Figure 7.12 – Bootstrapping

Bootstrap aggregation

With B bootstrap datasets and $\hat{f}^{*b}$ the estimated predictor based on the b'th bootstrap data set the bootstrapped, or *bagged*, estimator is

$$\hat{f}^B_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x)$$

For $B \to \infty$

$$\hat{f}^B_{\text{bag}}(x) \to E(\hat{f}^*(x)|\mathbf{Z})$$

where $\mathbf{Z}$ is the original data set and $\hat{f}^*(x)$ is a bootstrap estimated predictor.

The conditional expectation is over the random bootstrapped sampling from the data.

Bootstrapping for classification

For classification we can

- aggregate by "majority rules": Each predictor cast a vote on a class and the class with most votes is the result.
- aggregate estimated conditional class probabilities by averaging and classify to the class with the largest aggregated probability.

Majority rules can be implemented by averaging dummy variable encodings in terms of K-vectors representing the K classes. The resulting probability vector *can not* be seen as an estimate of the conditional class probabilities.

Figure 8.9 – Bootstrapped trees

Figure 8.10 – Bagging

Ensembles of Weak Predictors

A *weak predictor* is a predictor that performs only a little better than random guessing, or at least a lot worse than the Bayes classifier.

With an ensemble, or collection, of weak predictors $\hat{f}_1, \ldots, \hat{f}_B$ we seek to combine their predictions, e.g. as

$$\hat{f}^B = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b$$

hoping to improve performance.

Bootstrap aggregation or *Bagging* is an example where the ensemble of preditors is obtained by estimation of the predictor on bootstrapped data sets.

Bagging is treated in Section 8.7 in the book. Note the "weak predictor" is often taken to mean "simple predictor" where again simple refers to a predictor with few parameters, which has low variance and besides from special cases considerable bias. There is, however, nothing that prevent us from considering ensembles of weak predictors where "weak" refers to high variance as opposed to high bias.

Combining Weak Predictors

Recall that

$$V(\hat{f}^B(x)) = \frac{1}{B^2} \sum_{b=1}^{B} V(\hat{f}_b(x)) + \frac{1}{B^2} \sum_{b \neq b'} \operatorname{cov}(\hat{f}_b(x), \hat{f}'_b(x)).$$

The variance is a tradeoff between the variance of the individual predictors and their correlation.

If the weak predictors are pairwise identically distributed then

$$V(\hat{f}_b(x)) = \sigma^2(x) \quad \operatorname{cov}(\hat{f}_b(x), \hat{f}'_b(x)) = \rho(x)\sigma^2(x)$$

with $\rho(x) \geq \frac{1}{1-B}$.

$$V(\hat{f}^B(x)) = \rho(x)\sigma^2(x) + \frac{\sigma^2(x)(1-\rho(x))}{B}.$$

To show this equality use the general formula to obtain

$$V(\hat{f}^B(x)) = \frac{\sigma^2(x)}{B} + \frac{\rho(x)\sigma^2(x)B(B-1)}{B^2}.$$

Then reduce the right hand side to the equation above.

Averages of conditional i.i.d. ensembles

Assume that $\hat{f}^b = \hat{f}(\mathbf{Z}, \mathbf{I}_b)$ for i.i.d. random variables $\mathbf{I}_b$. Then

- Conditionally on $\mathbf{Z}$ the $\hat{f}^b$'s are i.i.d.
- $\hat{f}^B(x) \to \hat{f}^\infty(x) = E(\hat{f}^1(x) \mid \mathbf{Z})$ for $B \to \infty$ by LLN.
- $V(\hat{f}^\infty(x)) = \mathrm{cov}(\hat{f}^1(x), \hat{f}^2(x))$.

With $\sigma^2(x) = V(\hat{f}^1(x))$ and $\rho(x) = \mathrm{corr}(\hat{f}^1(x), \hat{f}^2(x))$

$$V(\hat{f}^\infty(x)) = \sigma^2(x)\rho(x).$$

A proof of the last bullet point is as follows

$$\begin{aligned} \mathrm{cov}(\hat{f}^1(x), \hat{f}^2(x)) &= E(\underbrace{\mathrm{cov}(\hat{f}^1(x), \hat{f}^2(X) \mid \mathbf{Z})}_{=0}) + \mathrm{cov}(E(\hat{f}^1(x) \mid \mathbf{Z}), E(\hat{f}^2(x) \mid \mathbf{Z})) \\ &= \mathrm{cov}(\hat{f}^\infty(x), \hat{f}^\infty(x)) = V(\hat{f}^\infty(x)). \end{aligned}$$

It can also be obtained, and is so in the book, by taking the limit of the variance of $\hat{f}^B$, but this is not justified by LLN alone.

Bagged predictors

Bagged predictors are examples of averages of cond. i.i.d. ensembles.

- The bagged predictor $E(\hat{f}^*(x) \mid \mathbf{Z})$ will typically *not* improve on the bias as compared to $\hat{f}(x)$ – it is on average not better than the individual bootstrapped predictor $\hat{f}^*(x)$.
- The variance of $\hat{f}^*(x)$, $\sigma^2(x)$, will typically be larger than the variance of $\hat{f}(x)$.
- If the correlation, $\rho(x)$, is sufficiently small we hope for a *variance reduction* of the bagged estimator as compared to $\hat{f}(x)$.

If we attempt to *de-correlate* the individual learners in the ensemble we generally increase the variance of the learners – we have to find a good balance.

Random Forests

Leo Breiman invented random forests to improve on bagging of trees by de-correlation.

Random forests is a bagging algorithm with the modification that for building the b'th tree the tree recursion is modified at the splitting step to

1. Sample m indices (variables) from $1, \ldots, p$.
2. Compute the optimal split among the m *sampled* variables.

With $m = 1$ the random forest is grown by optimally splitting randomly selected variables. For $m = p$ we get classical bagging for trees.

Figure 15.1

Figure 15.10

$$V(f^*(x)) \quad = \quad \underbrace{V(E(f^*(x) \mid \mathbf{Z}))}_{\sigma^2(x)\rho(x)} + E(V(f^*(x) \mid \mathbf{Z}))$$

Basis expansions, ensemble learners and trees

The *basis expansion techniques* can be seen as ensemble learning (with or without regularization) where we have specified the base learners a priori and have to learn the weights.

With simple averaging of (weak) learners we specify the weights and build the base learners adaptively to the data.

For trees we build and combine sequentially and recursively the *simplest* base learners; the stumps or single splits.

Are there general ways to search *the space of learners and combinations of simple learners*?

Stagewise Additive Modeling

With $b(\cdot, \gamma)$ for $\gamma \in \Gamma$ a parameterized family of *basis functions* we can seek expansions of the form

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

With fixed γ_m this is standard, with unrestricted γ_m this is in general very difficult numerically.

Suggetion: Evolve the expansions in *stages* where (β_m, γ_m) is estimated in step m *and then fixed forever*.

Boosting

With any loss function L the *Forward Stagewise Additive Model* is estimated by the algorithm:

1. Set $m = 1$ and initialize with $\hat{f}_0(x) = 0$.

2. Compute

$$(\hat{\beta}_m, \hat{\gamma}_m) = \underset{\beta,\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, \hat{f}_{m-1}(x_i) + \beta b(x_i, \gamma))$$

3. Set $f_m = f_{m-1} + \hat{\beta}_m b(\cdot, \hat{\gamma}_m)$, $m = m + 1$ and return to 2.

Note that with squared error loss

$$L(y_i, \hat{f}_{m-1}(x_i) + \beta b(x_i, \gamma)) = ((y_i - \hat{f}_{m-1}(x_i)) - \beta b(x_i, \gamma))^2$$

every estimation step is a *reestimation on the residuals*.

Base Classifiers

With $Y \in \{-1, 1\}$ and any classifier $G(x) \in \{-1, 1\}$ the *misclassification error* is

$$\operatorname{err}(G) = \frac{1}{N} \sum_{i=1}^{N} 1(y_i \neq G(x_i)) = \frac{1}{2N} \sum_{i=1}^{N} (1 - y_i G(x_i))$$

With $\mathcal{G}$ a *class of classifiers* the (unweigted) optimal classifier is

$$\hat{G} = \underset{G \in \mathcal{G}}{\operatorname{argmin}} \operatorname{err}(G)$$

With $w_1, \ldots, w_N \geq 0$ the weighted optimal classifier is

$$\hat{G} = \underset{G \in \mathcal{G}}{\operatorname{argmin}} \sum_{i=1}^{N} w_i 1(y_i \neq G(x_i))$$

A simple class of classifiers is the class of stumps – trees with only two leafs. A stump is given simply by a pair (i, t) of the splitting variable and the split point, and if we use the misclassification node impurity the optimization over (i, t) is precisely the optimization we carry out when we make each greedy step in the algorithm for estimation of trees.

Surrogate Loss Functions

Most important property of the surrogate loss functions is that they are convexifications of the 0-1-loss.

AdaBoost – Classification with Exponential Loss

With *exponential loss* $L(y, f(x)) = \exp(-yf(x))$ and with $w_i^{(m)} = \exp(-y_i f_{m-1}(x_i))$

$$\begin{aligned} \sum_{i=1}^{N} L(y_i, \hat{f}_{m-1}(x_i) + \beta G(x_i)) &= \sum_{i=1}^{N} w_i^{(m)} \exp(-y_i \beta G(x_i)) \\ &= (e^{\beta} - e^{-\beta}) \sum_{i=1}^{N} w_i^{(m)} 1(y_i \neq G(x_i)) + e^{-\beta} \sum_{i=1}^{N} w_i^{(m)} \end{aligned}$$

The minimizer is $\hat{G}_m = \text{argmin}_{G \in \mathcal{G}} \sum_{i=1}^N w_i^{(m)} 1(y_i \neq G(x_i))$,

$$\hat{\beta}_m = \frac{1}{2} \log \frac{1 - \text{err}_m}{\text{err}_m}$$

The updated *weights* in step $m+1$ are

$$\begin{aligned} w_i^{(m+1)} &= w_i^{(m)} \exp(-y_i \hat{\beta}_m \hat{G}_m(x_i)) \\ &= w_i^{(m)} \exp(2\hat{\beta}_m 1(y_i \neq \hat{G}_m(x_i))) \exp(-\hat{\beta}_m). \end{aligned}$$

The weights in the AdaBoost algorithm evolve over time multiplying larger weights in the m'th step on those pairs (x_i, y_i) that are misclassified by the classifier in the m'th step.

Figure 10.1 – Schematic AdaBoost

$$G(x) = \sum_{m=1}^M \alpha_m G_m(x)$$

1. Initialize with weights $w_i = 1/N$ and set $m = 1$ and fix M.
2. Fit a classifier G_m using weights w_i.
3. Recompute weights as

   $$w_i \leftarrow w_i \exp(\alpha_m 1(y_i \neq G_m(x_i)))$$

   where $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$ and

   $$\text{err}_m = \frac{1}{\sum_{i=1}^N w_i} \sum_{i=1}^N w_i 1(y_i \neq G(x_i)).$$

4. Stop if $m = M$ or set $m \to m + 1$ and return to 2

Figure 10.2 and 10.3

Boosting using stumps only can outperform even large trees in terms of test error (simulation).

Even when the misclassification error is 0 on the training data it can pay to continue the boosting and the exponential loss will continue to decrease.

More Boosting

The computational problem in boosting is minimization of

$$\sum_{i=1}^{N} L(y_i, \hat{f}_{m-1}(x_i) + \beta b(x_i, \gamma)).$$

For classification with *exponential loss* this simplifies to *weighted* optimal classification.

For regression and *squared error loss* this is re-estimation based on the residuals.

With the notation

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

for $\mathbf{f} = (f_1, \ldots, f_N) \in \mathbb{R}^N$ we aim at finding *steps* $\mathbf{h}_1, \ldots, \mathbf{h}_M$ and with $\mathbf{h}_0$ the initial guess an approximate minimizer of the form

$$\mathbf{f}_M = \sum_{m=0}^{M} \mathbf{h}_m.$$

Gradient Boosting

The gradient of $L : \mathbb{R}^N \to \mathbb{R}$ is

$$\nabla L(\mathbf{f}) = (\partial_z L(y_1, f_1), \ldots, \partial_z L(y_N, f_N))^T$$

Gradient descent algorithms suggest steps from $\mathbf{f}_m$ in the direction of $-\nabla L(\mathbf{f}_m)$;

$$\mathbf{h}_m = -\rho_m \nabla L(\mathbf{f}_m).$$

Problem: $-\rho_m \nabla L(\mathbf{f}_m)$ is most likely *not* obtainable as a prediction within the class of *base learners* – it is *not* of the form $\beta(b(x_1, \gamma), \ldots, b(x_N, \gamma))^T$.

Solution: Fit a *base learner* $\hat{\mathbf{h}}_m$ to $-\nabla L(\mathbf{f}_m)$ and compute by iteration the expansion

$$\hat{f}_M = \sum_{m=0}^{M} \rho_m \hat{h}_m.$$

This is *gradient boosting* as implemented in the `mboost` library.

More information on gradient boosting and the `mboost` R-package can be found in: Peter Bühlmann and Torsten Hothorn. *Boosting Algorithms: Regularization, Prediction and Model Fitting.* Statistical Science 2007, Vol. 22, No. 4, 477-505. There is also an interesting discussion following the paper.