {H_0}^{-1}(-\log(U)\exp(-Z)) \tag{10}$$

where U is a random variable with uniform (0,1) distribution. ${H_0}^{-1}(.)$ is the inverse of cumulative hazard function. Furthermore, we assumed survival time follows exp (λ) distribution, thus gives:

$$ {H_0}^{-1}(t) = \lambda^{-1}t \tag{11} $$

Therefore, the survival time was generated from:

$$ \tilde{T} = \lambda^{-1}(-\log(U)\exp(-(X_1 + X_2 + 0.75X_3 - 0.75X_4 + 0.75X_5 + X_1X_2 - X_3X_4)))(12) $$

Where X_1, X_2 , X_3, X_4 and X_5 were generated from a multivariate normal distribution with some correlation (correlation matrix is provided in **Appendix Table 1**).

The censoring time C was assumed to follow a uniform distribution:

$$ C \sim \mathrm{Unif}(0, b) \tag{12} $$

where b was varied to achieve a specific event rate. The observed event time $\boldsymbol{T}$ is either the survival time ($\tilde{T}$) or censoring time (C), depending on whichever comes first. That is,

$$ T = min\,(\tilde{T}, C) \tag{13} $$

The outcome status (Y) was determined correspondingly:

$$ Y = \begin{cases} 1, & T = \tilde{T} \\ 0, & T = C \end{cases} \tag{14} $$

3.2 Simulation Result

We simulated a training set with 1000 subjects for internal cross-validation and a test set with 500 subjects for external validation. In each of the datasets, we generated 25 features, and the event rates in training and test sets are roughly 40%. The internal and external FS performance of HDSI-LASSO and HDSI-Ridge is presented in **Table 1.**

There are two findings from the simulation results. First, both algorithms did not select noisy marginal features, and HDSI-LASSO algorithm selected less features than HDSI-Ridge in different simulation settings. Second, HDSI-LASSO is more robust to the increase in the number of features for both marginal and interactive effects, while HDSI-Ridge is only robust to the increase in the number of marginal features, and it selected more noisy interactive features as the number of features considered increases.

Table 1 Feature selection performance of HDSI-LASSO and HDSI-Ridge on simulated datasets.

	Internal Validation				**External Validation**			
	true marginal (5)	noisy marginal (15)	true interaction (2)	noisy interaction (198)	true marginal (5)	noisy marginal (15)	true interaction (2)	noisy interaction (198)
HDSI-LASSO	4	0	2	0	2	0	1	0
	4	0	2	0	3	0	1	0
	3	0	1	0	2	0	1	0
	3	0	1	1	5	0	2	0
	3	0	1	0	3	0	1	0
HDSI-Ridge	2	0	1	0	3	0	1	1
	4	0	1	1	3	0	1	1
	3	0	1	2	3	0	1	2
	4	0	1	1	3	0	1	1
	4	1	1	2	3	0	1	1

4. Application to Gene Expression Dataset

4.1 The TCGA lung cancer (LUNG) cohort

We implemented the proposed algorithm into the TCGA lung cancer (LUNG) cohort collected by the Cancer Genome Atlas program(*The Cancer Genome Atlas Program (TCGA) - NCI*, 2022). The LUNG cohort contains the gene expression profile data of lung cancer patients with either lung adenocarcinoma (LUAD) or lung squamous cell carcinoma (LUSC)(Ellrott et al., 2018). We extracted gene expression and the overall survival time and status data. Meanwhile, the demographic characteristics including age, gender, and cancer stage were extracted from the Genetic Data Commons Data Portal.

The univariate Cox regression analyses were conducted on each of the genes, all *P*-values were adjusted using the Benjamini & Hochberg method (details are provided in the **Appendix Figures 1**). The data was split into training and test with a ratio of 8:2, the HDSI algorithm was implemented on the training and test sets with statistically significant baseline covariate(s) which was/were identified from the univariate Cox regressions controlled.

4.2 Real-world Data Study

The algorithm was applied to the LUNG dataset. The inclusion and exclusion of subjects is presented in **Appendix Figures 2**. In total, 996 lung cancer patients with complete gene expression, outcome, and clinical information were considered.

Table 2 shows the summary statistics of the demographic information of the working dataset. 597 (60%) subjects are males and 399 (40%) are females. The median age at lung cancer diagnosis in the cohort is 69.6 years [IQR: 62.1, 75.4]. 394 (40%) died before the end of the follow-up period. The overall survival (OS) is presented in **Figure 2**. The median OS time is 1.94 years, and the 3-year and 5-year survival probabilities are 0.606 (95% CI: 0.569, 0.645) and 0.435 (95% CI: 0.391, 0.484), respectively.

Table 2 Demographic characteristics of patients in the LUAD and LUSC cohort

	Full Sample[1] (N=996)	**LUAD[2] (N=502)**	**LUSC[3] (N=494)**
Survival status			
Alive	602 (60)	320 (64)	282 (57)
Dead	394 (40)	182 (36)	212 (43)
OS[4] time (month)			
Mean (sd)	33.8 (33.0)	32.6 (32.8)	35.1 (33.2)
Median (Q1,Q3)	23.3 (14.5, 41.0)	22.3 (15.6, 37.8)	25.2 (13.2, 46.0)
Range (min, max)	(0.1, 241.6)	(0.4, 241.6)	(0.1, 158.8)
Age at diagnosis (yrs)			
Mean (sd)	68.5 (9.6)	67.4 (10.2)	69.5 (8.8)
Median (Q1,Q3)	69.6 (62.1, 75.4)	68.5 (61.0, 74.8)	70.4 (64.0, 75.7)
Range (min, max)	(39.5, 92.3)	(39.5, 91.1)	(40.3, 92.3)
Missing	27	22	5
Gender			
Female	399 (40)	271 (54)	128 (26)
Male	597 (60)	231 (46)	366 (74)
Race			
White	735 (88)	387 (86)	348 (90)
Other	99 (12)	61 (14)	38 (10)
Missing	162	54	108
Cancer stage[5]			
Early stage	789 (80)	389 (79)	400 (82)

Late stage	195 (20)	105 (21)	90 (18)
Missing	12	8	4

[1] Full sample refers to the combination of lung adenocarcinoma and lung squamous cell carcinoma data; [2] lung adenocarcinoma; [3] lung squamous cell carcinoma; [4] Overall survival; [5] Cancer stage is dichotomized into early stage (stage I and II) and late stage (stage III and IV).

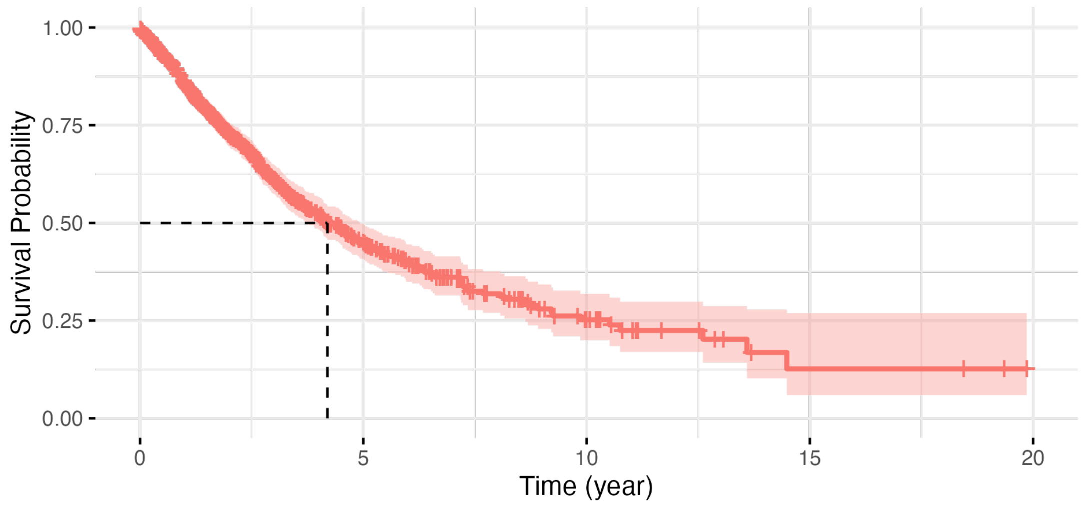


Figure 2 Overall survival of lung cancer patients over time

Table 3 shows the univariate Cox regression results for each of the demographic baseline covariates. The Age at diagnosis and cancer stage are of statistical significance with hazard ratios of 1.12 (95% CI: [1.00, 1.25], *P*-value: 0.04) and 1.98 (95%CI: [1.59, 2.47], *P*-value: <0.0001), respectively. The remaining baseline covariates are not statistically significant and were not considered in the following HDSI algorithm.

Table 3 Results of univariate Cox regression data analysis for demographic characteristics

Variables[2]	**HR[2] (95%CI)**	***P*-value**
Age at diagnosis (yrs)	1.12 (1.00, 1.25)	**0.04**

Gender	1.15 (0.93, 1.41)	0.20
Race	1.04 (0.75, 1.44)	0.82
Cancer type[3]	1.11 (0.91, 1.36)	0.30
Cancer stage	1.98 (1.59, 2.47)	**<0.01**

[1] *reference group for each categorical variable: gender= famle, race= other, caner type = LUSC, cancer stage: late stage;* [2] *HR: hazard (rate) ratio;* [3] *LUAD: lung adenocarcinoma, LUSC: lung squamous cell carcinoma*;

The univariate Cox regression analyses were employed for each of the 20,531 genes. We considered two scenarios, where in the first scenario, the top fifty significant genes from the univariate Cox regression were considered, and further the top one hundred significant genes were selected as the input features of the HDSI-LASSO/ HDSI-Ridge algorithm. The FS performance of different HDSI algorithms is summarized in **Table 4**. There are three observations from the results: First, the HDSI-LASSO algorithm is robust to the number of features as indicated by comparing the number of marginal and interactive terms being selected in having 50 genes or 100 genes in the input feature space. The number of selected marginal and interactive terms are comparable. Conversely, the HDSI-Ridge algorithm tends to select more interactive terms with the increase of the number of input features, and the marginal effect is comparable, implying HDSI-Ridge may select more noisy features and could potentially lead to overfitting when the feature space is high. Second, the model performance (the C-index column) shows that HDSI-Ridge outperforms HDSI-LASSO in both scenarios as HDSI-Ridge selected more features. Third, regarding the incremental value in the model performance, we see the HDSI-LASSO selected less marginal and interactive features than HDSI-Ridge in both scenarios, along with slightly lower performance.

Table 4 FS performance and model performance on the LUNG dataset in scenario 1

	Marginal (50)	Interactive (1225)	Main effect C-index	Interactive C-index	Incremental value
HDSI-LASSO	23	14	0.580 (0.018)	0.657 (0.019)	0.077
	8	4	0.630 (0.019)	0.635 (0.019)	0.005
	9	5	0.641 (0.019)	0.648 (0.018)	0.007
	19	13	0.640 (0.019)	0.641 (0.019)	0.001
	9	6	0.648 (0.018)	0.644 (0.018)	-0.004
HDSI-Ridge	32	48	0.648 (0.018)	0.705 (0.016)	0.057
	36	59	0.662 (0.019)	0.683 (0.018)	0.021
	30	44	0.655 (0.018)	0.689 (0.018)	0.034

Table 5 FS performance and model performance on the LUNG dataset in scenario 2

	marginal (100)	Interactive (4950)	Main effect C-index	Interactive C-index	Incremental value
HDSI-LASSO	19	12	0.643 (0.018)	0.649 (0.018)	0.006
	9	5	0.638 (0.018)	0.640 (0.018)	0.002
	9	7	0.639 (0.018)	0.648 (0.018)	0.009
	7	4	0.638 (0.018)	0.640(0.018)	0.002
	10	6	0.637 (0.018)	0.639 (0.018)	0.002
	24	61	0.623 (0.018)	0.672 (0.018)	0.049

HDSI-Ridge	27	72	0.630 (0.019)	0.648 (0.018)	0.018
	33	64	0.641 (0.018)	0.700 (0.016)	0.059
	34	66	0.642 (0.018)	0.700 (0.016)	0.058
	28	69	0.648 (0.018)	0.685 (0.017)	0.037

5. Discussion

This study introduced the HDSI algorithms for time-to-event outcome that accounts for interactive effects and statistical significance in high-dimensional settings. The HDSI algorithms provided a flexible framework that allows incorporating existing prediction algorithms, along with adding interaction terms in the FS process.

The simulation and real-world study results showed that the HDSI-LASSO selected less true features compared to the HDSI-Ridge method, thus generating slightly lower model performance. However, the number of noise features being selected by the HDSI-Ridge increased with the increase of the number of input features. Therefore, when applying the proposed algorithm, it is recommended to consider HDSI-LASSO when the feature space is high.

This study has a few limitations. One limitation of this study is that this study has not integrate other algorithms in the HDSI framework. We incorporated Cox-LASSO and Cox-Ridge algorithms in the framework. Given the different behaviors that Cox-LASSO and Cox-Ridge performed in different scenarios (different number of input features) as we discussed in the last paragraph, it is worth investigating how the algorithm will behave when simultaneously adding LASSO and Ridge (i.e., Elastic Net) into the HDSI framework. It could potentially take advantage of both algorithms, and consequently, improve the FS performance and model performance. Another limitation of this study is that the robustness of the proposed algorithm needs to be further confirmed before application. In the simulation, we started with 25 features (300 interaction terms) and effect sizes were set to be either 1.00 [HR =exp (1.00) $\approx$2.7] or 0.75 [HR =exp (0.75) $\approx$2.1]. The number of bootstraps was fixed to be 100. Moving forward, the number of bootstraps could be increased to make sure each feature could have the probability of being selected for examination. The FS performance and model performance under the HDSI framework could be examined under various settings including increased number of features, different effect sizes to confirm the robustness of this algorithm.

Acknowledgement

I would like to express my gratitude and appreciation to everyone who has supported me through the process of the practicum. In particular, I would like to give a huge thank you to Dr. Wei Xu, my supervisor who has always been there to guide me throughout the entire project. I would like to thank him for his continuous support and assistance. I would not have made it this far without him!

I would like to thank Professor Tony Panzarella for this great course, which offers me the opportunity to learn and grow. I would also like to thank the professors and staff who have contributed to the practicum course.

I would like to thank members in Dr. Wei Xu's research group for their generous help. I want to acknowledge and thank Ziqian Zhuang and Jasper Zhang, in particular, for their advice and for taking the time to help me.

References

Bazgir, O., & Lu, J. (2023). REFINED-CNN framework for survival prediction with high-dimensional features. *iScience*, *26*(9), 107627. https://doi.org/10.1016/j.isci.2023.107627

Bender, R., Augustin, T., & Blettner, M. (2005). Generating survival times to simulate Cox proportional hazards models. *Statistics in Medicine*, *24*(11), 1713–1723. https://doi.org/10.1002/sim.2059

Ellrott, K., Bailey, M. H., Saksena, G., Covington, K. R., Kandoth, C., Stewart, C., Hess, J., Ma, S., Chiotti, K. E., McLellan, M., Sofia, H. J., Hutter, C., Getz, G., Wheeler, D., Ding, L., Caesar-Johnson, S. J., Demchok, J. A., Felau, I., Kasapi, M., … Mariamidze, A. (2018). Scalable Open Science Approach for Mutation Calling of Tumor Exomes Using Multiple Genomic Pipelines. *Cell Systems*, *6*(3), 271-281.e7. https://doi.org/10.1016/j.cels.2018.03.002

Jain, R., & Xu, W. (n.d.). HDSI: High dimensional selection with interactions algorithm on feature selection and testing. *PLOS ONE*.

Pires, A. M., & Branco, J. A. (2019). *High dimensionality: The latest challenge to data analysis* (arXiv:1902.04679). arXiv. http://arxiv.org/abs/1902.04679

The Cancer Genome Atlas Program (TCGA)—NCI (nciglobal,ncienterprise). (2022, May 13). [cgvMiniLanding]. https://www.cancer.gov/ccg/research/genome-sequencing/tcga

Theng, D., & Bhoyar, K. K. (2023). Feature selection techniques for machine learning: A survey of more than two decades of research. *Knowledge and Information Systems*. https://doi.org/10.1007/s10115-023-02010-5

Walter, S., & Tiemeier, H. (2009). Variable selection: Current practice in epidemiological studies. *European Journal of Epidemiology*, *24*(12), 733–736. https://doi.org/10.1007/s10654-009-9411-2

Wang, S., Nan, B., Rosset, S., & Zhu, J. (2011). RANDOM LASSO. *The Annals of Applied Statistics*, *5*(1), 468–485. https://doi.org/10.1214/10-AOAS377

Wang, X., & Wen, Y. (2022). A penalized linear mixed model with generalized method of moments for prediction analysis on high-dimensional multi-omics data. *Briefings in Bioinformatics*, *23*(4), bbac193. https://doi.org/10.1093/bib/bbac193

Zhuang, Z., Xu, W., & Jain, R. (n.d.). *High Dimensional Selection with Interactions Algorithm on Feature Selection for Binary Outcome*.

Appendix

Appendix Codes

Main function calls and codes are presented here, the complete codes for this project are archived at : CHL5208/R at master · thisislinyu/CHL5208 (github.com)

```
#' Simulate a survival data set
#'
#' @param nsample total number of observations
#' @param varnum  total number of covariates
#' @param dist distribution of baseline hazard function
#' @param lambda scale parameter of the distribution
#' @param rho shape parameter of the distribution
#' @param beta pre-specified effect size of covariates
#' @param crate rate parameter of the exponential distribution
#' @param cor logical; whether consider the correlation between
covariates
#' @param seed
#'
#' @return a dataframe; time, status, covariates
#' @export
#' @examples
#'library(survival)
#'dat <- sim_survdat_f(nsample =1000, varnum = 25,dist='g',lambda=0.01,
rho=1, beta=c(2.3,0.3,0.4), crate=0.001,cor=TRUE,seed=20231106)
#'fit <- survival::coxph(Surv(time, status) ~ X1+X3+X1*X3, data=dat)


sim_survdat_f <- function(nsample = 100,
                          varnum =25,
                          dist = 'w',
                          lambda = 0.01,
                          rho = 1,
                          beta=NA,
                          crate = 0.001,
                          cor=TRUE,
                          seed = 20231106,
                          ...)
{

  library(tidyverse)

  Sigma=matrix(rep(0,varnum), nrow=varnum, ncol=varnum, byrow=F)

  for(i in 1:varnum){Sigma[i,i]=10}
  # Correlation Settings
  if(cor){
    Sigma[1,2]=3;Sigma[1,3]=3;Sigma[1,4]=6;Sigma[1,5]=6
    Sigma[2,1]=3;Sigma[3,1]=3;Sigma[4,1]=6;Sigma[5,1]=6
    Sigma[2,3]=3;Sigma[2,4]=2;Sigma[2,5]=1
    Sigma[3,2]=3;Sigma[4,2]=2;Sigma[5,2]=1
    Sigma[3,4]=2;Sigma[3,5]=1
    Sigma[4,3]=2;Sigma[5,3]=1
    Sigma[4,5]=1
    Sigma[5,4]=1
  }

  set.seed(seed)
  covar_df <-  data.frame(MASS::mvrnorm(n = nsample, rep(0, varnum),
Sigma/10))


  beta_df <- matrix(rep(beta,each = nsample),ncol = length(beta)) %>%
data.frame()


  modelvar_df <- covar_df %>% select(X1,X2,X3,X4,X5) %>% mutate(X1_X2 =
X1*X2,X3_X4=X3*X4)
  Z <- apply((beta_df*modelvar_df), 1,sum) %>% unlist()## element-wise
product of two data frames


  # X1 <- covar_df$X1
  # X3 <- covar_df$X3
  # Z = beta[1]*X1+beta[2]*X3+beta[3]*X1*X3


  U <- runif(n=nsample)


  if(dist == 'e'){
    time <- (- log(U) / (lambda * exp(Z)))
  }


  if(dist == 'w'){
    time <- (- log(U) / (lambda * exp(Z)))^(1 / rho)
  }


  if(dist =='g'){
    nd_term = ( (rho * log(U)) / (lambda * exp(Z)))
    time <-(1/rho) * log(1- nd_term)
  }
  # censoring times
  C <- rexp(n=nsample, rate=crate)
  C <- runif(nsample,0,quantile(time,0.30))
  C <- runif(nsample,0,quantile(time,0.9))
  print(paste(
    'event rate is',
    sum(time <= C )/nsample
  )
  )
  # follow-up times and event indicators
  status <- as.numeric(time <= C)
  time <- pmin(time, C) ## over-write time variable, compare between
time and censor time


  # data set
  data.frame(time=time,
             status=status,
             covar_df)
}
#
# set.seed(1234)
# betaHat <- rep(NA, 1e3)
# for(k in 1:1e3)
# {
#   dat <- sim_survdat_f(varnum = 25,nsample =1000, lambda=0.01, rho=1,
beta=c(2.3,0.3,0.4), crate=0.001,cor=TRUE,seed=20231106)
#   fit <- coxph(Surv(time, status) ~ X1+X3+X1*X3, data=dat)
#   betaHat[k] <- fit$coef[[2]]
# }
# mean(betaHat)
```

```
#dat <- sim_survdat_f(nsample =1000, varnum = 25,dist='w',lambda=0.01,
rho=1, beta=c(0.1,0.2,0.3), crate=0.001,cor=TRUE,seed=20231106)
set.seed(100)
## censoring rate
## event rate


#' Generating bootstrap sample datasets
#'
#' @param df working dataset,simulated or real data collected
#' @param nboot number of bootstrap datasets
#' @param boot_ft number of features in each bootstrap dataset
#' @param seed random seed
#'
#' @return a list, size = nboot, each element is a dataframe
#' @export
#'
#' @examples
#'library(survival)
#'dat <- sim_survdat_f(nsample =1000, varnum = 25,dist='g',lambda=0.01,
rho=1, beta=c(2.3,0.3,0.4), crate=0.001,cor=TRUE,seed=20231106)
#'fit <- survival::coxph(Surv(time, status) ~ X1+X3+X1*X3, data=dat)
#' boot_lst <- boot_f(df = ,nboot = 100,boot_ft = 5,seed = 20231106)


boot_f <- function(df = NA,
                   nboot = 100,
                   boot_ft = 5,
                   seed = 20231106) {
  ## get the feature names in df total features
  tot_ft <- df %>% select(matches("^X\\d+$")) %>%
    # starts_with("X") # ^ start sign;
    # $ end of a string, d+ one or more digital
    ## This may be problematic if df is not generated from sim_surv_f
    colnames()

  set.seed(seed)

  boot_lst <- lapply(1:nboot, function(x) {
    ### sample boot_ft from tot_ft
    boot_feature <- sample(tot_ft, boot_ft, replace = FALSE) %>%
      naturalsort::naturalsort()

    ### sample nrow(df) people from df with replacement;

    boot_sample <- df %>%
      #### sample status = 1
      filter(status == 1) %>%
      sample_n(size = nrow(.), replace = TRUE) %>%
      bind_rows(df %>% filter(status == 0) %>%
        #### sample status = 0
        sample_n(size = nrow(.), replace = TRUE)) %>%
      select(time, status, all_of(boot_feature))
  })

  return(boot_lst)
}
```

## boot_f1 for real data

```
# > colnames(reg_dat) %>% head()
# [1] "time"
# [2] "status"
# [3] "age_at_diagnosis"

# [4] "cancer_stage"

# [5] "X_CIDEC"

# [6] "X_LOC441869"


boot_f <- function(df = NA,

                   nboot = 100,

                   boot_ft = 5,

                   seed = 20231106) {

  ## get the feature names in df total features

  tot_ft <- df %>% select(matches("^X\\d+$")) %>%

    # starts_with("X") # ^ start sign;

    # $ end of a string, d+ one or more digital

    ## This may be problematic if df is not generated from sim_surv_f

    colnames()


  set.seed(seed)


  boot_lst <- lapply(1:nboot, function(x) {

    ### sample boot_ft from tot_ft

    boot_feature <- sample(tot_ft, boot_ft, replace = FALSE) %>%

      naturalsort::naturalsort()


    ### sample nrow(df) people from df with replacement;


    boot_sample <- df %>%

      #### sample status = 1

      filter(status == 1) %>%

      sample_n(size = nrow(.), replace = TRUE) %>%
      bind_rows(df %>% filter(status == 0) %>%
                  #### sample status = 0
                  sample_n(size = nrow(.), replace = TRUE)) %>%
      select(time, status, all_of(boot_feature))
  })

  return(boot_lst)
}
boot_f1 <- function(df = NA,
                    nboot = 100,
                    boot_ft = 5,
                    gene_index = 5,
                    cov_vars = c('age_at_diagnosis','cancer_stage'),
                    seed = 20231106,
                    ...) {
  ## get the feature names in df total features
  tot_ft <- df[,-c(1:gene_index-1)] %>%
    # starts_with("X") # ^ start sign;
    # $ end of a string, d+ one or more digital
    ## This may be problematic if df is not generated from sim_surv_f
    colnames()

  set.seed(seed)

  boot_lst <- lapply(1:nboot, function(x) {
    ### sample boot_ft from tot_ft
    boot_feature <- sample(tot_ft, boot_ft, replace = FALSE) %>%
      naturalsort::naturalsort()
    ### sample nrow(df) people from df with replacement;

    boot_sample <- df %>%
      #### sample status = 1
      filter(status == 1) %>%
      sample_n(size = nrow(.), replace = TRUE) %>%
      bind_rows(df %>% filter(status == 0) %>%
                  #### sample status = 0
                  sample_n(size = nrow(.), replace = TRUE)) %>%
      select(time, status,all_of(cov_vars) ,all_of(boot_feature))
  })

  return(boot_lst)
}

HDSI_model_f <- function(boot_df_train = NA,
                         # boot_df_test = NA,
                         cov_interact = FALSE,
                         method = "lasso") {

  library(glmnet)

  ### x input in cv.glmnet
  y <- "survival::Surv(time,status)"
  ## f=stats::as.formula(paste(y," ~ .*.*."))
  f <- stats::as.formula(paste(y, " ~ .*.")) ## pair-wise interaction
  f1 <- stats::as.formula(paste(y, " ~ ."))

  if(cov_interact == TRUE){
    X <- stats::model.matrix(f, boot_df_train)[, -1]
  }else{
    X_tmp <-  boot_df_train %>%
      select(time, status,age_at_diagnosis,cancer_stage) %>%
      bind_cols(
        stats::model.matrix(f, boot_df_train %>%
                              select(-age_at_diagnosis,-
cancer_stage))[, -1] )
    X <- stats::model.matrix(f1, X_tmp)[,-1]
  }

  #X_test <- stats::model.matrix(f, boot_df_test)[, -1]
  ### y input in cv.glmnet
  time <- boot_df_train$time
  status <- boot_df_train$status
  Y <- Surv(boot_df_train$time, boot_df_train$status)
  #Y_test <- Surv(boot_df_test$time, boot_df_test$status)

  if (method == "lasso") {
    cv_fit <- glmnet::cv.glmnet(
      x = X, y = Y, alpha = 1,
      standardize = T, family = "cox", nfolds = 3
    )
  #  print('so far so good lasso')
    lambda.1se <- cv_fit$lambda.1se
    lambda.min <- cv_fit$lambda.min

    model_fit <- glmnet::glmnet(x = X, y = Y, lambda = lambda.1se,
alpha = 1, standardize = T, family = "cox")
    model_coef <- as.matrix(coef(model_fit, s = lambda.1se)) %>%
      data.frame() %>%
      mutate(varname = rownames(.))

    pred <- predict(model_fit, s = lambda.1se, newx = X, type = "link")

    model_cindex <- apply(pred, 2, Cindex, y = Y)

    model_out <- model_coef %>% mutate(model_cindex = model_cindex)

    # y1 = cbind(time = time, status = status)
    #
    # apply(risk_scores, 2, glmnet::Cindex, y=y1)
  }

  if (method == "ridge") {
  #  print("so far so good (ridge) ")
    cv_fit <- glmnet::cv.glmnet(
      x = X, y = Y, alpha = 0,
      standardize = T, family = "cox", nfolds = 3
    )

    lambda.1se <- cv_fit$lambda.1se
    lambda.min <- cv_fit$lambda.min

    model_fit <- glmnet::glmnet(x = X, y = Y, lambda = lambda.1se,
alpha = 0, standardize = T, family = "cox")

    model_coef <- as.matrix(coef(model_fit, s = lambda.1se)) %>%
      data.frame() %>%
      mutate(varname = rownames(.))
    pred <- predict(model_fit, s = lambda.1se, newx = X, type = "link")

    model_cindex <- apply(pred, 2, Cindex, y = Y)

    model_out <- model_coef %>% mutate(model_cindex = model_cindex)
  }

  return(model_out)
}

#boot_lst <- sim_set1[[11]]
inter_perf_f <- function(boot_lst=NA,
                         cov_interact = NA,
                         method = "lasso",
                         k = 2, ## cv fold number
                         qtl = NA,
                         Rf = NA){
  library(caret)
  library(dplyr)
  library(survival)
  library(stringr)
  m <- method
  model_out_all <- lapply(1:length(m), function(x) {
    lapply(1:length(boot_lst), function(y) {
      print(y)
      folds <- createFolds(boot_lst[[y]]$status, k = k, list = TRUE,
returnTrain = FALSE)

      ## I want to know how many true effects are
      ## selected in each boot dataset
      selected_true <- c("X1","X2","X3","X4","X5") %in%
        (boot_lst[[y]] %>% colnames())

      X1X2 <- ifelse((selected_true[1]==TRUE & selected_true[2]==TRUE),
                     TRUE, FALSE )
      X3X4 <- ifelse((selected_true[3]==TRUE & selected_true[4]==TRUE),
                     TRUE, FALSE )
      selected_X1 <- c("X1") %in%
        (boot_lst[[y]] %>% colnames())
      selected_X2 <- c("X2") %in%
        (boot_lst[[y]] %>% colnames())
      selected_X3 <- c("X3") %in%
        (boot_lst[[y]] %>% colnames())
      selected_X4 <- c("X4") %in%
        (boot_lst[[y]] %>% colnames())
      selected_X5 <- c("X5") %in%
        (boot_lst[[y]] %>% colnames())

      boot_true_margin <- selected_true %>% sum()
      boot_true_inter <- sum(X1X2,X3X4)



      ## the validation performance
      if(k!=1){
      res <- lapply(1:length(folds),function(z){
       # print(paste0('This is the ',y,'th boot sample ', z,'th fold'))
        res <-  HDSI_model_f(boot_df_train = boot_lst[[y]][-
folds[[z]],],
                             cov_interact = cov_interact,
                             # boot_df_test =
boot_lst[[y]][folds[[z]],],
                             method = m[[x]])
        res$boot = y
        res$fold = -z
        res$boot_true_margin = boot_true_margin
        res$boot_true_inter = boot_true_inter
        res$boot_trueX1= selected_X1
        res$boot_trueX2= selected_X2
        res$boot_trueX3= selected_X3
        res$boot_trueX4= selected_X4
        res$boot_trueX5= selected_X5

      })
      }
      if(k==1){
        res <-  lapply(1:length(folds),function(z){
          print(paste0('This is the ',y,'th boot sample ', z,'th
fold'))
          res <-  HDSI_model_f(boot_df_train =
boot_lst[[y]][folds[[z]],],
                               cov_interact = cov_interact,
                               # boot_df_test =
boot_lst[[y]][folds[[z]],],
                               method = m[[x]])
          res$boot = y
          res$fold = -z
          res$boot_true_margin = boot_true_margin
          res$boot_true_inter = boot_true_inter
          res
        })
      }

      res

    })
  })

  print('model_out_all computation done!! good job!!!')
  fs_out5 <- lapply(1:length(m), function(x) {
    ## bind B boot sample results into a dataframe
    perf_tmp <- model_out_all[[x]] %>% bind_rows()
    lapply(1:k,function(y){
      perf_onefold <-  perf_tmp[perf_tmp$fold==-y,]

      fe_star <-  perf_onefold %>%  group_by(model_cindex) %>%
summarise(n=n())
      ### compute min_cindex

      min_cindex <- perf_onefold %>%
        group_by(varname) %>%
        summarise(min_cindex = min(model_cindex)) %>%
        mutate(
          miu_min_cindex = mean(min_cindex),
          # sigma_min_cindex = sqrt(mean((min_cindex -
miu_min_cindex)^2) / (fe_star$n %>% unique() - 1)),
          sd_min_cindex = sd(min_cindex),
          Rf = Rf, ## hyperparameter
          include_yn = min_cindex > (miu_min_cindex + Rf *
sd_min_cindex)
        )

      ### compute coef estimate and CI
      beta_quantile <- perf_onefold %>%
        group_by(varname) %>%
        mutate(quantile = qtl) %>% ## quantile is a hyperparameter
        summarise(
          qtl = qtl,
          beta_hat = mean(X1),
          qtl_lower = quantile(X1, probs = qtl/2),
          qtl_upper = quantile(X1, probs = 1 - (qtl /2) ),
          include0_yn = !(((qtl_lower <= 0) & (0 <= qtl_upper)) |
((qtl_upper <= 0) & (0 <= qtl_lower)))
        )

      ### join beta and cindex in a dataframe; filter selected features
      perf_tmp1  <- full_join(beta_quantile, min_cindex, by =
"varname")

      perf <-  perf_tmp1 %>%
        filter(include0_yn == TRUE & include_yn == TRUE)

      ### adding back main effect if only interaction terms are
selected
      included_inter <- perf$varname[str_detect(perf$varname, ":")] ##
detect interaction terms

      ## detect main effect terms from interaction
      included_inter1 <- stringr::str_split(included_inter, ":") %>%
unlist()


      ## final selected features

      included_fe <- c(perf$varname, included_inter1) %>% unique()

      #included_fe

      perf2  <- perf_tmp1 %>%

        filter(varname %in% included_fe)

      list(included_fe,perf2)


    })


  })


  ###

  inter_vars_tmp <- lapply(1:length(m),function(x){

    lapply(1:k,function(y){

      fs_out5[[x]][[y]][[1]]

    })

  })


  ## calculate averaged performance over 5 datasets; internal
validation

  inter_cindex_tmp <- lapply(1:length(m),function(x){

    lapply(1:k,function(y){

      fs_out5[[x]][[y]][[2]] %>% bind_rows()

    })

  })


  inter_val_hyper <- lapply(1:length(m),function(x){
    avg_cindex <- inter_cindex_tmp[[x]] %>% bind_rows()%>%
      summarise(avg_cindex = mean(min_cindex))
    hyper_table <- data.frame(method = m[x],
                              avg_cindex = avg_cindex$avg_cindex,
                              hyper_quantile = qtl,
                              Rf = Rf
    )
  })


  return(list(model_out_all,fs_out5,inter_val_hyper))

}
```

Appendix Text

1. Simulate survival data using inverse transform sampling technique.

The survival function in the Cox model can be written as $S(t|x) = \exp\left[-H_0(t)\exp(Z)\right]$, so the cumulative distribution function $F(t|x)$ can be expressed as 1-exp $[-H_0(t)\exp(Z)]$. Since F is the cumulative distribution function, then $F(T)$ follows a uniform (0,1) distribution and $S(T) = 1 - F(T)$ also follows a uniform (0,1) distribution (i.e., U = $\exp[-H_0(t)\exp(Z)] \sim$ Uniform (0,1)). Therefore, the survival time T can be expressed as ${H_0}^{-1}(-\log(U)\exp(-Z))$, where U is a random variable with uniform (0,1) distribution. And ${H_0}^{-1}(.)$ is the inverse of cumulative hazard function.

Appendix Tables

Supplement Table 1: Correlation matrix of the five predictors

Supplement Table 1: correlation matrix of predictors

	X1	X2	X3	X4	X5
X1	0	0.3	0.3	0.6	0.6
X2	0.3	0	0.3	0.2	0.1
X3	0.3	0.3	0	0.2	0.1
X4	0.6	0.2	0.2	0	0.1
X5	0.6	0.1	0.1	0.1	0

Supplement Table 2: Cox regression model on the simulated dataset

```
survival:coxph(formula = Surv(time, status) ~ X1 + X2 + X3 + X4 + X5
+ X1*X2 + X3*X4, data = dat)
```

Supplement Table 2: Cox regression results on the simulated dataset

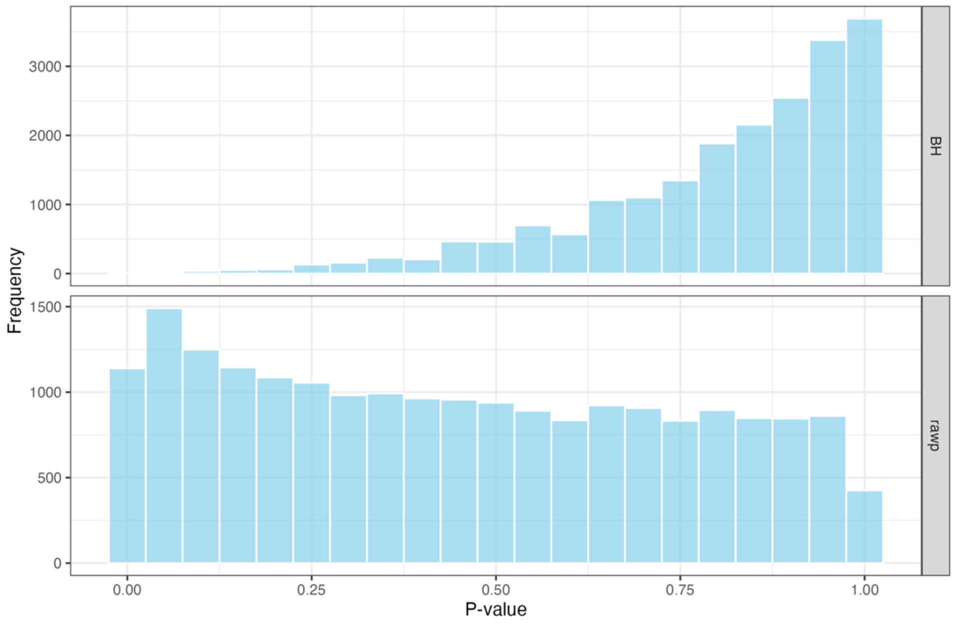
	coef	exp(coef)	se(coef)	Z	P-value
X1	0.982	2.670	0.139	7.085	1.39E-12
X2	1.138	3.121	0.098	11.648	<2e-16
X3	0.632	1.881	0.079	8.028	9.90E-16
X4	-0.778	0.459	0.108	-7.226	4.99E-13
X5	0.746	2.109	0.100	7.470	8.01E-14
X1:X2	1.062	2.893	0.110	9.675	<2e-16
X3:X4	-1.049	0.350	0.095	-11.024	<2e-16

Appendix Figures

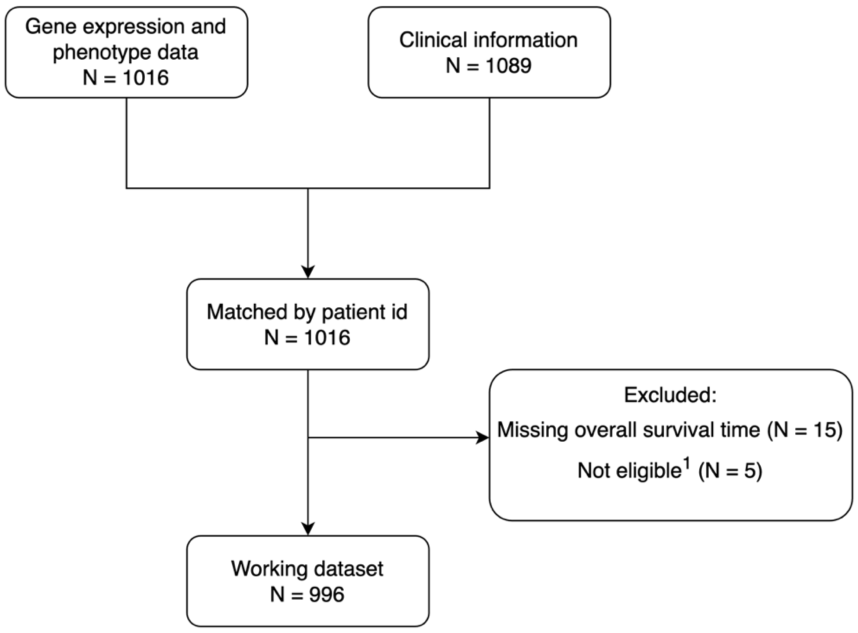
Supplement Figure 1: The Benjamini & Hochberg method for false discovery rate adjustment

The P-values of univariate data analysis before and after FDR adjustment using BH method (Adjusted p-values for the Benjamini & Hochberg (1995) step-up FDR-controlling procedure)

BH method: 25 out of 20531 (~0.1%) are of statistical significance (significance level: 0.05).

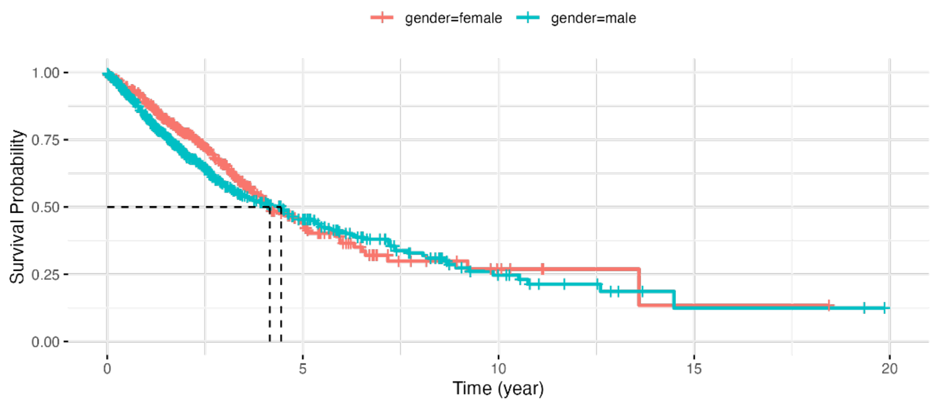


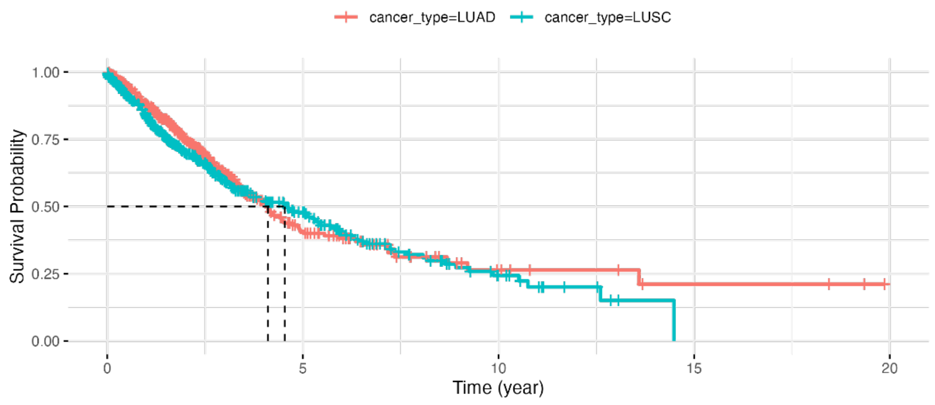
Supplement Figure 2: The inclusion and exclusion of subjects

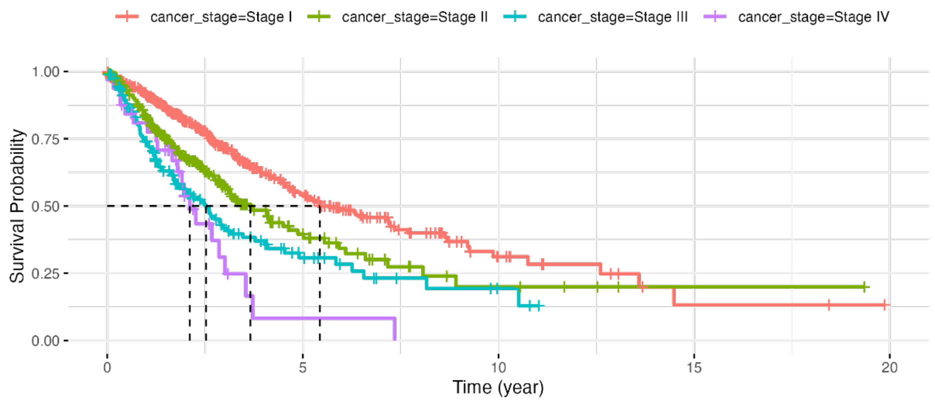

Gene expression and phenotype data N = 1016
Clinical information N = 1089
Matched by patient id N = 1016
Excluded:
Missing overall survival time (N = 15)
Not eligible1 (N = 5)
Working dataset N = 996

1: subjects who died or censored at the enrollment

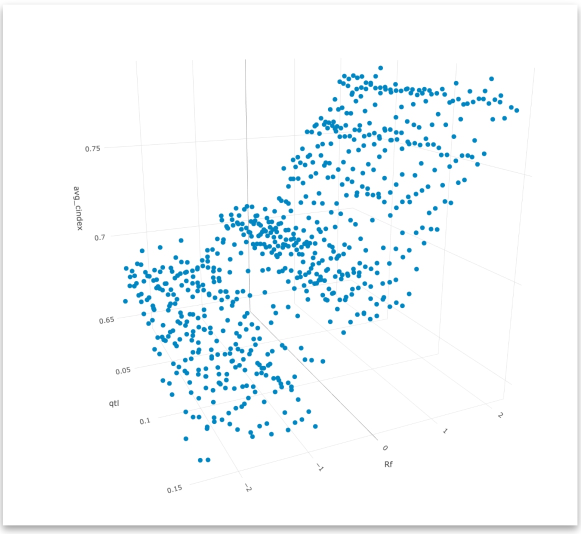
Supplement Figure 3: Kaplan-Meier curves in subgroups

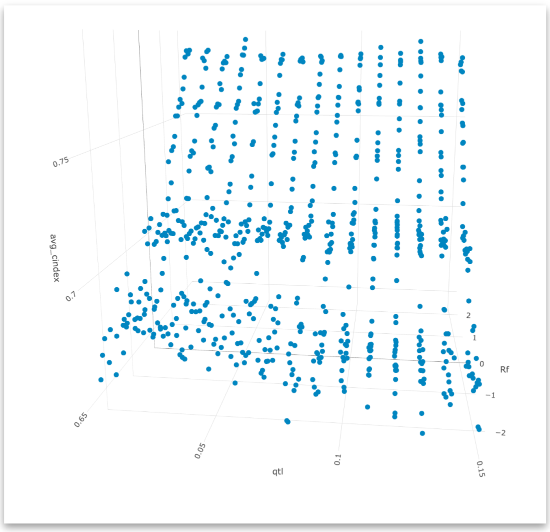


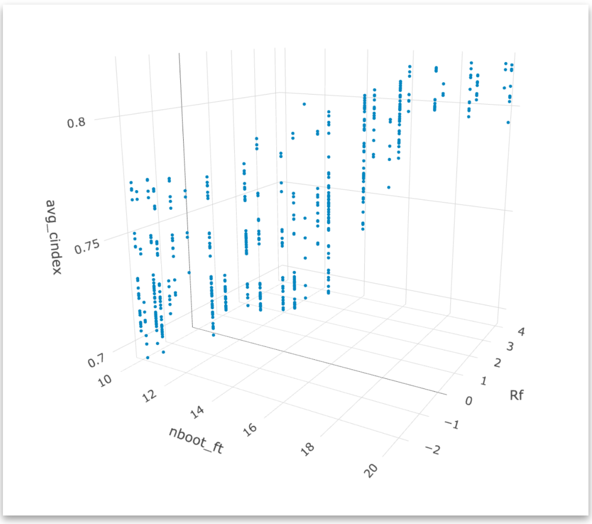




Supplement Figure 4: The hyper-parameter tuning results


avg_cindex
qtl
Rf


avg_cindex
qtl
Rf


avg_cindex
nboot_ft
Rf