Unit 7

Ensemble learning: Boosting

Boosting

- Originally designed for classification problems, but has also been extended to regression.
- Combines “weak” classifiers to produce a powerful “committee”.
- Iterative procedure where weak classifiers are fitted to a sequence of datasets which are constructed by reweighting the observations depending on how well they are predicted by the current estimate.
- If the number of iterations is too high, overfitting will occur. However, in general *slow overfitting behavior* is observed.

AdaBoost

- An algorithm for binary classification proposed by Freund and Schapire (1997).
- Assume that the output variable Y is coded as $\{-1, 1\}$.
- Given a vector of predictor variables X, a classifier $G(X)$ produces a prediction as one of the two values $\{-1, 1\}$.
- The error rate on the training sample is

$$\overline{\text{err}} = \frac{1}{N} \sum_{i=1}^{N} I(y_i \neq G(x_i)),$$

and the expected error rate for future predictions is

$$\text{E}_{XY} I(Y \neq G(X)).$$

AdaBoost / 2

- In boosting weak classifiers (i.e., classifiers slightly better than random guessing) are sequentially fitted to modified versions of the data, leading to the sequence of classifiers $G_m(x)$, $m = 1, 2, \ldots, M$.
- The predictions are combined through a weighted majority vote to produce the final prediction:

$$G(x) = \operatorname{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right),$$

  where α_m, $m = 1, 2, \ldots, M$ are computed by the boosting algorithm with higher weights assigned to more accurate classifiers.
- The data modifications at each boosting step consist of applying weights w_1, w_2, …, w_N to each training observation.

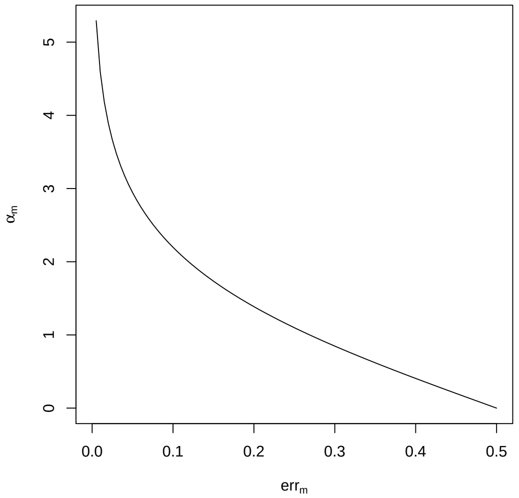
AdaBoost / 3

- Initially all weights are set to $w_i = 1/N$.
- For each successive iteration the weights are modified such that those observations misclassified in step $m - 1$ receive a higher weight in step m.

AdaBoost: algorithm

1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^N w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^M \alpha_m G_m(x)\right]$.

AdaBoost: algorithm / 2



Boosting fits an additive model

- Boosting fits an additive expansion in a set of elementary “basis” functions.
- The individual classifiers $G_m(x) \in \{-1, 1\}$ are the basis functions.
- Basis function expansions take the form

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m),$$

  where

  - β_m, $m = 1, 2, \ldots, M$ are the expansion coefficients and
  - $b(x; \gamma) \in \mathbb{R}$ are usually simple functions of the multivariate argument x characterized by a set of parameters γ.

Boosting fits an additive model / 2

- Typically these additive basis function expansions are fitted by minimizing a loss function averaged over the training data, such as the squared-error or a likelihood-based loss function,

$$\min_{\{\beta_m,\gamma_m\}_1^M} \sum_{i=1}^N L\left(y_i, \sum_{m=1}^M \beta_m b(x_i;\gamma_m)\right).$$

- For many loss functions $L(y, f(x))$ and/or basis functions $b(x;\gamma)$, the minimization requires intensive numerical optimization techniques.
- A simple alternative often can be found if it is feasible to rapidly solve the subproblem of fitting just a single basis function

$$\min_{\beta,\gamma} \sum_{i=1}^N L(y_i, \beta b(x_i;\gamma)).$$

Forward stagewise additive modeling

- New basis functions are sequentially added without adjusting the parameters and coefficients of those that have already been added.
- For squared-error loss

$$L(y, f(x)) = (y - f(x))^2,$$

  one has

$$\begin{aligned} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) &= (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2 \\ &= (r_{im} - \beta b(x_i; \gamma))^2, \end{aligned}$$

  where $r_{im} = y_i - f_{m-1}(x_i)$ is the residual of the current model on the ith observation.
- For the squared-error loss the term $\beta_m b(x_i; \gamma_m)$ fits best the current residuals and is added to the expansion.

Forward stagewise additive modeling: algorithm

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute

   $$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$

   (b) Set

   $$f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m).$$

Exponential loss and AdaBoost

- AdaBoost is equivalent to forward stagewise modeling using the loss function

$$L(y, f(x)) = \exp(-yf(x)).$$

- For AdaBoost the basis functions are the individual classifiers $G_m(x) \in \{-1, 1\}$.
- Using the exponential loss function, one must solve

$$(\beta_m, G_m) = \arg\min_{\beta, G} \sum_{i=1}^{N} \exp[-y_i(f_{m-1}(x_i) + \beta G(x_i))]$$

for the classifier G_m and the corresponding coefficient β_m to be added at each step.

Exponential loss and AdaBoost / 2

- This can be expressed as

$$(\beta_m, G_m) = \arg\min_{\beta,G} \sum_{i=1}^{N} w_i^{(m)} \exp[-\beta y_i G(x_i)],$$

with $w_i^{(m)} = \exp(-y_i f_{m-1}(x_i))$ which does neither depend on β nor on $G(x)$.

Exponential loss and AdaBoost / 3

- The solution can be obtained in two steps:
  1. For any value $\beta > 0$ the solution for $G_m(x)$ is

$$G_m = \arg\min_G \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i)),$$

     which is the classifier minimizing the weighted error rate in predicting y.

Exponential loss and AdaBoost / 4

2. Plugging this G_m into the criterion and solving for β one obtains

$$\beta_m = \frac{1}{2} \log \frac{1 - \mathrm{err}_m}{\mathrm{err}_m},$$

where err_m is the minimized weighted error rate

$$\mathrm{err}_m = \frac{\sum_{i=1}^N w_i^{(m)} I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i^{(m)}}.$$

Exponential loss and AdaBoost / 5

- The approximation is then updated by

$$f_m(x) = f_{m-1}(x) + \beta_m G_m(x),$$

which causes the weights for the next iteration to be

$$w_i^{(m+1)} = w_i^{(m)} \cdot e^{-\beta_m y_i G_m(x_i)} = w_i^{(m)} \cdot e^{\alpha_m I(y_i \neq G_m(x_i))} \cdot e^{-\beta_m},$$

for $\alpha_m = 2\beta_m$ and using $-y_i G_m(x_i) = 2 \cdot I(y_i \neq G_m(x_i)) - 1$.

Why exponential loss?

- It is easy to show that the population minimizer is given by

$$f^*(x) = \arg\min_{f(x)} \mathrm{E}_{Y|x}(e^{-Yf(x)}) = \frac{1}{2}\log\frac{\mathrm{Prob}(Y=1|x)}{\mathrm{Prob}(Y=-1|x)},$$

  or equivalently

$$\mathrm{Prob}(Y=1|x) = \frac{1}{1+e^{-2f^*(x)}}.$$

Why exponential loss? / 2

- Another loss criterion with the same population minimizer is the binomial negative log-likelihood or *deviance*, interpreting f as the logit transform. Let

$$p(x) = \text{Prob}(Y = 1|x) = \frac{1}{1 + e^{-2f(x)}}$$

and define $Y' = (Y + 1)/2 \in \{0, 1\}$. Then the binomial log-likelihood loss function is

$$\ell(Y', p(x)) = Y' \log p(x) + (1 - Y') \log(1 - p(x)),$$

or equivalently

$$-\ell(Y, f(x)) = \log(1 + e^{-2Yf(x)}).$$

Loss functions and robustness

- The "margin" $yf(x)$ plays a similar role in classification as the residual $y - f(x)$ in regression.
- The classification rule $G(x) = \text{sign}(f(x))$ implies:
  - A positive margin $y_i f(x_i) > 0$ indicates correctly classified observations.
  - A negative margin $y_i f(x_i) < 0$ indicates incorrectly classified observations.
  - The decision boundary is defined by $f(x) = 0$.
- Exponential and deviance loss can be seen as continuous approximations to the misclassification loss. Both penalize increasingly negative margins more than positive margins.
- The deviance penalty increases linearly for large increasingly negative margins, while this is exponential for the exponential loss. $\Rightarrow$ Deviance loss is more robust.

Loss functions and robustness / 2

- At any point of the training process the exponential criterion concentrates much more influence on observations with large negative margins. The deviance penalty more evenly spreads out the influence among all of the data.
  $\Rightarrow$ Better performance of deviance penalty in situations where the Bayes error rate is not close to zero and where there are misspecifications of class labels in the training data.
- The population risk minimizer for squared-error loss is

$$f^*(x) = \arg\min_{f(x)} \mathrm{E}_{Y|x}(Y - f(x))^2 = \mathrm{E}(Y|x) = 2\mathrm{Prob}(Y = 1|x) - 1.$$

  The classification rule is again $\mathrm{sign}(f(x))$.

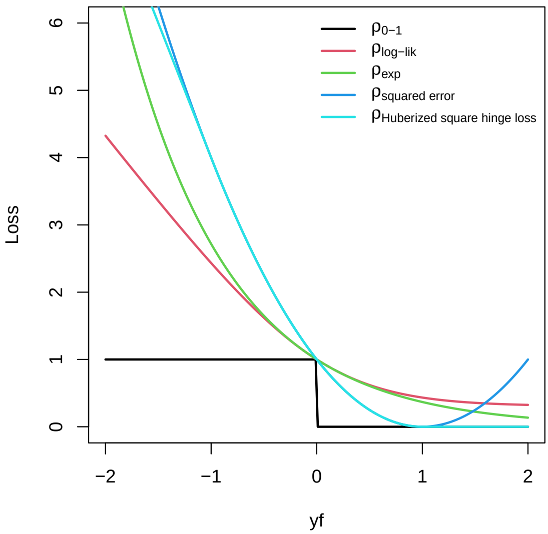
Loss functions and robustness / 3

- Compared to squared error loss, exponential and deviance loss are monotone decreasing criteria which are thus better surrogate loss functions for misclassification loss.
  $\Rightarrow$ The “Huberized” square hinge loss was proposed:

$$L(y, f(x)) = \begin{cases} -4yf(x) & yf(x) < -1, \\ ((1 - yf(x))_+)^2 & \text{otherwise,} \end{cases}$$

  with $()_+$ indicating the positive part.
  - It has the same population minimizer as squared error loss.
  - It is zero for $y \cdot f(x) > 1$.
  - It becomes linear for $y \cdot f(x) < -1$.
  - It has the advantage that quadratic functions are easier to compute than exponentials.

Loss functions and robustness / 4



Boosted trees

- Boosted trees are a sum of trees given by

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m),$$

  which are induced in a forward stagewise manner and where Θ_m denote the parameters of the tree consisting of the regions R_j and the node values γ_j.
- At each step in the forward stagewise manner procedure one must solve

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)),$$

  given the current model $f_{m-1}(x)$.

Boosted trees / 2

- Given the regions R_{jm} finding the optimal constants γ_{jm} in each region is typically straightforward.
- The difficulty of finding the regions depends on the loss:
  - For squared error loss, only the regression tree best predicting the current residuals $y_i - f_{m-1}(x_i)$ needs to be determined.
  - For exponential loss in 2-class classification, this approach leads to the AdaBoost method for boosting classification trees.

Gradient boosting

- Fast approximate algorithms for solving

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m))$$

  with any differentiable loss function can be derived by analogy to numerical optimization.
- The loss in using $f(x)$ to predict y on the training data is

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i)).$$

- The goal is to minimize $L(f)$ with respect to f, where f is constrained to be a sum of trees.

Gradient boosting / 2

- Ignoring this constraint, minimizing can be viewed as a numerical optimization problem

$$\hat{\boldsymbol{f}} = \arg\min_{\boldsymbol{f}} L(\boldsymbol{f}),$$

where the parameters $\boldsymbol{f} \in \mathbb{R}^N$ are the values of the approximating function $f(x_i)$ at each of the N data points x_i:

$$\boldsymbol{f} = \{f(x_1), \ldots, f(x_N)\}.$$

- Numerical optimization procedures obtain the solution as a sum of component vectors

$$\boldsymbol{f}_M = \sum_{m=0}^{M} \boldsymbol{h}_m, \qquad \boldsymbol{h}_m \in \mathbb{R}^N,$$

where $\boldsymbol{f}_0 = \boldsymbol{h}_0$ is an initial guess.

Gradient boosting / 3

- Each successive $\boldsymbol{f}_m$ is induced based on the current parameter vector $\boldsymbol{f}_{m-1}$.
- Numerical optimization methods differ in their prescription for computing each increment vector $\boldsymbol{h}_m$ (“step”).

Steepest descent

- Steepest descent chooses

$$\boldsymbol{h}_m = -\rho_m \boldsymbol{g}_m,$$

  where ρ_m is a scalar and $\boldsymbol{g}_m \in \mathbb{R}^N$ is the gradient of $L(\boldsymbol{f})$ evaluated at $\boldsymbol{f} = \boldsymbol{f}_{m-1}$.
- The components of $\boldsymbol{g}_m$ are

$$g_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}.$$

- The *step length* ρ_m is the solution to

$$\rho_m = \arg\min_{\rho} L(\boldsymbol{f}_{m-1} - \rho \boldsymbol{g}_m).$$

Steepest descent / 2

- The current solution is then updated

$$\boldsymbol{f}_m = \boldsymbol{f}_{m-1} - \rho_m \boldsymbol{g}_m.$$

- Steepest descent can be viewed as a very greedy strategy, since $-\boldsymbol{g}_m$ is the local direction in $\mathbb{R}^N$ for which $L(\boldsymbol{f})$ is most rapidly decreasing at $\boldsymbol{f} = \boldsymbol{f}_{m-1}$.

Gradient boosting

- At each step a base learner is fitted such that the predictions $\boldsymbol{t}_m$ are as close as possible to the negative gradient.
- For trees as base learners and squared error loss this gives

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} (-g_{im} - T(x_i; \Theta))^2.$$

- This can be achieved by fitting the tree T to the negative gradient values by least squares.

Gradient tree boosting algorithm

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^N L(y_i, \gamma)$.
2. For $m = 1, \ldots, M$:
   (a) For $i = 1, 2, \ldots, N$ compute
   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$$
   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute
   $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma).$$
   (d) Update
   $$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm}).$$
3. Output $\hat{f}(x) = f_M(x)$.

Right-sized trees for boosting

- Selecting a suitable tree size separately for each boosting step increases computation and degrades performance because it is based on the assumption that each tree is the last one in the expansion.
- The simplest strategy is to restrict all trees to be the same size $J_m = J, \forall m$.
  $\Rightarrow$ J is a meta-parameter of the entire boosting procedure.
- The interaction level of tree-based approximations is limited by the tree size J. Only interactions effects up to level $J - 1$ are possible.
  - $J = 2$ produces boosted models including only main effects.
  - For $J = 3$ two-variable interactions are possible.

  $\Rightarrow$ One can choose J depending on the level of dominant interactions in the target function.
- In many applications $J = 2$ will be insufficient, but also $J > 10$ will be unlikely.

Regularization

- The number of boosting iterations M is another meta-parameter.
- Each iteration usually reduces the training risk $L(f_M)$ implying that M too large will lead to overfitting.
- M^* could be selected based on a validation or test dataset.
- Further shrinkage is possible by reducing the contribution of each tree by a factor $0 < \nu < 1$:

$$f_m(x) = f_{m-1}(x) + \nu T(x, \Theta_m).$$

- The parameter ν is also regarded as controling the learning rate of the boosting procedure.
- Smaller values of ν induce more shrinkage.
- M and ν do not operate independently: Smaller values of ν require larger values of M for the same training error.

Regularization / 2

- In general smaller values of ν give better results, but lead to a computational more demanding procedure.
- A further meta-parameter could be to use sub-sampling, such that only a certain proportion of the data is used for fitting in each step.

Interpretation

- Relative importance of predictor variables in gradient tree boosting can be obtained by

$$\mathcal{I}_l^2 = \frac{1}{M} \sum_{m=1}^{M} \mathcal{I}_l^2(T_m),$$

  with

$$\mathcal{I}_l^2(T) = \sum_{t=1}^{J-1} \hat{i}_t^2 I(v(t) = l),$$

  where $\hat{i}_t^2$ is the squared improvement in this node achieved by selecting X_l as splitting variable.
- Partial dependence plots visualize an approximation of

$$f_{\mathcal{S}}(X_{\mathcal{S}}) = \mathsf{E}_{X_{\mathcal{C}}} f(X_{\mathcal{S}}, X_{\mathcal{C}}).$$

Implementations of gradient boosting

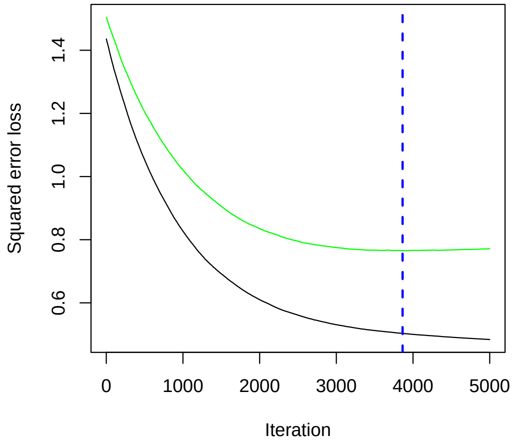
- **gbm**: gradient tree boosting.
- **mboost**: also allows for other base learners, such as linear models, etc.

Example: Prostate Cancer

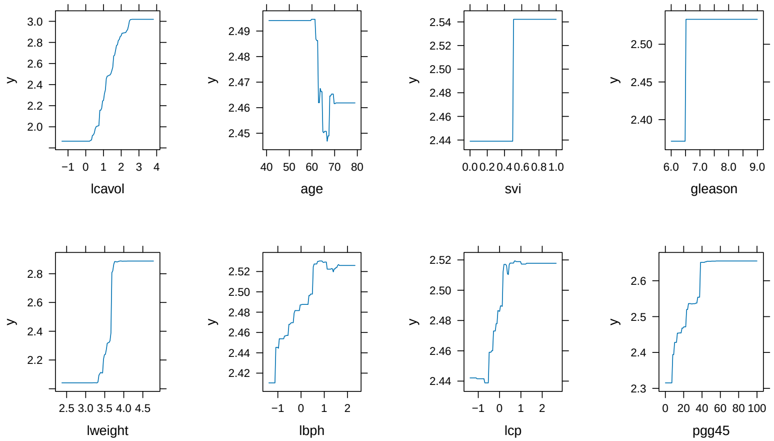
- Data set provided in the R package **ElemStatLearn**.
- Use only the 67 observations from the training data set.
- The dependent variable is `lpsa`, the level of a prostate-specific antigen.
- The independent variables are clinical measures.
- There is a substantial amount of correlation between the clinical measures.

```
> suppressMessages(library("gbm"))
> set.seed(1234)
> gbm.model <- gbm(lpsa ~ ., data = prostate,
+   distribution = "gaussian", n.trees = 5000,
+   shrinkage = 0.001, cv.folds = 10)
```

Example: Prostate Cancer / 2



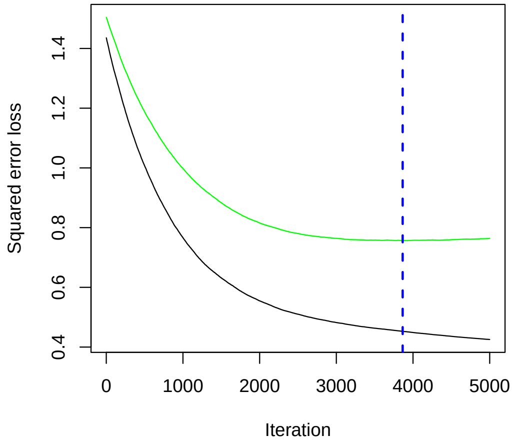
Example: Prostate Cancer / 3



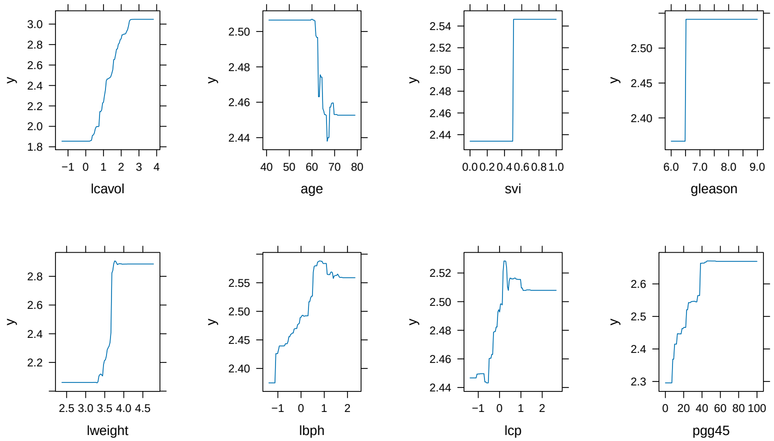
Example: Prostate Cancer / 4

```
> set.seed(1234)
> gbm.int.model <- gbm(lpsa ~ ., data = prostate,
+   distribution = "gaussian", n.trees = 5000,
+   shrinkage = 0.001, cv.folds = 10,
+   interaction.depth = 2)
```

Example: Prostate Cancer / 5



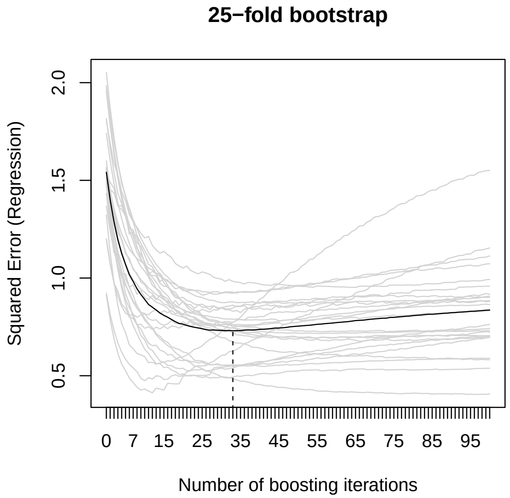
Example: Prostate Cancer / 6



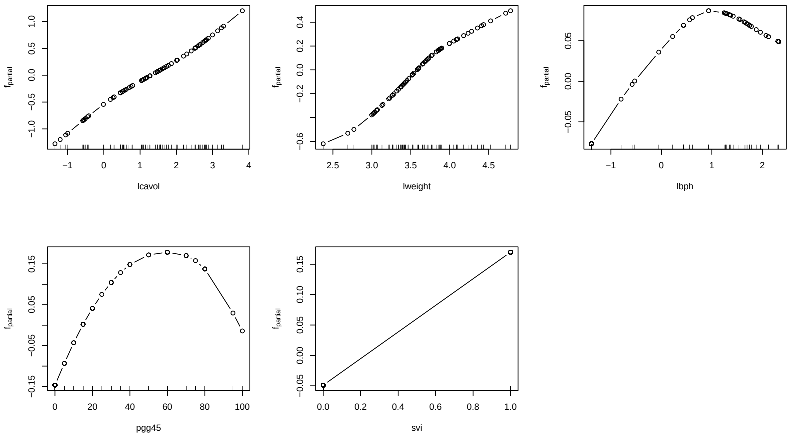
Example: Prostate Cancer / 7

```
> library("mboost")
> gam.model <- gamboost(lpsa ~ age + lcavol + lweight +
+   lbph + lcp + pgg45 + bols(svi) + bols(gleason),
+   data = prostate)
> gam.iter <- cvrisk(gam.model)
> gam.model <- gam.model[mstop(gam.iter)]
> summary(gam.model)
```

Example: Prostate Cancer / 8



Example: Prostate Cancer / 9



Example: Prostate Cancer / 10

- Prediction performance on the test dataset:

```
GBM int = 1 GBM int = 2    GAMBOOST
  0.6270044   0.6308392   0.4746177
```