A general, prediction error based criterion for selecting model complexity for high-dimensional survival models

Christine Porzelius\textsuperscript{1,2}, Martin Schumacher\textsuperscript{2}, Harald Binder\textsuperscript{1,2}

\textsuperscript{1}Freiburg Center for Data Analysis and Modeling
\textsuperscript{2}Institute of Medical Biometry and Medical Informatics
DFG Forschergruppe FOR 534

cp@fdm.uni-freiburg.de

July 1, 2009
Situation

- Risk prediction models for $p > n$ settings
- **Selection of optimal model complexity**, i.e. penalty parameter(s) (for shrinkage methods) or number of boosting steps (for boosting techniques), important issue, to avoid under- or overfitting
- **In time-to-event settings**: Typically done by predictive partial log-likelihood (PLL) for Cox proportional hazards model
- For judging prediction performance: prediction error curves, which only require a risk prediction (R package `peperr`)
Situation

- Risk prediction models for $p > n$ settings
- **Selection of optimal model complexity**, i.e. penalty parameter(s) (for shrinkage methods) or number of boosting steps (for boosting techniques), important issue, to avoid under- or overfitting
- **In time-to-event settings**: Typically done by predictive partial log-likelihood (PLL) for Cox proportional hazards model
- For judging prediction performance: prediction error curves, which only require a risk prediction (R package `peperr`)

An alternative to the PLL for model selection could be the integrated prediction error curve (IPEC)

- Does the IPEC perform as well as the PLL as model selection criterion?
- Is it reasonable to apply the IPEC in non-parametric approaches as well, such as tree-based methods?
Quantifying prediction performance: Apparent error

- Apparent error:

\[ \overline{err}(t; r) = \frac{1}{n} \sum_{i=1}^{n} (Y_i(t) - r_x(t|z_i))^2 W(t; \hat{G}; x_i), \]

with

- true event status \( Y_i(t) \) at time \( t \) (1: no event yet, 0: event),
- risk prediction model \( r(t|z_i) \), where the index indicates the observations used for fitting and \( p \)-vector of covariates \( z_i \),
- weights \( W(t; \hat{G}; x_i) \), which account for censoring (Gerds and Schumacher, 2006)
Quantifying prediction performance: Apparent error

- Apparent error:

\[
\overline{\text{err}}(t; r) = \frac{1}{n} \sum_{i=1}^{n} (Y_i(t) - r_x(t|z_i))^2 W(t; \hat{G}; x_i),
\]

with

- true event status \( Y_i(t) \) at time \( t \) (1: no event yet, 0: event),
- risk prediction model \( r(t|z_i) \), where the index indicates the observations used for fitting and \( p \)-vector of covariates \( z_i \),
- weights \( W(t; \hat{G}; x_i) \), which account for censoring (Gerds and Schumacher, 2006)
Quantifying prediction performance: Apparent error

- Apparent error:

\[
\bar{\text{err}}(t; r) = \frac{1}{n} \sum_{i=1}^{n} (Y_i(t) - r_x(t|z_i))^2 W(t; \hat{G}; x_i),
\]

with

- true event status \( Y_i(t) \) at time \( t \) (1: no event yet, 0: event),
- risk prediction model \( r(t|z_i) \), where the index indicates the observations used for fitting and \( p \)-vector of covariates \( z_i \),
- weights \( W(t; \hat{G}; x_i) \), which account for censoring (Gerds and Schumacher, 2006)
Compute the apparent error

Simplified example:

cplx <- complexity(y, x)
model <- fit.model(y, x, cplx)
pred.fun(model)

y  time-to-event response
x  data set
complexity function for complexity selection
cplx model complexity parameter(s)
fit.model function to fit predictive model
pred.fun function to determine prediction error
Compute the apparent error

Simplified example:

```r
cplx <- complexity(y, x)
model <- fit.model(y, x, cplx)
pred.fun(model)
```

- `y`: time-to-event response
- `x`: data set
- `complexity`: function for complexity selection
- `cplx`: model complexity parameter(s)
- `fit.model`: function to fit predictive model
- `pred.fun`: function to determine prediction error

But: The same data for validation and testing is used, i.e. the true prediction error is potentially underestimated.
Quantifying prediction performance: Bootstrap error estimate

- Bootstrap error estimate:

\[
\hat{\text{Err}}_{B0}(t; r) = \frac{1}{B} \sum_{b=1}^{B} \frac{1}{|x_b|} \sum_{i \in x_b} \left( Y_i(t) - r_{x_b}(t|z_i) \right)^2 W(t; \hat{G}; x_i),
\]

with \( B \) bootstrap samples \( x_1, \ldots, x_B \) and corresponding test sets \( x_{-1}, \ldots, x_{-B} \).
Quantifying prediction performance: Bootstrap error estimate

- Bootstrap error estimate:

$$\bar{\text{Err}}_{B0}(t; r) = \frac{1}{B} \sum_{b=1}^{B} \frac{1}{|x_b - \cdot|} \sum_{i \in x_b} \left( Y_i(t) - r_{x_b}(t|z_i) \right)^2 W(t; \hat{G}; x_i),$$

with $B$ bootstrap samples $x_1, \ldots, x_B$ and corresponding test sets $x_{-1}, \ldots, x_{-B}$.

- Drawing the bootstrap samples with replacement is sufficient for a fixed complexity value.
- When complexity selection is performed in bootstrap samples, drawing without replacement is advisable (Binder and Schumacher, 2008a).
Compute the Bootstrap error estimate

Simplified example:

```r
for (i in 1:n.resample){
  cplx.resample <- complexity(y.train[[i]], x.train[[i]])
  model <- fit.model(y.train[[i]], x.train[[i]], cplx.resample)
  pred.fun(model, x.test[[i]])
}
```

- `n.resample`: number of train and test data sets
- `x.train`: list of training data sets obtained by resampling
- `y.train`: corresponding response
- `complexity`: function for complexity selection
- `cplx.resample`: model complexity parameter in given training set
- `x.test`: list of test data sets obtained by resampling
- `pred.fun`: function to determine prediction error

But: Known to be biased upwards, i.e. overestimating the true prediction error.
Compute the Bootstrap error estimate

Simplified example:

```r
for (i in 1:n.resample){
    cplx.resample <- complexity(y.train[[i]], x.train[[i]])
    model <- fit.model(y.train[[i]], x.train[[i]], cplx.resample)
    pred.fun(model, x.test[[i]])
}
```

- `n.resample`: number of train and test data sets
- `x.train`: list of training data sets obtained by resampling
- `y.train`: corresponding response
- `complexity`: function for complexity selection
- `cplx.resample`: model complexity parameter in given training set
- `x.test`: list of test data sets obtained by resampling
- `pred.fun`: function to determine prediction error

But: Known to be biased upwards, i.e. overestimating the true prediction error.
Bootstrap .632+ estimate (Gerds and Schumacher, 2007)

- Bootstrap .632+ estimate:
  Weighted combination of apparent error \( \text{err}(t; r) \) and bootstrap error estimate \( \hat{\text{Err}}_{B0}(t; r) \) (Efron and Tibshirani, 1997):

\[
\hat{\text{Err}}_{.632+}(t; r) = (1 - w(t)) \text{err}(t; r) + w(t) \hat{\text{Err}}_{B0}(t; r),
\]

with weight \( w(t) = 0.632/(1 - 0.368 \hat{R}(t)) \), based on relative overfit \( \hat{R}(t) \)

- Tracking an error estimate over time results in a prediction error curve
Bootstrap .632+ estimate using peperr

peperr(y, x,
    fit.fun=fit.model, complexity=complexity)

y time-to-event response
x data set
fit.model function to fit predictive model
complexity function for complexity selection or fixed value
Example: prediction error curves

Null model natural benchmark

![Graph showing prediction error curves for the null model over evaluation time points. The Y-axis represents prediction error ranging from 0.00 to 0.30, while the X-axis represents evaluation time points ranging from 0 to 15. The graph shows a curve for the null model that peaks at around the 5th evaluation time point and then decreases.]
Example: prediction error curves

![Prediction Error Curves](image-url)
Example: prediction error curves
Example: prediction error curves
Example: prediction error curves

Strong variation in bootstrap samples ⇒ weighting required

[Graph showing prediction error curves for different models, including Null model, Apparent error, Bootstrap samples, Bootstrap estimate, and .632+ estimate.]
Criteria for complexity selection

Selection of the optimal complexity parameter by

1. Partial log-likelihood (PLL)

\[
\ell(\beta) = \sum_{i=1}^{n} \delta_i (z'_i \beta - \log(\sum_{j=1}^{n} I(t_j \geq t_i) \exp(z'_j \beta)))
\]
Criteria for complexity selection

Selection of the optimal complexity parameter by

(1) Partial log-likelihood (PLL)

\[ \ell(\beta) = \sum_{i=1}^{n} \delta_i (z_i' \beta - \log(\sum_{j=1}^{n} I(t_j \geq t_i) \exp(z_j'\beta))) \]

(2) Brier score: Comparison of true event status \( Y(t) \) and risk prediction \( \hat{r}(t|z) \)

\[ BS(t, \hat{r}) = E(Y(t) - \hat{r}(t|z))^2 \]

\[ IPEC(\hat{r}) = \int_0^{\infty} BS(t, \hat{r})p(t)dt, \]

with probability density \( p(t) \)

choosing the complexity value that maximizes PLL / minimizes IPEC
Estimation of complexity selection criteria

(1) Predictive partial log-likelihood (Verweij and van Houwelingen, 1993):

\[
\ell_{\text{pred}}(\hat{\beta}_{-b}) = \sum_{b=1}^{B} (\ell(\hat{\beta}_{-b}) - \ell_{-b}(\hat{\beta}_{-b})) ,
\]

where index \( -b \) indicates that the \( b \)th part of the data is left out, estimated via \( B \) cross-validation or bootstrap samples.
Estimation of complexity selection criteria

(1) Predictive partial log-likelihood (Verweij and van Houwelingen, 1993):

\[
\ell_{\text{pred}}(\hat{\beta}_{-b}) = \sum_{b=1}^{B} (\ell(\hat{\beta}_{-b}) - \ell_{-b}(\hat{\beta}_{-b})),
\]

where index \(-b\) indicates that the \(b\)th part of the data is left out, estimated via \(B\) cross-validation or bootstrap samples.

(2) Sum of the bootstrap .632+ estimate at timepoints \(t^* = \{t_1^*, \ldots , t_S^*\}:

\[
\hat{I}_{\text{PEC}.632+}(t^*; \hat{r}) = \sum_{s=1}^{S-1} \hat{\text{Err}}.632+ (t_s^*; \hat{r}) (\hat{r}^{KM}(t_s^*) - \hat{r}^{KM}(t_{s+1}^*))
\]

with Kaplan-Meier estimate \(\hat{r}^{KM}(t)\) at time \(t\).
Relative integrated prediction error curve

- For small \( n \), time structure of single bootstrap samples might differ strongly
- Good prediction performance might be harder to obtain in one bootstrap sample than in another one
Relative integrated prediction error curve

- For small $n$, time structure of single bootstrap samples might differ strongly.
- Good prediction performance might be harder to obtain in one bootstrap sample than in another one.

$\Rightarrow$ We consider only the relative gain in each bootstrap sample, compared to the null model $\hat{r}^{KM}$ (Kaplan-Meier estimate), i.e.

$$
\overline{err}^{rel}(t; \hat{r}) = \overline{err}(t; \hat{r}) - \overline{err}(t; \hat{r}^{KM}) \quad \text{and}
$$

$$
\widehat{Err}^{rel}_{B0}(t; \hat{r}) = \frac{1}{B} \sum_{b=1}^{B} \left( \left( \sum_{i: x_i \in x_b} (Y_i(t) - \hat{r}_{xb}(t|z_i))^2 \right) W(t; \hat{G}; x_i) \right) - (Y_i(t) - \hat{r}_{xb}^{KM}(t|z_i))^2 )
$$

instead of $\overline{err}(t; \hat{r})$ and $\widehat{Err}_{B0}(t; \hat{r})$.
Considered approaches

- Proportional hazards model: CoxBoost (Binder and Schumacher, 2008b)
  - Componentwise likelihood based boosting approach
  - Here: Update of one element of the parameter vector in each boosting step to obtain sparse model fits.
  - Model complexity corresponds to number of boosting steps.

- Semi-/Non-parametric: Random survival forests (Ishwaran et al., 2008)
  - Survival trees are constructed for bootstrap samples of the data.
  - At each split, $m$ try covariates are randomly selected as candidates.
  - Ensemble cumulative hazard estimate by combining the information from the trees.
  - The complexity parameter that has to be determined is $m_{try}$. 
Considered approaches

- Proportional hazards model: CoxBoost (Binder and Schumacher, 2008b)
  - Componentwise likelihood based boosting approach
  - Here: Update of one element of the parameter vector in each boosting step to obtain sparse model fits.
  - Model complexity corresponds to number of boosting steps.

- Semi-/Non-parametric: Random survival forests (Ishwaran et al., 2008)
  - Survival trees are constructed for bootstrap samples of the data.
  - At each split, $m_{\text{try}}$ covariates are randomly selected as candidates.
  - Ensemble cumulative hazard estimate by combining the information from the trees.
  - The complexity parameter that has to be determined is $m_{\text{try}}$. 
Data simulation

Simulation of

- **Training data**, mimicking microarray survival data (Binder and Schumacher, 2008a) with
  - $n$ observations, of which approximately 50% are censored, and
  - $p$ in blocks correlated covariates, of which 10 are little/highly informative ($c_e = 0.05/c_e = 0.1$),

resulting in six settings:

<table>
<thead>
<tr>
<th></th>
<th>$c_e = 0.05$</th>
<th></th>
<th>$c_e = 0.1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$n_1 = 50$</td>
<td>$n_2 = 200$</td>
<td>$n_1 = 50$</td>
</tr>
<tr>
<td>$p_1 = 1000$</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>$p_2 = 5000$</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
</tbody>
</table>

- Corresponding **test data** sets with 5000 observations each 50 repetitions
Procedure

1. **Selection of optimal complexity parameter** on training data by
   - predictive partial log-likelihood (PLL), if available, and
   - relative integrated prediction error curve (IPEC).

   estimated by the bootstrap (bs) or cross-validation (cv)
Procedure

1. **Selection of optimal complexity parameter** on training data by
   - predictive partial log-likelihood (PLL), if available, and
   - relative integrated prediction error curve (IPEC).
   estimated by the bootstrap (bs) or cross-validation (cv)

2. **Corresponding test data:**
   - Determination of **prediction performance** of selected model by
     - predictive partial log-likelihood (PLL), if available, multiplied by \(-2\) and
     - integrated prediction error curve (IPEC).
   - Additionally, each test set is used to determine the actual optimal complexity parameter value, which would be obtained by an optimal criterion (opti).
   - In general, only the difference to the null model is shown for each data set. \(\rightarrow\) Larger values are better.
Prediction performance (CoxBoost) measured by PLL

\( n_2 = 200, \ p_1 = 1000, \ c_e = 0.05 \)
Prediction performance (CoxBoost) measured by IPEC

\( n_2 = 200, p_1 = 1000, c_e = 0.05 \)
## Prediction performance (CoxBoost) measured by PLL

<table>
<thead>
<tr>
<th>n</th>
<th>p</th>
<th>$c_e$</th>
<th>opti PLL10Bs</th>
<th>75021</th>
<th>75179</th>
<th>75177</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>PLL10Bs</td>
<td>74545</td>
<td>74618</td>
<td>74616</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IPEC10Bs</td>
<td>74589</td>
<td>74641</td>
<td>74628</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>73627</td>
<td>73783</td>
<td>73798</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>72790</td>
<td>72853</td>
<td>72913</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>72867</td>
<td>72919</td>
<td>73113</td>
</tr>
</tbody>
</table>

(absolute values shown)
## Simulation study

Prediction performance measured by IPEC

<table>
<thead>
<tr>
<th>n</th>
<th>p</th>
<th>$c_e$</th>
<th>opti</th>
<th>PLL10Bs</th>
<th>IPEC10Bs</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>1000</td>
<td>0.05</td>
<td>0.1670</td>
<td>0.1685</td>
<td>0.1687</td>
</tr>
<tr>
<td>200</td>
<td>1000</td>
<td>0.05</td>
<td>0.1559</td>
<td>0.1568</td>
<td>0.1567</td>
</tr>
<tr>
<td>200</td>
<td>5000</td>
<td>0.05</td>
<td>0.1565</td>
<td>0.1577</td>
<td>0.1576</td>
</tr>
<tr>
<td>50</td>
<td>1000</td>
<td>0.1</td>
<td>0.1434</td>
<td>0.1462</td>
<td>0.1460</td>
</tr>
<tr>
<td>200</td>
<td>1000</td>
<td>0.1</td>
<td>0.1286</td>
<td>0.1286</td>
<td>0.1296</td>
</tr>
<tr>
<td>200</td>
<td>5000</td>
<td>0.1</td>
<td>0.1290</td>
<td>0.1297</td>
<td>0.1313</td>
</tr>
</tbody>
</table>

(absolute values shown)
Prediction performance (Random survival forests)

\[ n_2 = 200, \ p_1 = 1000, \ c_e = 0.05 \]
Rosenwald data: Comparison of different criteria

- Example: DLBCL study (Rosenwald et al., 2002):
  - $n = 240$ patients with diffuse large-B-cell lymphoma (DLBCL)
  - $p = 7399$ microarray features for each patient
  - 138 patients die during their individual follow up time

- CoxBoost: Selection of optimal number of boosting steps

- Selection criteria:
  - PLL (10-fold cross-validation)
  - relative IPEC (10 bootstrap samples)

- Evaluation of resulting prediction performance by
  - $-2 \cdot \text{PLL}$
  - IPEC

using 100 bootstrap samples without replacement (Binder and Schumacher, 2008a), each of size $[0.632n] = 152$. 
Rosenwald data: IPEC vs. PLL

Selected boosting steps

Null model – (−2) PLL

Null model – IPEC

PLL criterion

IPEC criterion
Rosenwald data: IPEC vs. PLL

**Selected boosting steps**

**Null model – (−2) PLL**

**Null model – IPEC**
Rosenwald data: IPEC vs. PLL

Selected boosting steps

Null model − (−2) PLL

Null model − IPEC
Breast cancer data

Breast cancer data of van de Vijver et al. (2002) with
- 295 patients with primary breast carcinomas
- 70-gene prognosis profile
- 79 patients die during their individual follow-up time
Breast cancer data

- Breast cancer data of van de Vijver et al. (2002) with
  - 295 patients with primary breast carcinomas
  - 70-gene prognosis profile
  - 79 patients die during their individual follow-up time

- Comparison of CoxBoost and random survival forests:
  1. Selection of optimal number of boosting steps / optimal value of $m_{try}$ on training data set by relative IPEC
  2. Evaluation of resulting prediction performance on test data set by IPEC using 100 bootstrap samples without replacement.
Comparison of random survival forests and CoxBoost

Prediction error measured by IPEC

![Scatter plot comparing IPEC criterion for CoxBoost and Random survival forests](scatter-plot.png)
Model selection by predictive PLL: Estimation via the bootstrap works as well as (leave-one-out) cross-validation.

10 bootstrap samples result in performance similar to 100 bootstrap samples.

Model selection by IPEC is not consistently better or worse than by predictive PLL
⇒ Reasonable alternative in cases where the PLL is not available.
Model selection by predictive PLL: Estimation via the bootstrap works as well as (leave-one-out) cross-validation.

10 bootstrap samples result in performance similar to 100 bootstrap samples.

Model selection by IPEC is not consistently better or worse than by predictive PLL

⇒ Reasonable alternative in cases where the PLL is not available.

Model selection by IPEC performed reasonable in non-parametric examples as well (random survival forests).
R package peperr (Porzelius et al., 2009)

- (Potentially parallelized) prediction error estimation using resampling
- Based on `snowfall/sfCluster`
- Complexity selection incorporated, or determination of prediction error for different values of complexity parameter(s)
- Easy connection to all model fitting routines, where risk predictions are available
- Simple usage, only risk prediction function required
References