Introduction to R code for "De-biased lasso for stratified Cox models with applications to the national kidney transplant data"

by Lu Xia, Bin Nan and Yi Li

This folder contains the following R/Rcpp files:

- `strat_cox_Clib.cpp` contains main functions in Rcpp that calculate the quantities related to the negative log of stratified partial likelihood and solve for the lasso estimator and its cross-validation.
- `strat_cox_Rlib.R` contains an auxiliary function that is called iteratively for solving the lasso problem in `strat_cox_Clib.cpp`.
- `simulated_example.R` provides a simulated example that uses the proposed de-biased lasso method for drawing inferences for stratified Cox models.

1 Library of key functions

The Rcpp file `strat_cox_Clib.cpp` contains the key functions that calculate the quantities related to the negative log of stratified partial likelihood and solve for the lasso estimator and its cross-validation. The functions following "`// [[Rcpp::export]]`" are exported to R that can be directly called in R.

- `all_neg_loglik_cpp_ext` calculates the negative log stratified partial likelihood $\ell(\beta)$; it can be used in the cross-validation for the tuning parameter γ without the burden to also compute $\dot{\ell}(\beta)$, $\ddot{\ell}(\beta)$ and $\widehat{\Sigma}$.
- `all_neg_loglik_function_cpp_ext` calculates the quantities related to the negative log stratified partial likelihood: $\ell(\beta)$, $\dot{\ell}(\beta)$, $\ddot{\ell}(\beta)$ and $\widehat{\Sigma}$.
- `lasso_stratCox_cpp_ext` solves the lasso estimator for some specified tuning parameter λ and returns a column vector as the solution.
- `cv_lasso_stratCox_cpp` implements cross-validation for the lasso estimator and automatically selects the optimal tuning parameter; it returns a list object containing the final lasso estimator (`beta_opt`), the λ sequence (`lambda_seq`) and the selected λ (`lambda_opt`), cross-validated values (`cv_value`), $\ell(\widehat{\beta})$ (`neg_loglik`), $\dot{\ell}(\widehat{\beta})$ (`neg_dloglik`), $\ddot{\ell}(\widehat{\beta})$ (`neg_ddloglik`) and the matrix $\widehat{\Sigma}$ (`score_sq`).

Before running the code, please make sure the required R packages (see those included in the R code) have been installed.

2 Simulated examples

`simulated_example.R` consists of seven parts and has been clearly annotated. The corresponding parts can be located as below.

- Simulation scenarios (Line 28–Line 65): four scenarios with different (K, n_k, p) combinations are considered as stated in the main article; this example highlights Scenario 1 due to its shortest running time and other scenarios may take considerably longer running time.

- The remaining setup that is common across scenarios (Line 68–Line 100): true regression parameters, baseline hazards and covariance matrix for covariates X.

- Data generation (Line 102–Line 131): covariates `X`, observed survival time `time`, event indicator `delta` and stratum index number `strata_idx`.

- Lasso estimator (Line 133–Line 162): the lasso estimator can be retrieved as `beta_glmnet`.

- Cross-validation for the tuning parameter γ in the proposed de-biased lasso approach, DBL-QP, as shown in Algorithm 1 (Line 169–Line 257): the sequence of tuning parameters for γ can be retrieved using `multiplier_seq*sqrt(log(p)/total_n)` and the cross-validated values using `all_cvpl2`.

- Implementation of the de-biased lasso estimator and inference with the selected tuning parameter γ (Line 259–293): the final DBL-QP estimator can be retrieved from `b_hat_new`, as well as its model-based standard error `se_new` and p-values `pval_new`.

- Other estimators in comparison (Line 296–Line 318): maximum stratified partial likelihood estimator and oracle estimator.

For simulation results, since the simulation for the four scenarios described in Section 4 would take extremely long time to finish on a desktop machine, the code was originally run as multiple array jobs on a cluster. Hence, the simulation results cannot be entirely reproduced on a desktop machine within reasonable time limit.

Interested readers may try one replication in one of the four scenarios described in Section 4. Running time for one replication varies a lot by scenario and CPU. Scenario 1, the simplest case among the four, usually takes < 1 minute to finish one replication. The other three scenarios may take a few hours.