Ensemble Learning

Aldo Solari

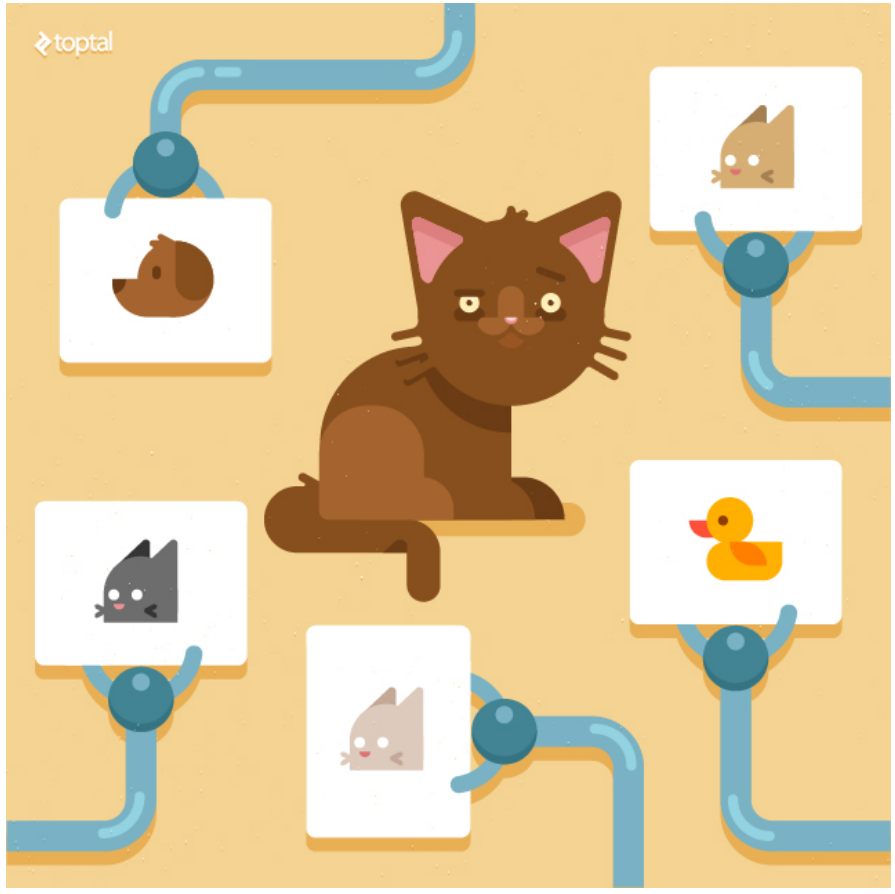
Ensembles

- Ensemble methods are techniques that create multiple models and then combine them to produce improved results
- These models, when used as inputs of ensemble methods, are often called "base learners" or "weak learners"
- Ensemble learning is appealing because that it is able to boost weak learners which are slightly better than random guess to strong learners which can make very accurate predictions
- However, ensemble learning increases computation time and reduces interpretability

Majority voting

- Majority voting is a simple ensemble method used for classification
- Each classification model in the ensemble makes a prediction (vote) for each test observation and the final prediction is the one that receives more than half of the votes
- If none of the predictions get more than half of the votes, we may say that the ensemble method could not make a stable prediction for this observation

Majority voting



Majority voting

- Suppose that we have three independent classifiers
- Assume that each classifier has probability π of making the correct classification
- The number of correct classifications is

$$C \sim \text{Binomial}(3, \pi)$$

- The majority vote classifier makes the correct prediction when $C \geq 2$, that is, with probability

$$\Pr(C \geq 2) = 3\pi^2(1 - \pi) + \pi^3$$

- e.g. if $\pi = 70\%$, then the majority vote classifier makes the correct prediction with probability 78.4%

Ensemble of trees

- Classification and regression trees are simple and useful for interpretation
- However they are typically not competitive with other approaches in terms of prediction accuracy
- Ensemble methods such as bagging, random forests and boosting grow multiple trees which are then combined to yield a single prediction
- Combining a large number of trees can often result in dramatic improvements in prediction accuracy, at the expense of some interpretation loss

Instability of trees

- The primary disadvantage of trees is that they are rather unstable (high variance)
- In other words, a small change in the data often results in a completely different tree
- One major reason for this instability is that if a split changes, all the splits under it change as well, thereby propagating the variability
- Leo Breiman's idea: use the instability!

Outline

Bagging

- Bootstrap aggregation, or bagging, is a general procedure for reducing the variance of a model and it is particularly useful in the case of regression or classification trees
- Recall that given a set of independent variables $Z_1, \ldots, Z_B$, each with variance σ^2, the variance of the average

$$\bar{Z} = \frac{1}{B} \sum_{b=1}^{B} Z_b$$

  is σ^2/B
- In other words, averaging a set of variables reduces variance. Of course, this is not practical because we generally do not have multiple training sets
- Instead, we can bootstrap, by taking repeated samples from the (single) training set

Bagging algorithm

- First, generate B different bootstrapped training sets

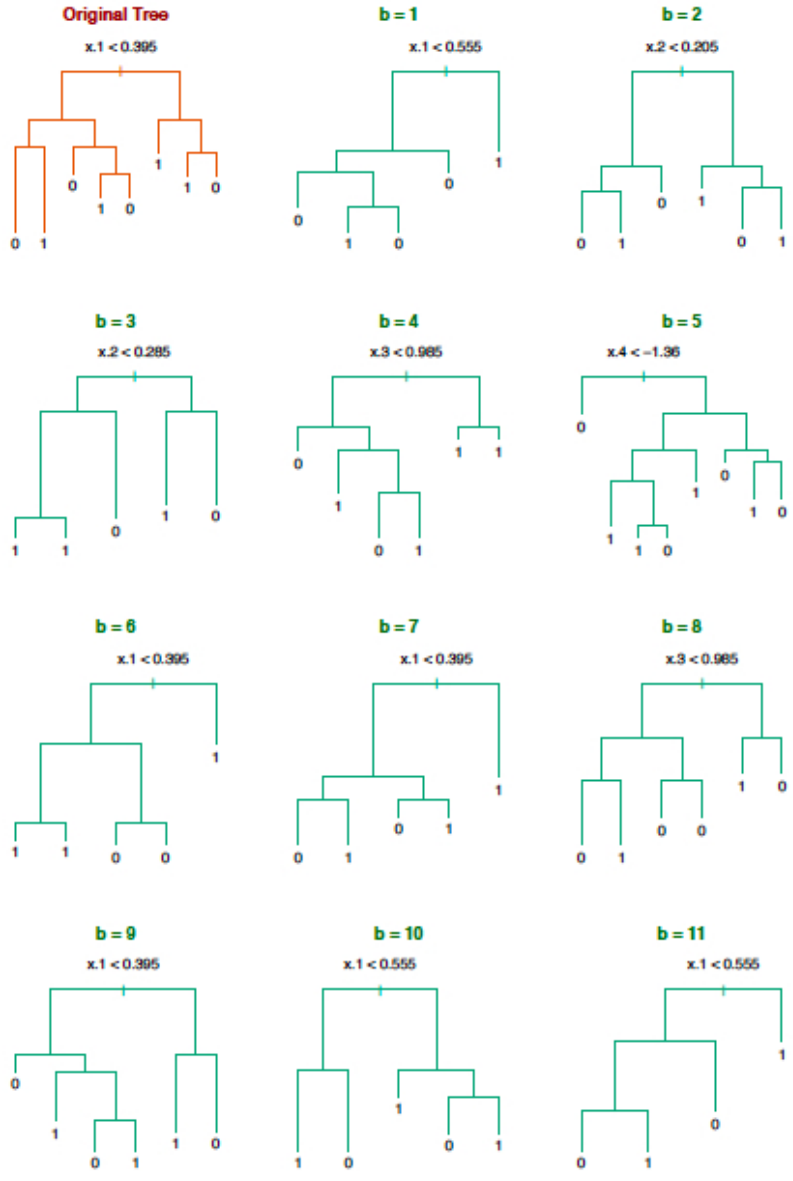
$$(x_1^{*b}, y_1^{*b}), (x_2^{*b}, y_2^{*b}), \ldots, (x_n^{*b}, y_n^{*b}) \quad b = 1, \ldots, B$$

  by sampling with replacement from the training set
- Then, fit a tree on the bth bootstrapped training set in order to get $\hat{f}^{*b}(x)$, the prediction at a point x
- Finally, average all the predictions to obtain

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

- For classification trees, we can record the class $\hat{C}^{*b}(x)$ predicted by each of the B trees, and take a majority vote

Many trees

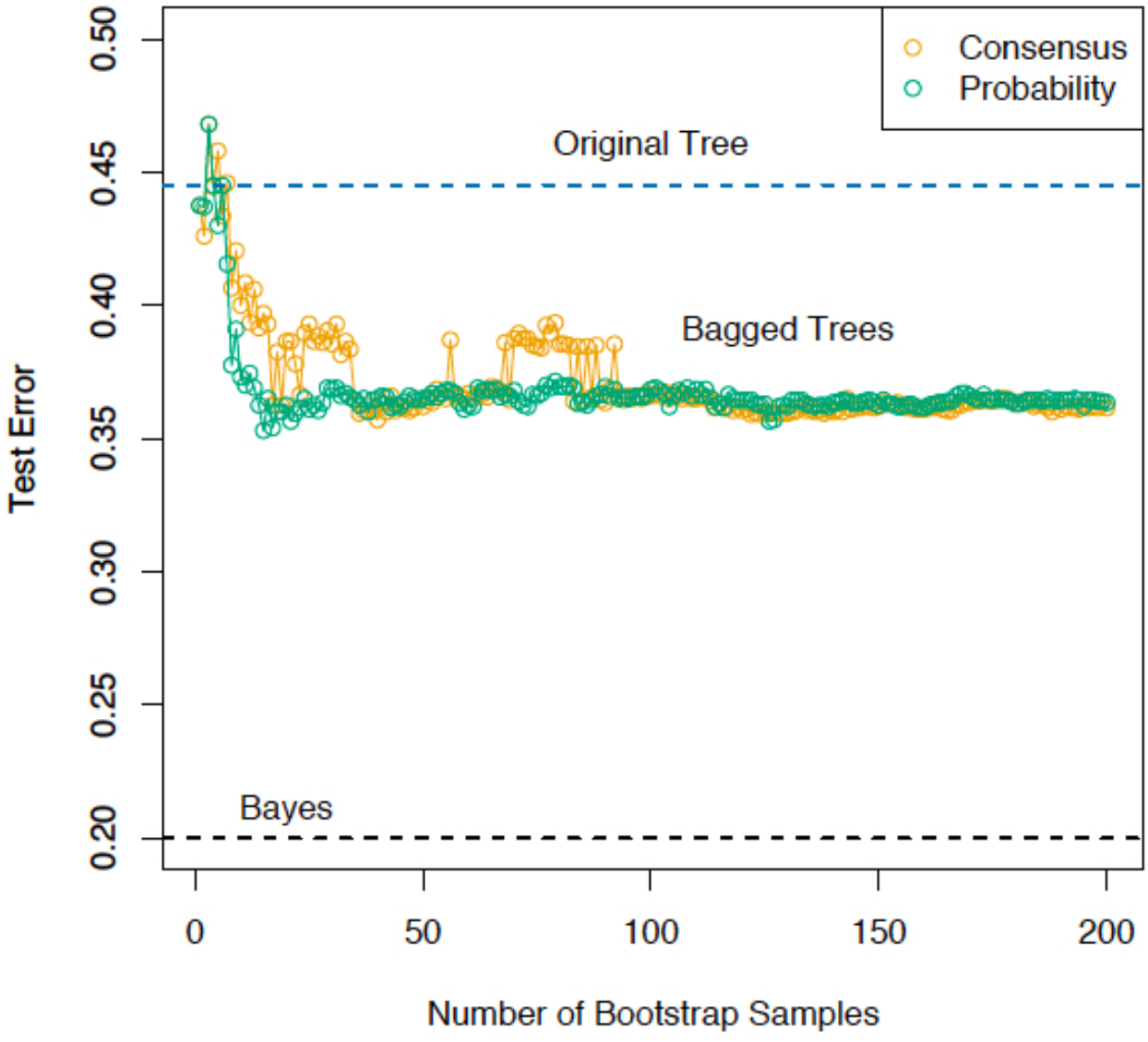


Source: Hastie et al. (2009) p. 284

`spam` data

- 4601 email messages sent to "George" at HP-Labs
- The goal is to build a customized spam filter for George: predict whether an e-mail message is spam (junk mail) or good
- For this problem not all errors are equal; we want to avoid filtering out good e-mail, while letting spam get through is not desirable but less serious in its consequences
- Recorded for each email message is the relative frequency of certain key words (e.g. `business`, `address`, `free`, `George`) and certain characters: (, [, !, $, #. Included as well are three different recordings of capitalized letters.

Bagged trees



Source: Hastie et al. (2009) p. 285

Out-of-bag observations

- In the bth bootstrapped training set

$$(x_1^{*b}, y_1^{*b}), (x_2^{*b}, y_2^{*b}), \ldots, (x_n^{*b}, y_n^{*b})$$

  the probability for one observation not to be drawn in one draw

$$1 - \frac{1}{n}$$

- the probability for one observation not to be drawn in any one of the n draws can be approximated by

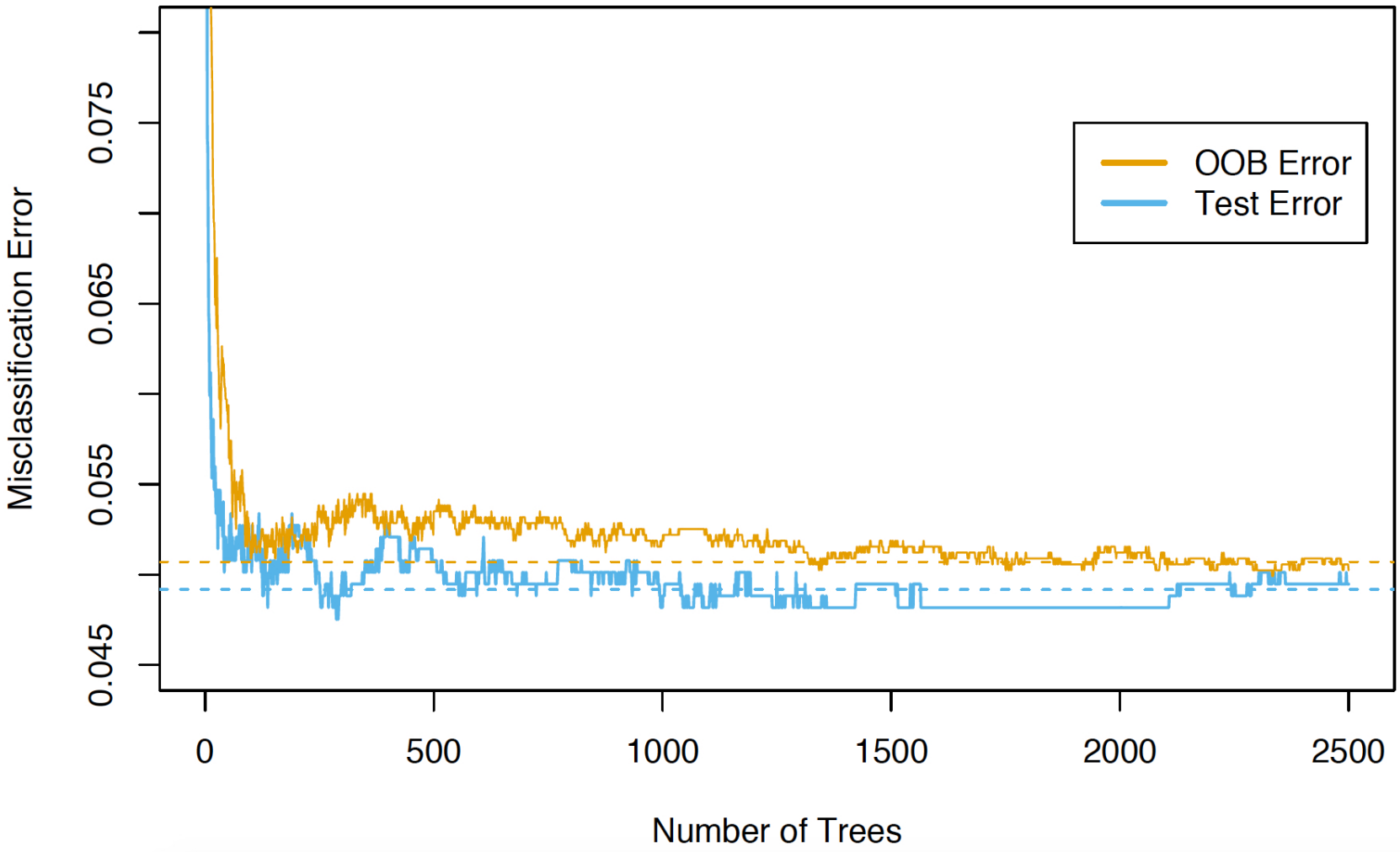
$$\lim_{n \to \infty} \left(1 - \frac{1}{n}\right)^n = \frac{1}{e} \approx 0.368$$

- $\approx 1/3$ of the n original observations are out-of-bag (OOB)

Out-of-bag error

- Each bagged tree makes use of $\approx 2/3$ of the original observations
- We can predict the response for the ith observation using each of the bagged trees in which that observation was OOB
- This yields $\approx B/3$ predictions for the ith observation, which we average/majority vote
- This estimate is essentially the LOOCV error for bagging, if B is large

`spam` data: OOB error



Source: Hastie et al. (2009) p. 592

Outline

Random forests

- Create even more variation in individual trees
- Bagging varies the rows of the training set (randomly draw observations)
- Random forests varies also the columns of the training set (randomly draw predictors)

Random forests algorithm

- Before each split, select $m \leq p$ of the predictors at random as candidates for splitting
- $m = p$ gives Bagging as a special case
- m is called a tuning parameter. Typically $m \approx \sqrt{p}$

De-correlating trees

- Random sampling of the predictors decorrelates the trees. This reduces the variance when we average the trees
- Recall that given a set of identical distributed (but not necessarily independent) variables $Z_1, \ldots, Z_B$ with pairwise correlation $\mathbb{C}\mathrm{orr}(Z_j, Z_l) = \rho$, mean $\mathbb{E}(Z_j) = \mu$ and variance $\mathbb{V}\mathrm{ar}(Z_j) = \sigma^2$, then

$$\mathbb{V}\mathrm{ar}(\bar{Z}) = \rho\sigma^2 + \frac{(1-\rho)}{B}\sigma^2 \tag{1}$$

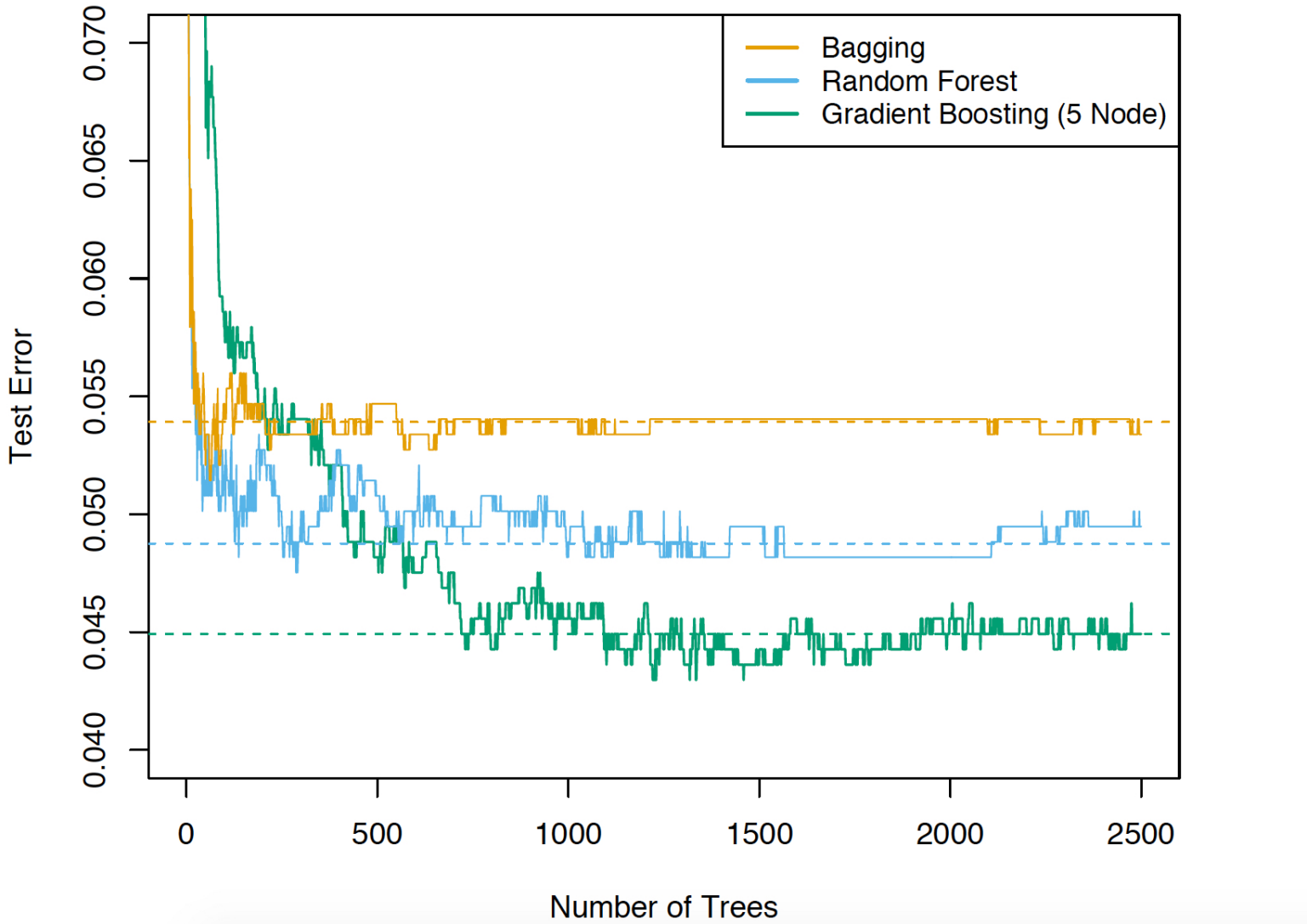
- The idea in random forests is to improve the variance reduction of bagging by reducing the correlation ρ between the trees, without increasing the variance σ^2 too much

Proof of (1)

- $\rho = \frac{1}{\sigma^2}[\mathbb{E}(Z_i Z_j) - \mathbb{E}(Z_i)\mathbb{E}(Z_j)]$
- $\mathbb{E}(Z_i Z_j) = \rho\sigma^2 + \mu^2$ if $i \neq j$
- $\mathbb{E}(Z_i^2) = \sigma^2 + \mu^2$
- $\mathbb{E}[(\sum_{j=1}^{B} Z_j)^2] = \sum_{i=1}^{B}\sum_{j=1}^{B} \mathbb{E}(Z_i Z_j) = B\mathbb{E}(Z_i^2) + (B^2 - B)\mathbb{E}(Z_i Z_j)$
- $\mathbb{E}(\sum_{j=1}^{B} Z_j) = \sum_{j=1}^{B} \mathbb{E}(Z_j) = B\mu$

$$
\begin{aligned}
\mathbb{V}\mathrm{ar}(\bar{Z}) &= \frac{1}{B^2}\mathbb{V}\mathrm{ar}(\sum_{j=1}^{B} Z_j) \\
&= \frac{1}{B^2}\{\mathbb{E}[(\sum_{j=1}^{B} Z_j)^2] - [\mathbb{E}(\sum_{j=1}^{B} Z_j)]^2\}
\end{aligned}
$$

`spam` data: random forest



Source: Hastie et al. (2009) p. 589

Variable importance

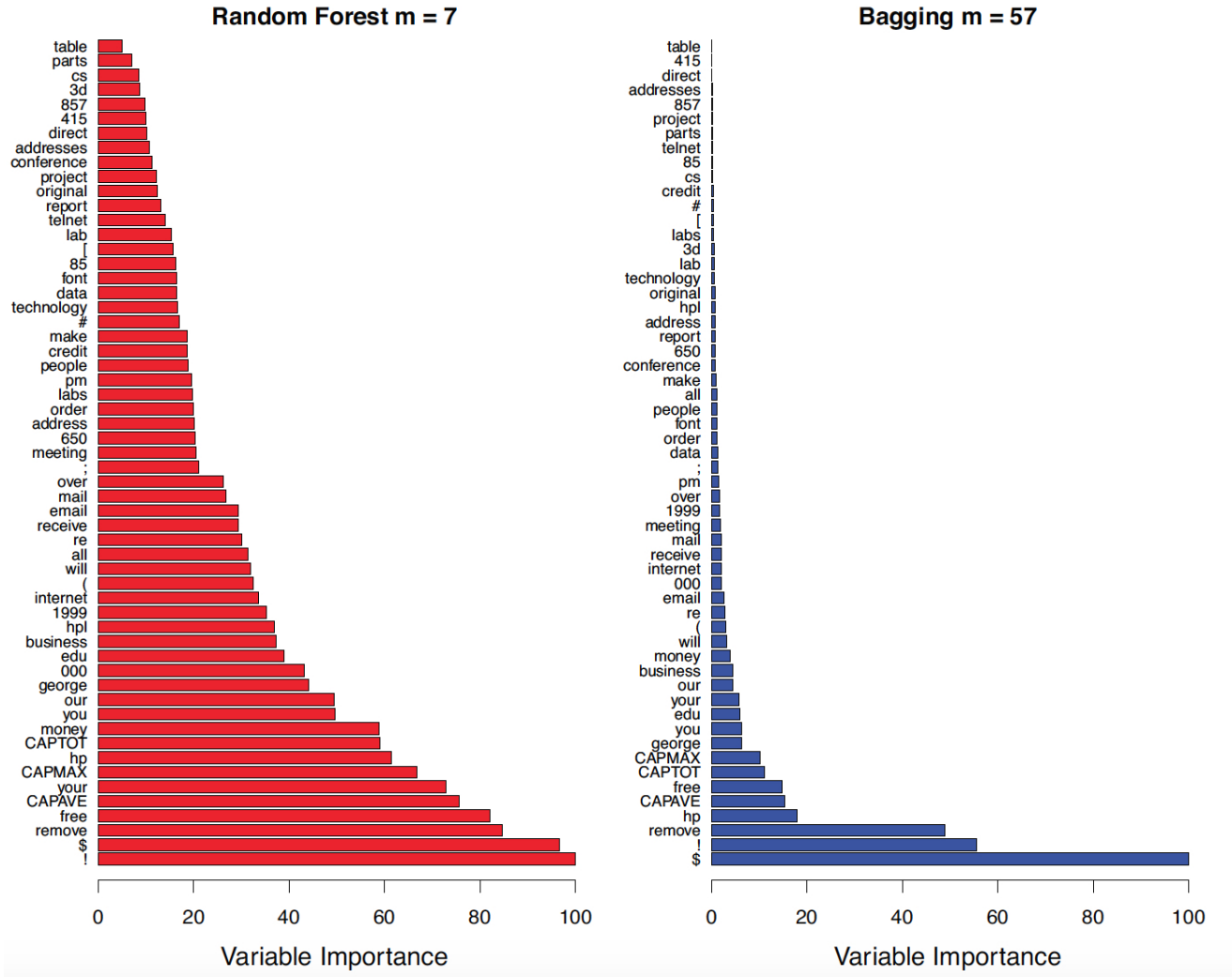
- We can calculate the importance of a predictor X_j
- The function `importance` of the R package `randomForests` with argument `type=1` :
- For each tree, record the accuracy on the OOB observations
- Do the same is done but with X_j values randomly permuted in the OOB observations
- Compute the decrease in each tree's accuracy
- If the average decrease over all the trees is large, then the predictor is considered important - its value makes a big difference in predicting the response
- If the average decrease is small, then the predictor doesn't make much difference to the response

randomForest()

```
fit <- randomForest(y ~ .,
            data = train,
            ntree = B,
            mtry = m, # default sqrt(p) or p/3
            nodesize = ns, # default 1 or 5
            importance=TRUE) # default: F
```

- In bagging and random forests trees are grown large without pruning, only the minimum number of observations per node - `nodesize` - is fixed $= 1$ for classification and $= 5$ for regression
- Number of randomly selected predictors `mtry` $= \sqrt{p}$ for classification and $= p/3$ for regression. Note that `mtry`$= p$ is bagging

spam data: variable importance



Source: Efron and Hastie (2016) p. 332

Bagging and random forest takeaways

- Bagging stabilizes decision trees and improves accuracy by reducing variance
- Random forests further improve decision tree performance by de-correlating the individual trees in the bagging ensemble
- Random forests' variable importance measures can help you determine which variables are contributing the most strongly to your model
- Because the trees in a random forest ensemble are unpruned and potentially quite deep, there's still a danger of overfitting

Outline

Boosting

- Boosting starts by fitting a base learner to the training data
- Next the base learner is re-fitted, but with more weight (importance) given to badly fitted/misclassified observations
- This process is repeated until some stopping rule is reached
- We discuss the following two algorithms
  - boosting with squared error loss (L2-boosting) by using regression trees as weak learners
  - boosting with exponential loss (AdaBoost.M1) by using classification trees as weak learners
- For a more general formulation (gradient boosting), see EH, p. 341 Algorithm 17.4

L2-boosting algorithm for regression trees

1. Initialize $\hat{f}(x) = \bar{y}$ and $r_i = y_i - \bar{y}$ for $i = 1, \ldots, n$

2. For $b = 1, 2, \ldots, B$, repeat:

(a) Fit a tree $\hat{f}^b$ with d splits to the data $(x_1, r_1), \ldots, (x_n, r_n)$

(b) Update $\hat{f}$ by adding in a shrunken version of the new tree:

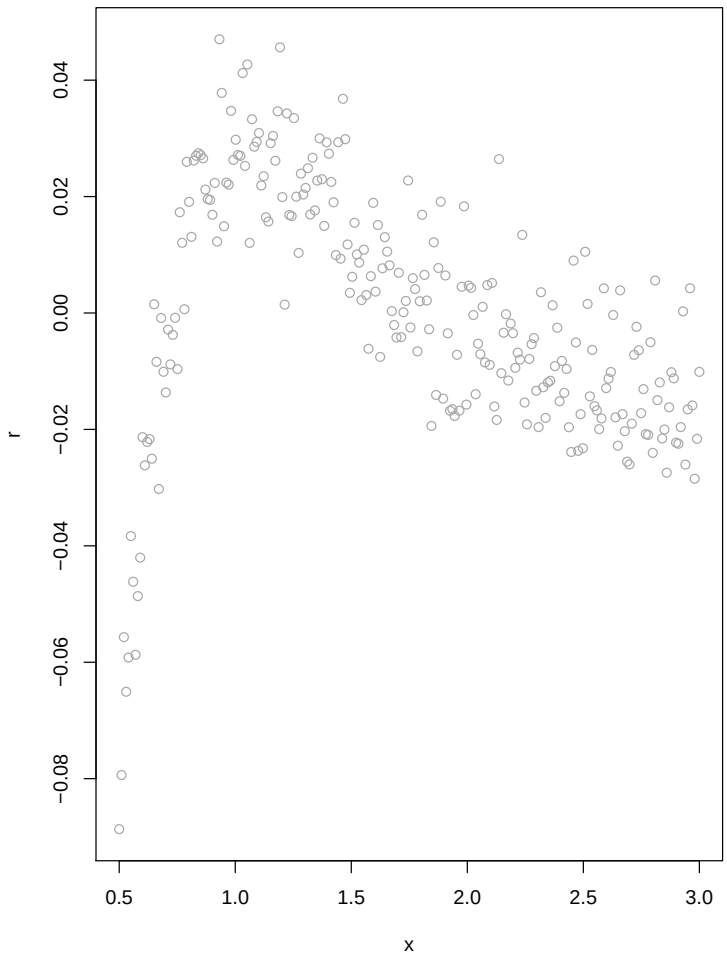
$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$$

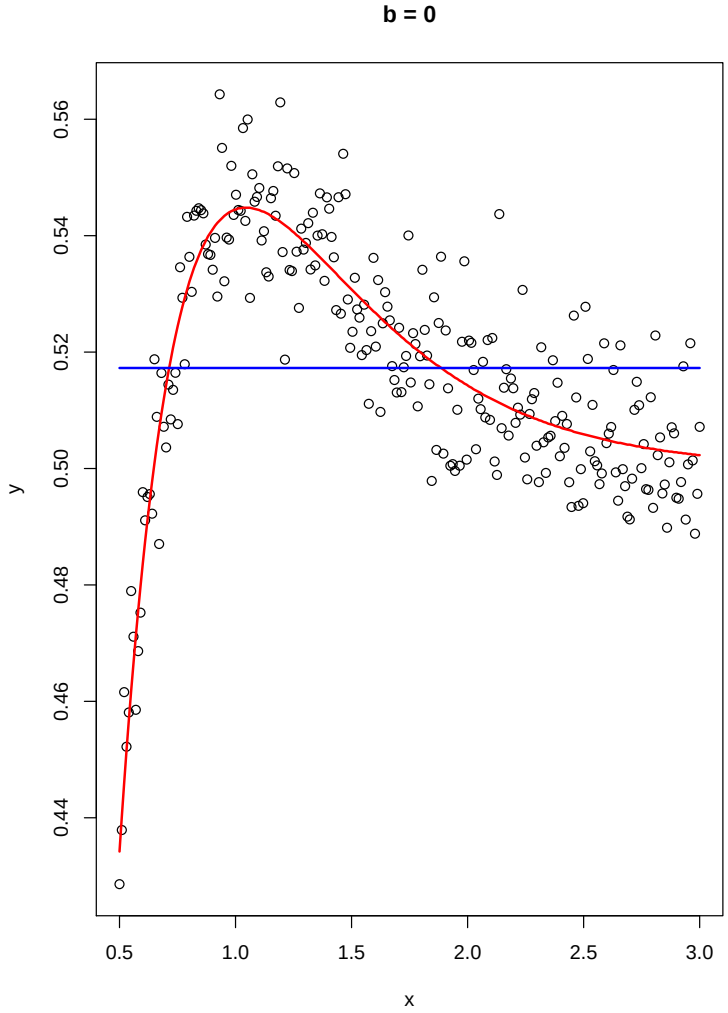
(c) Update the residuals:

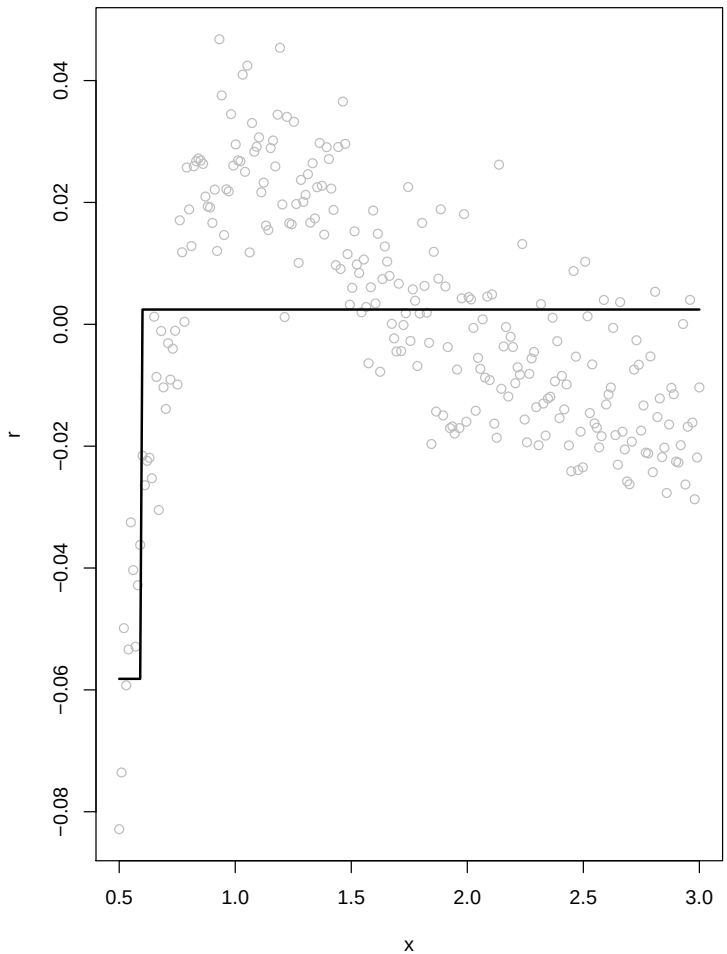
$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$

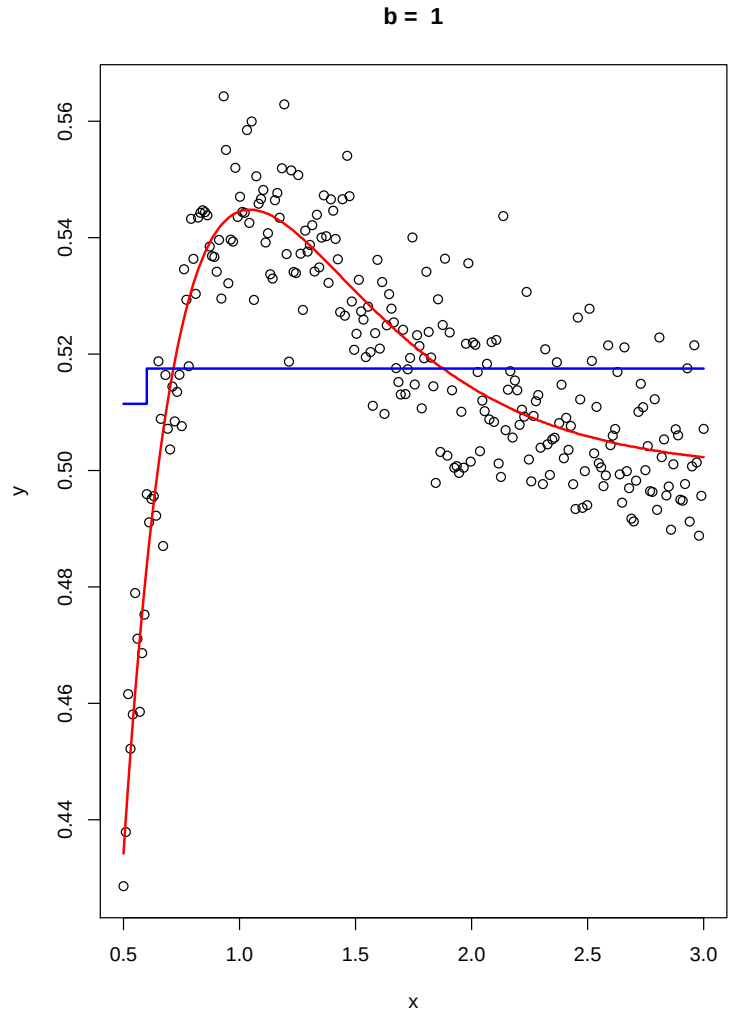
3. Output the boosted model:

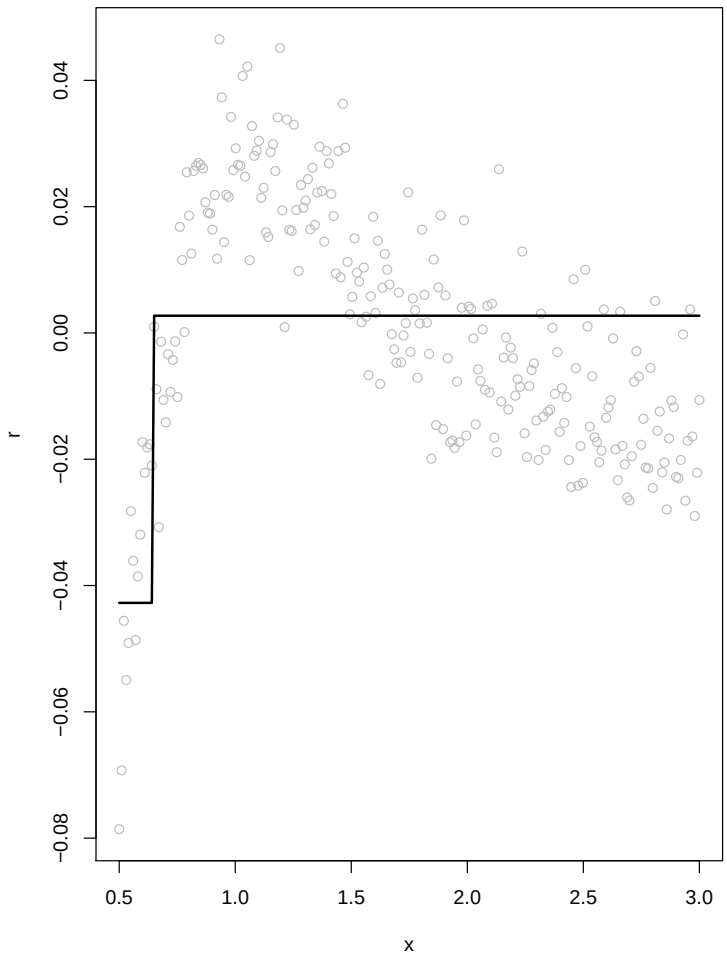
$$\hat{f}(x) = \bar{y} + \sum_{b=1}^{B} \lambda \hat{f}^b(x)$$

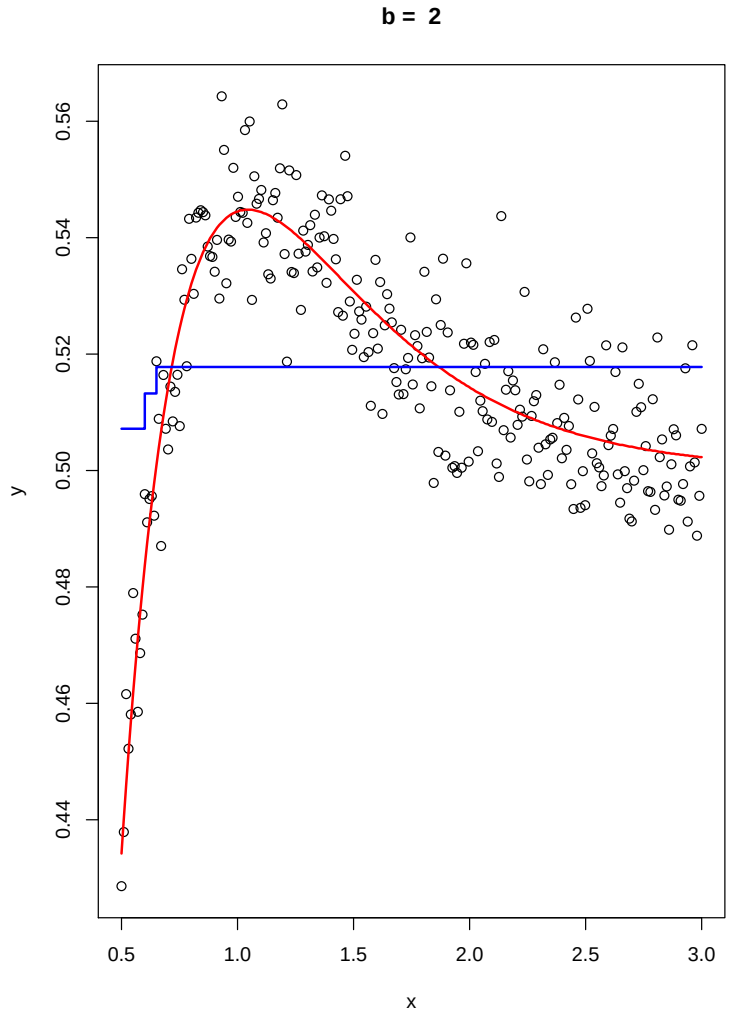

r
x
0.5
1.0
1.5
2.0
2.5
3.0
0.04
0.02
0.00
−0.02
−0.04
−0.06
−0.08

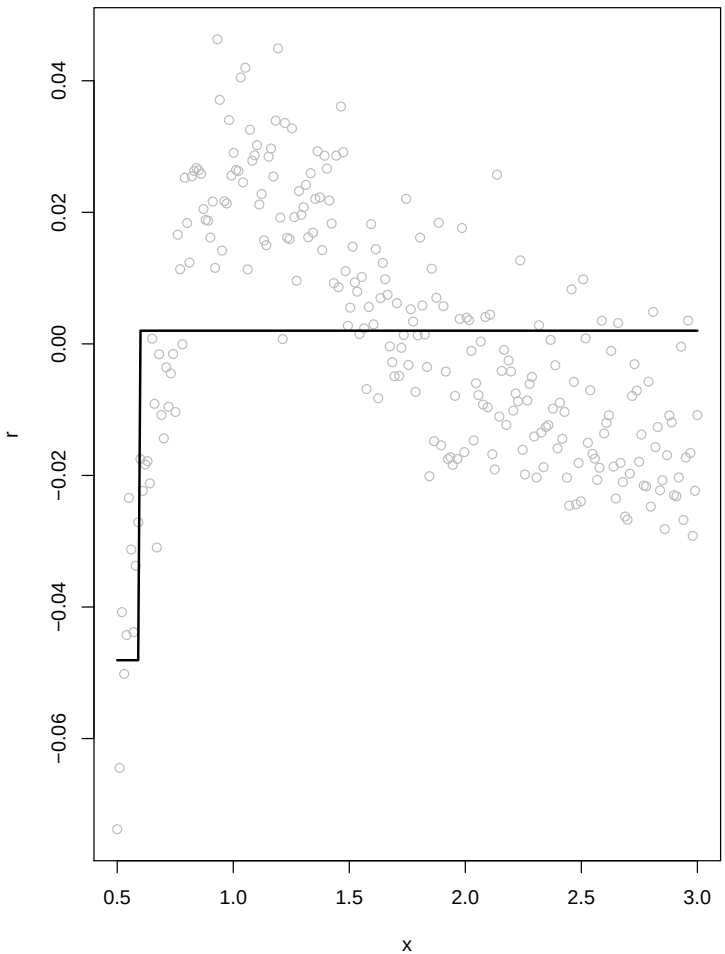

b = 0
y
x
0.5
1.0
1.5
2.0
2.5
3.0
0.56
0.54
0.52
0.50
0.48
0.46
0.44

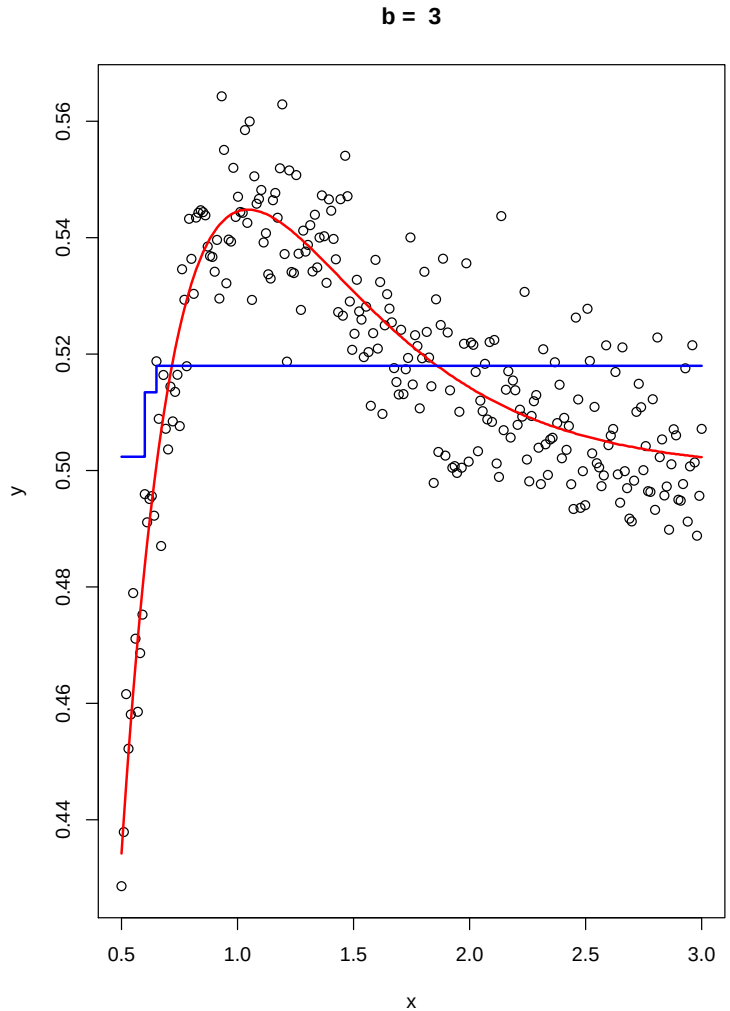

r
x
0.04
0.02
0.00
−0.02
−0.04
−0.06
−0.08
0.5
1.0
1.5
2.0
2.5
3.0

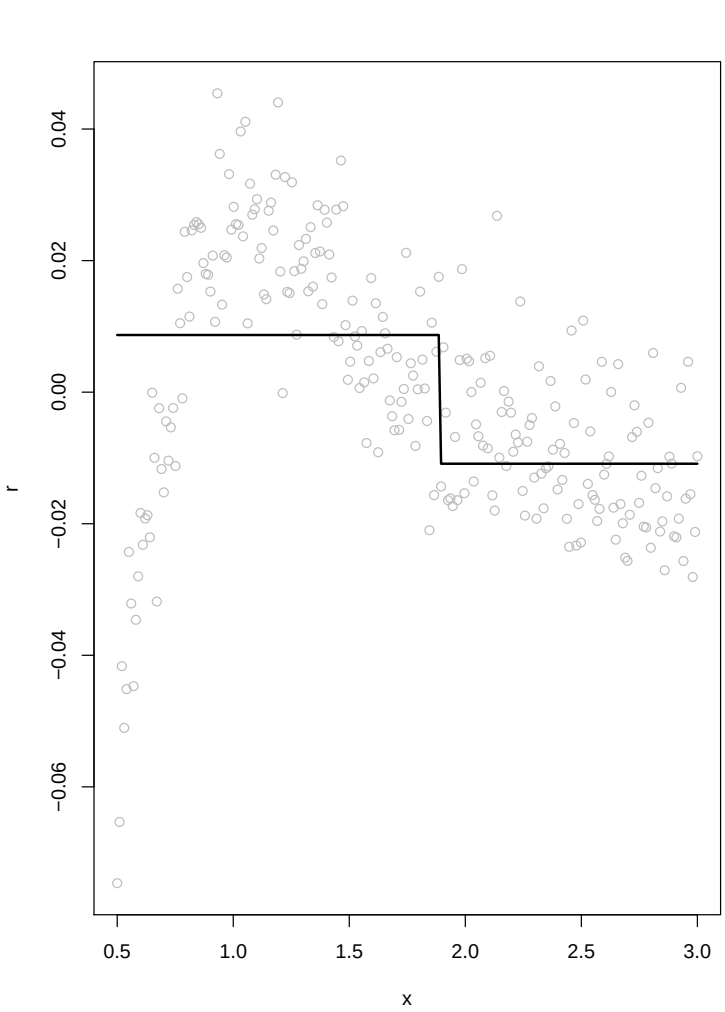

b = 1
y
x
0.56
0.54
0.52
0.50
0.48
0.46
0.44
0.5
1.0
1.5
2.0
2.5
3.0

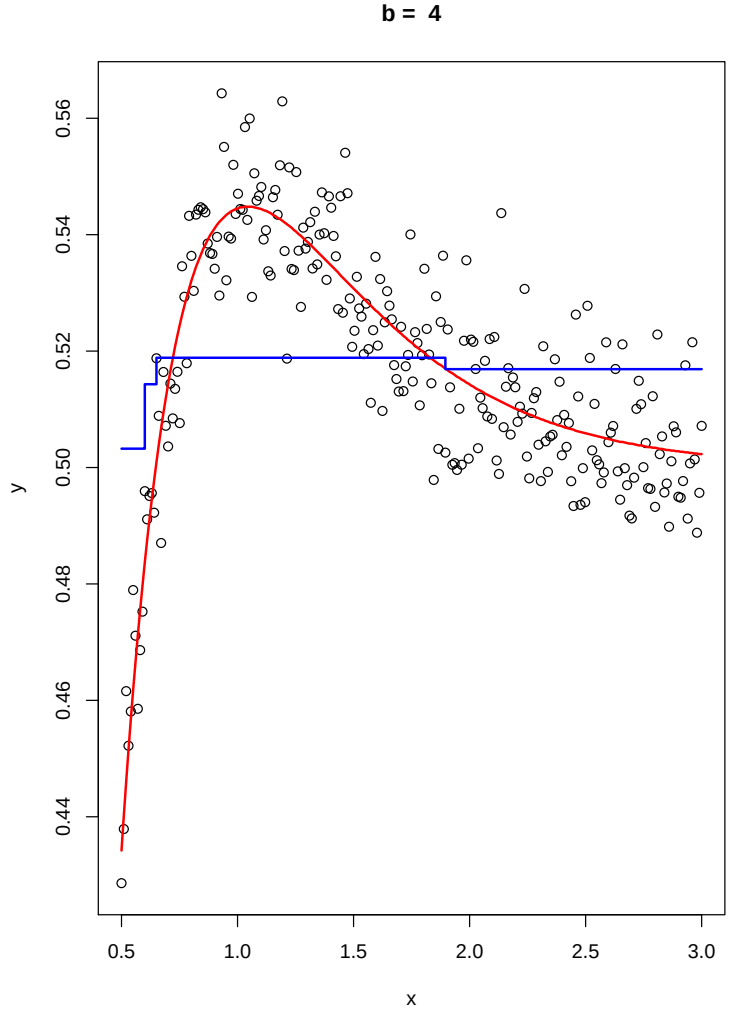

0.04
0.02
0.00
−0.02
−0.04
−0.06
−0.08
r
0.5
1.0
1.5
2.0
2.5
3.0
x

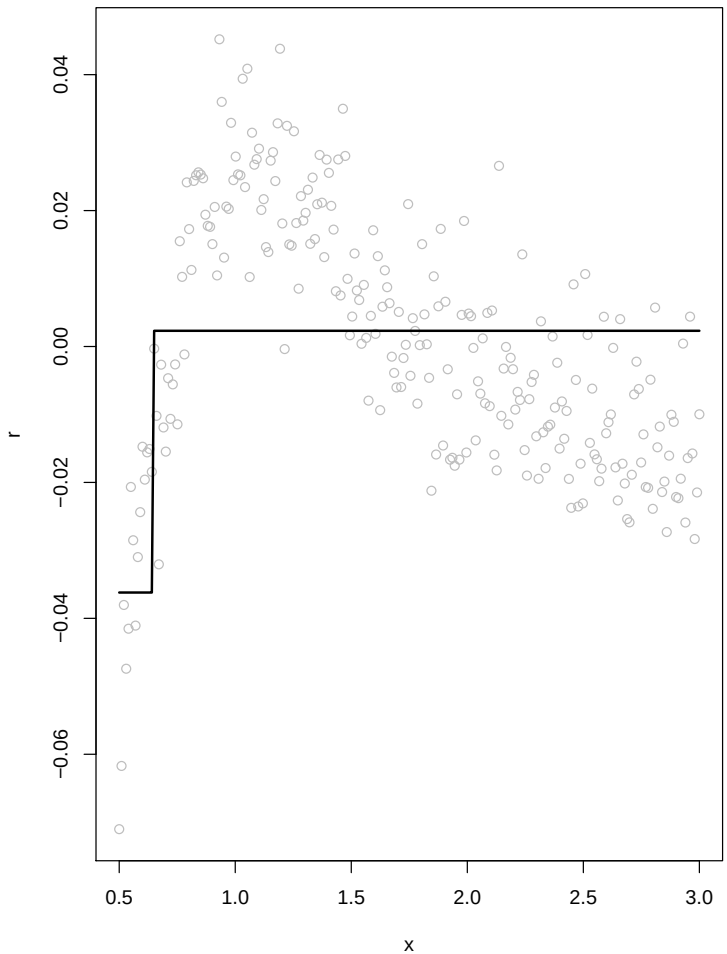

b = 2
0.56
0.54
0.52
0.50
0.48
0.46
0.44
y
0.5
1.0
1.5
2.0
2.5
3.0
x

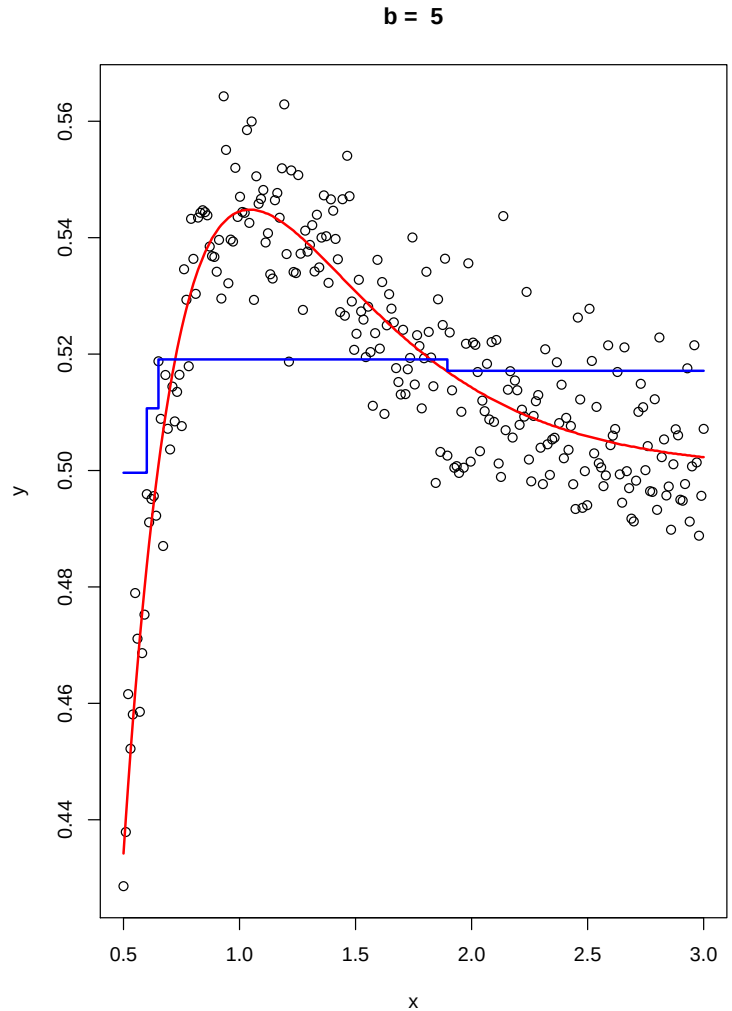

r
x
0.04
0.02
0.00
−0.02
−0.04
−0.06
0.5
1.0
1.5
2.0
2.5
3.0

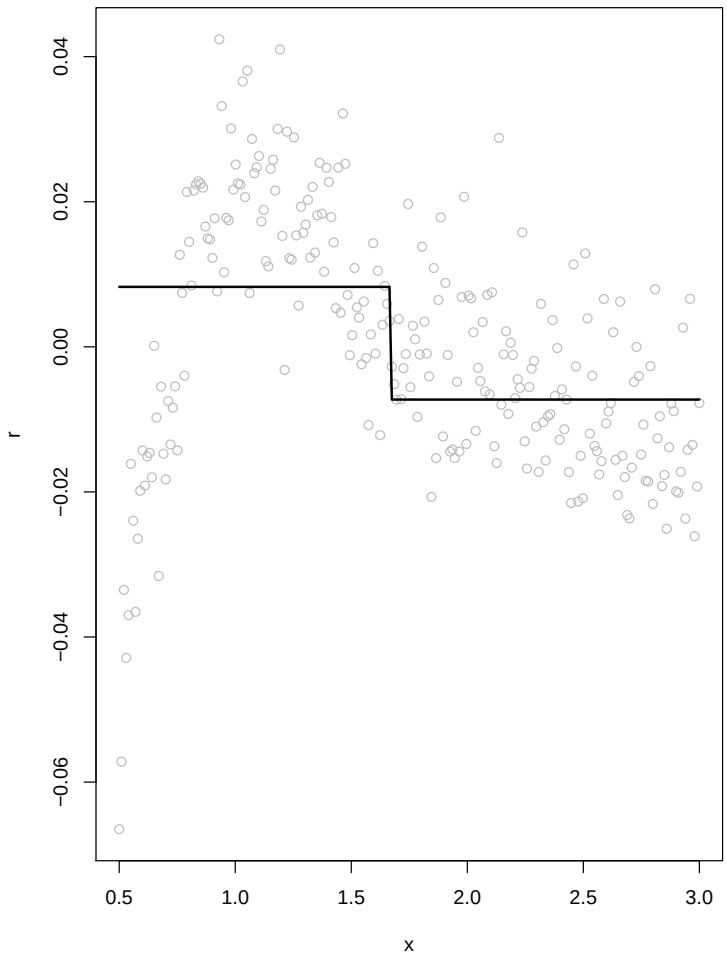

b = 3
y
x
0.56
0.54
0.52
0.50
0.48
0.46
0.44
0.5
1.0
1.5
2.0
2.5
3.0

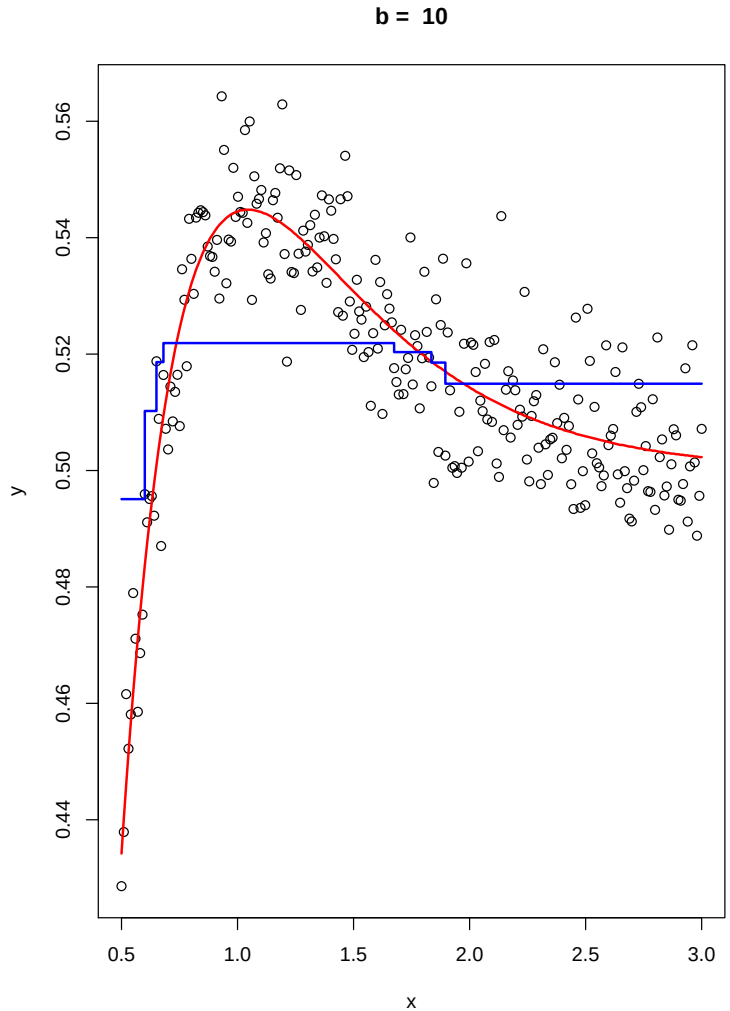

r
x
0.04
0.02
0.00
−0.02
−0.04
−0.06
0.5
1.0
1.5
2.0
2.5
3.0

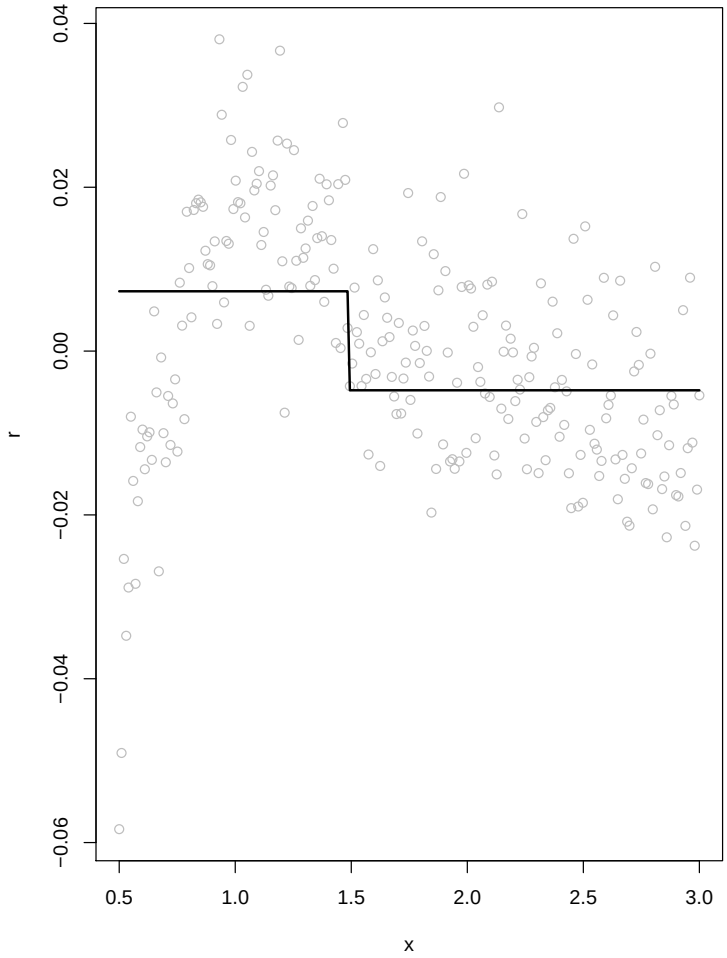

b = 4
y
x
0.56
0.54
0.52
0.50
0.48
0.46
0.44
0.5
1.0
1.5
2.0
2.5
3.0

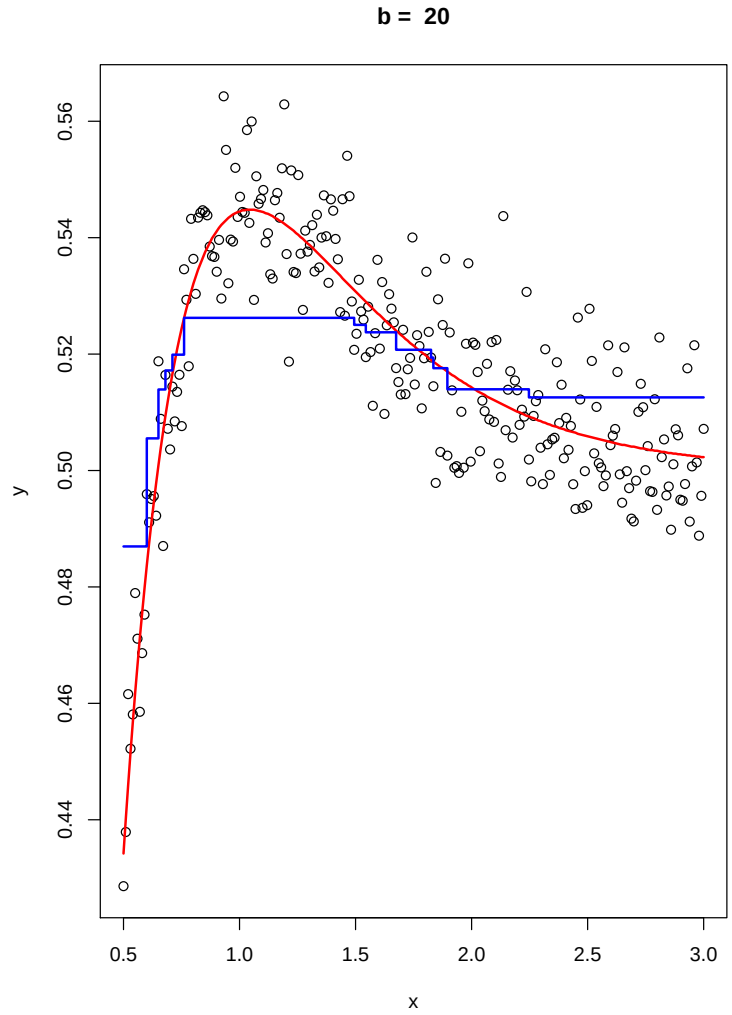

r
x
0.04
0.02
0.00
−0.02
−0.04
−0.06
0.5
1.0
1.5
2.0
2.5
3.0

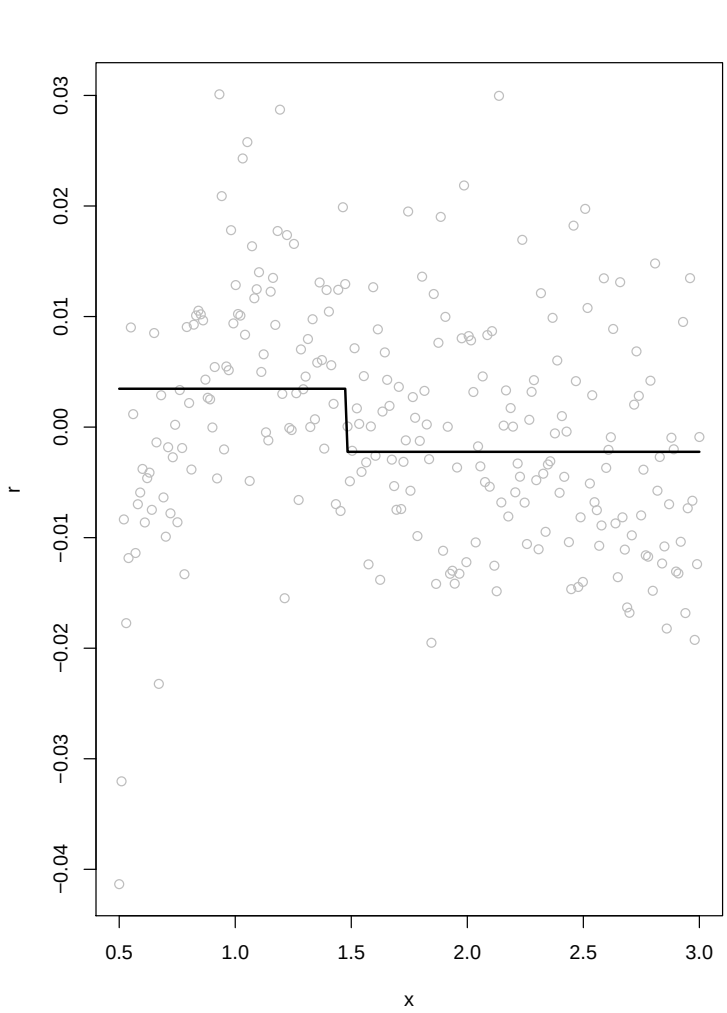

b = 5
y
x
0.56
0.54
0.52
0.50
0.48
0.46
0.44
0.5
1.0
1.5
2.0
2.5
3.0

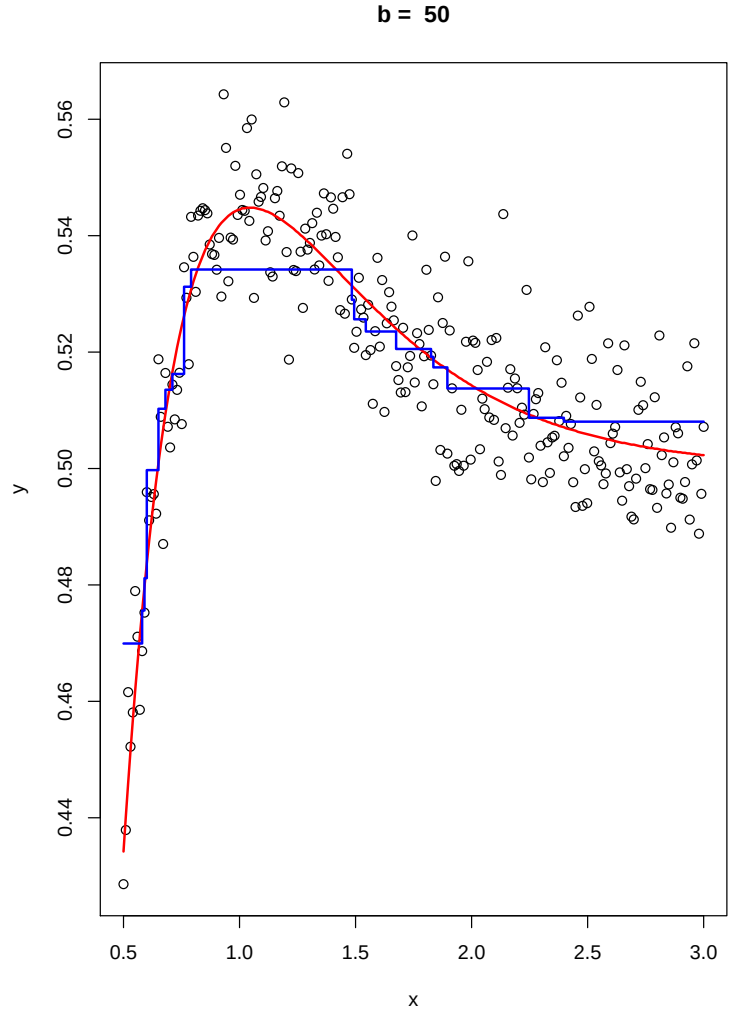

0.04
0.02
0.00
−0.02
−0.04
−0.06
r
0.5
1.0
1.5
2.0
2.5
3.0
x

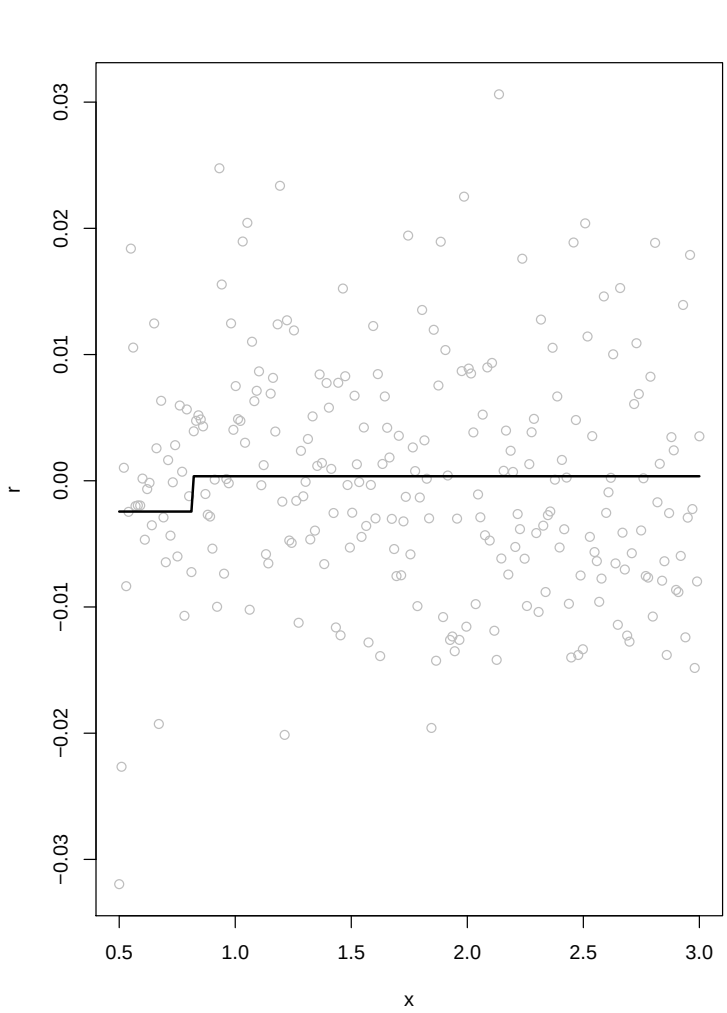

b = 10
0.56
0.54
0.52
0.50
0.48
0.46
0.44
y
0.5
1.0
1.5
2.0
2.5
3.0
x

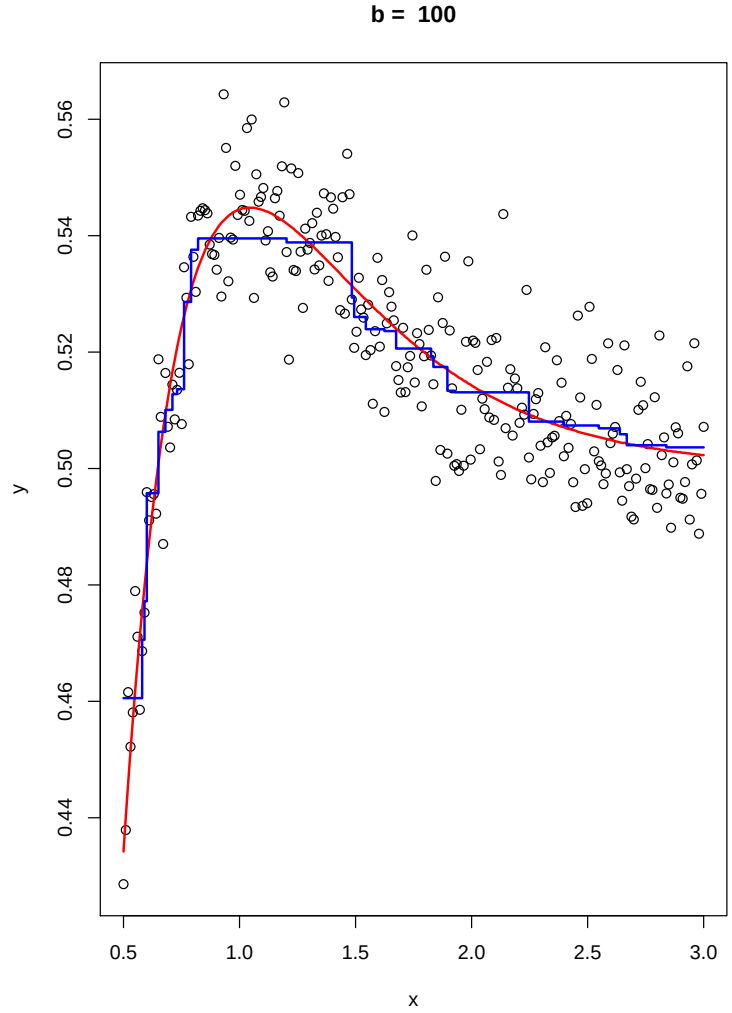

r
x
0.04
0.02
0.00
−0.02
−0.04
−0.06
0.5
1.0
1.5
2.0
2.5
3.0

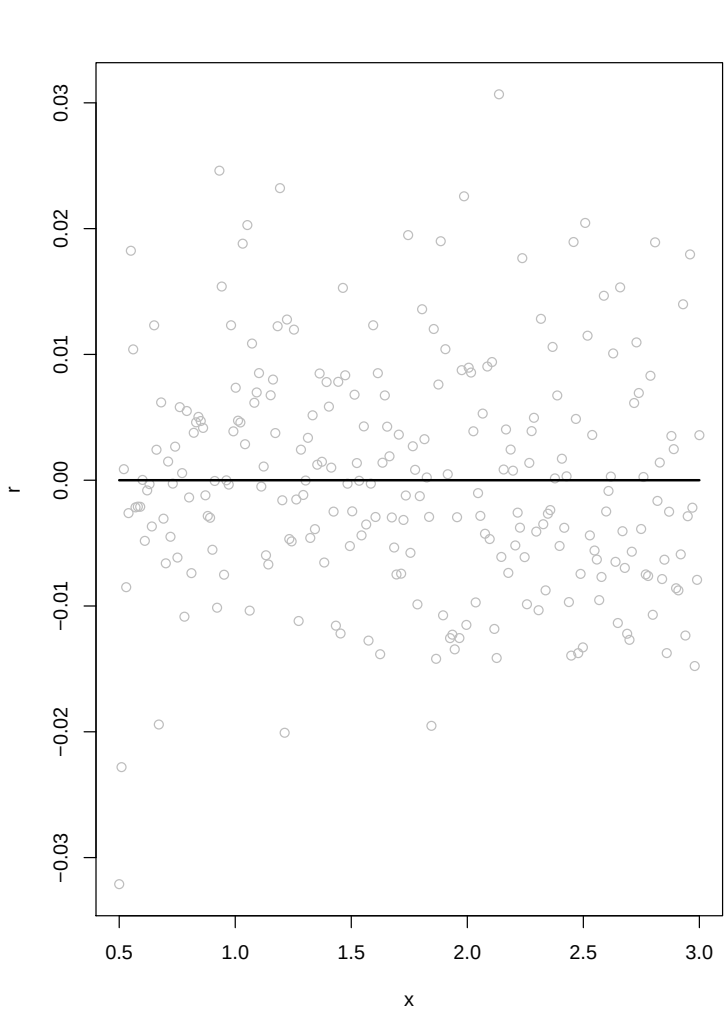

b = 20
y
x
0.56
0.54
0.52
0.50
0.48
0.46
0.44
0.5
1.0
1.5
2.0
2.5
3.0

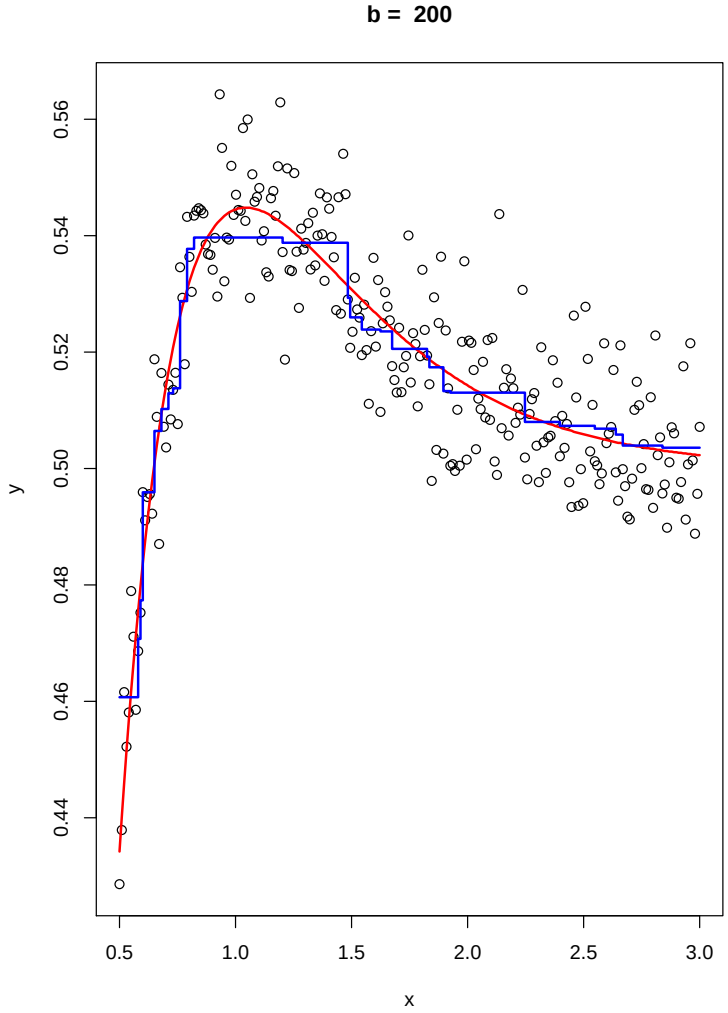

r
x
0.5
1.0
1.5
2.0
2.5
3.0
0.03
0.02
0.01
0.00
−0.01
−0.02
−0.03
−0.04


b = 50
y
x
0.5
1.0
1.5
2.0
2.5
3.0
0.56
0.54
0.52
0.50
0.48
0.46
0.44


r
x
0.5
1.0
1.5
2.0
2.5
3.0
0.03
0.02
0.01
0.00
−0.01
−0.02
−0.03


b = 100
y
x
0.5
1.0
1.5
2.0
2.5
3.0
0.56
0.54
0.52
0.50
0.48
0.46
0.44


r
x
0.5
1.0
1.5
2.0
2.5
3.0
0.03
0.02
0.01
0.00
−0.01
−0.02
−0.03


b = 200
y
x
0.5
1.0
1.5
2.0
2.5
3.0
0.56
0.54
0.52
0.50
0.48
0.46
0.44

Tuning parameters for boosting

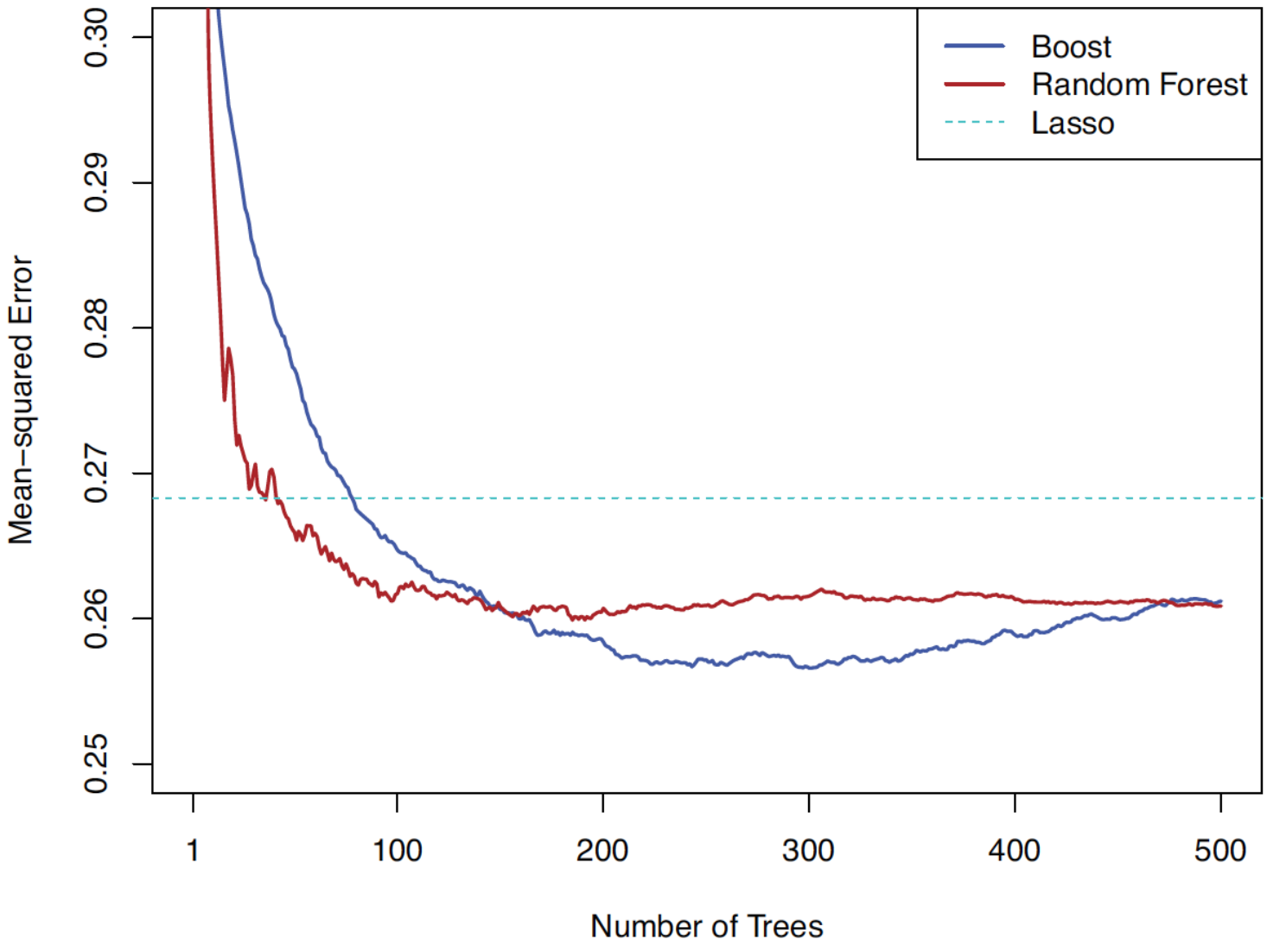
- The number of trees B
  Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. Use cross-validation to select B
- The shrinkage parameter λ
  A small positive number, which controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance
- The number of splits d
  It controls the complexity of the boosted ensemble. Often $d = 1$ works well, in which case each tree is a stump, consisting of a single split and resulting in an additive model. More generally d is the interaction depth, and controls the interaction order of the boosted model, since d splits can involve at most d variables

gbm()

```
fit <- gbm(y ~ .,
           distribution = "gaussian",
           data = train,
           n.trees = B,
           interaction.depth = d,
           shrinkage = lambda,
           bag.fraction = 0.5, # default
           cv.folds = 0)  # default
```

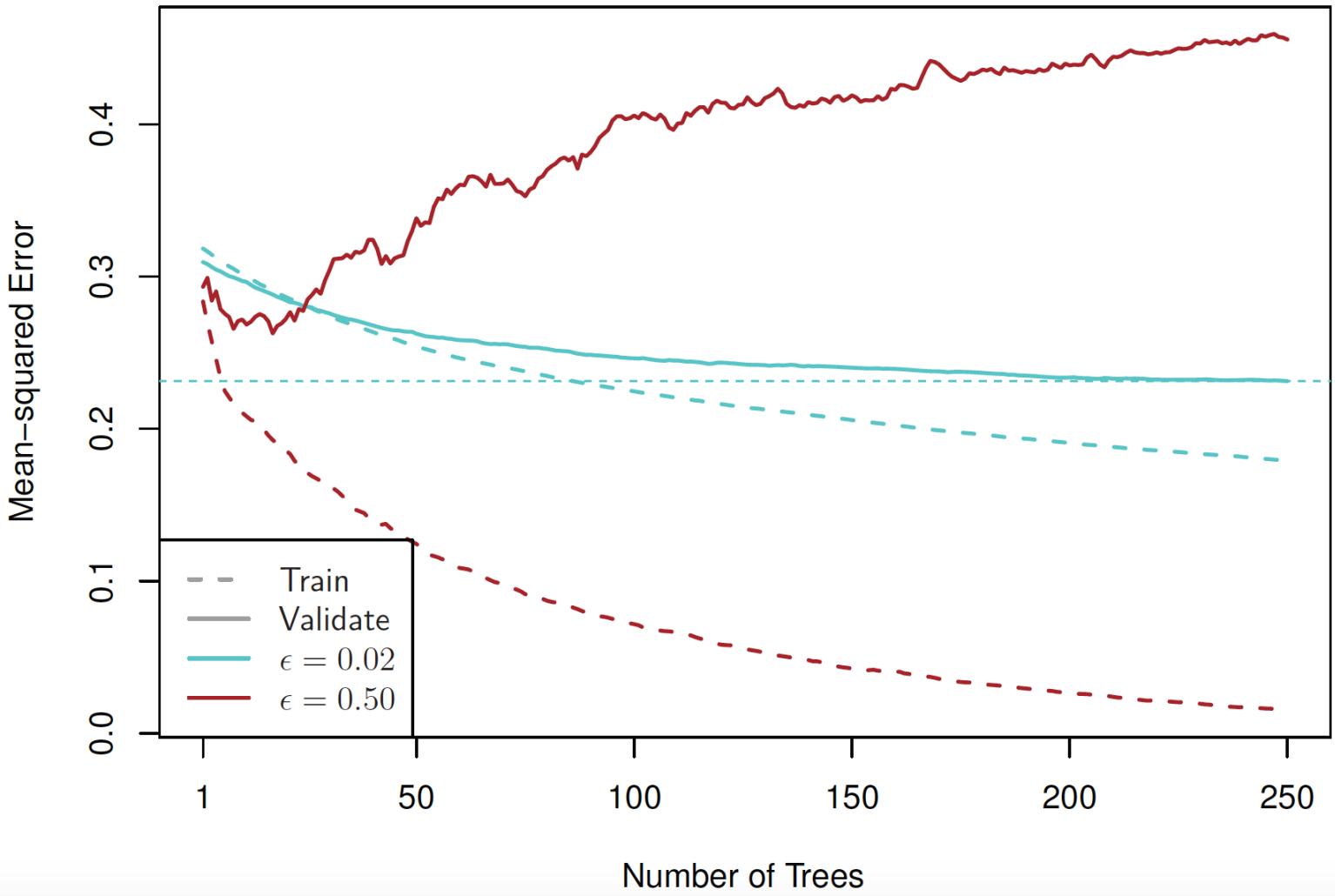
- `bag.fraction = 0.5` : grows each new tree on a 50% random sub-sample of the training data. Apart from speeding up the computations, this has a similar effect to bagging, and results in some variance reduction in the ensemble
- `cv.folds = 0` : no cross-validation

Number of trees B



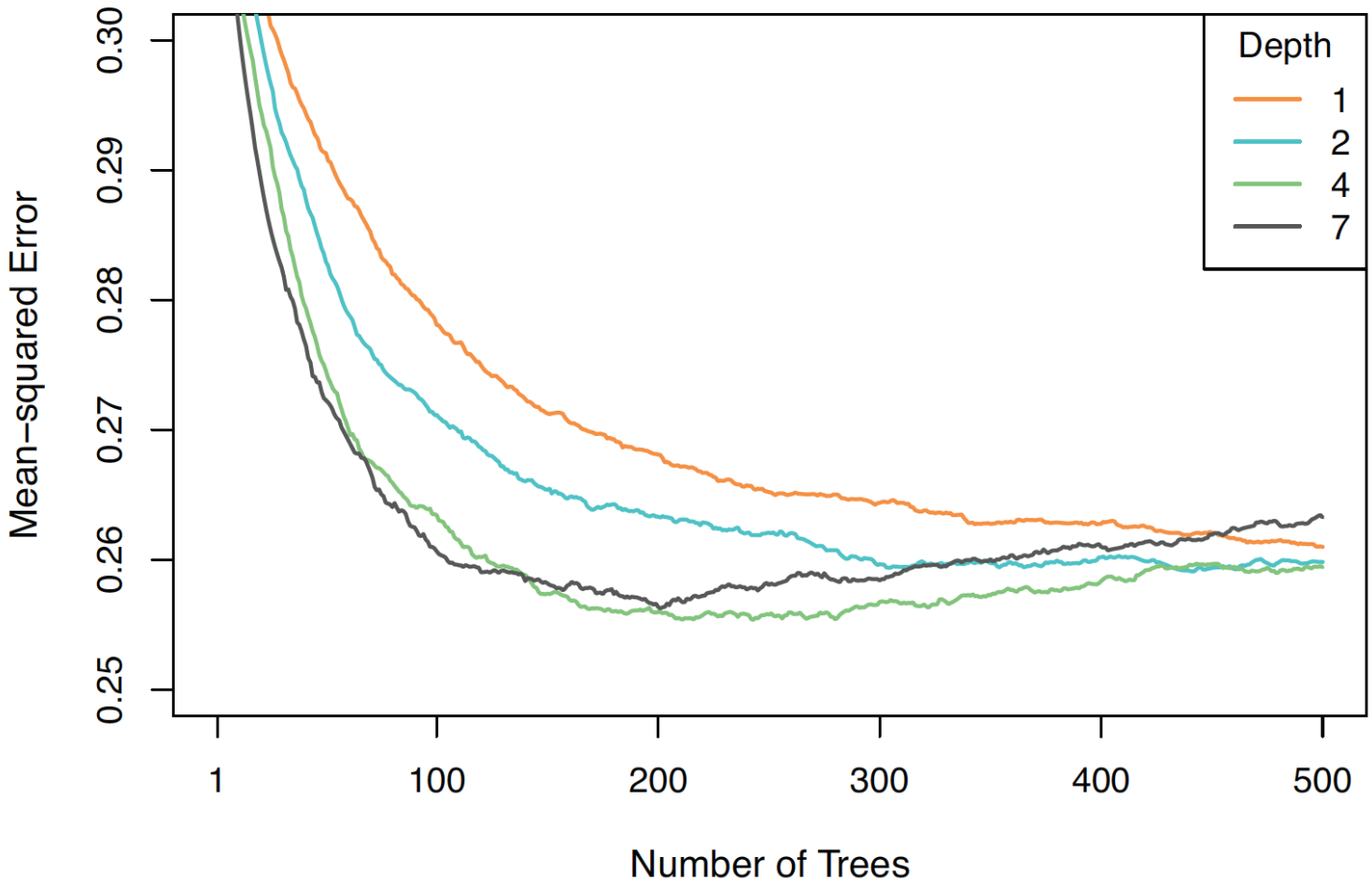
Source: Efron and Hastie (2016), Figure 17.6

Shrinkage parameter λ



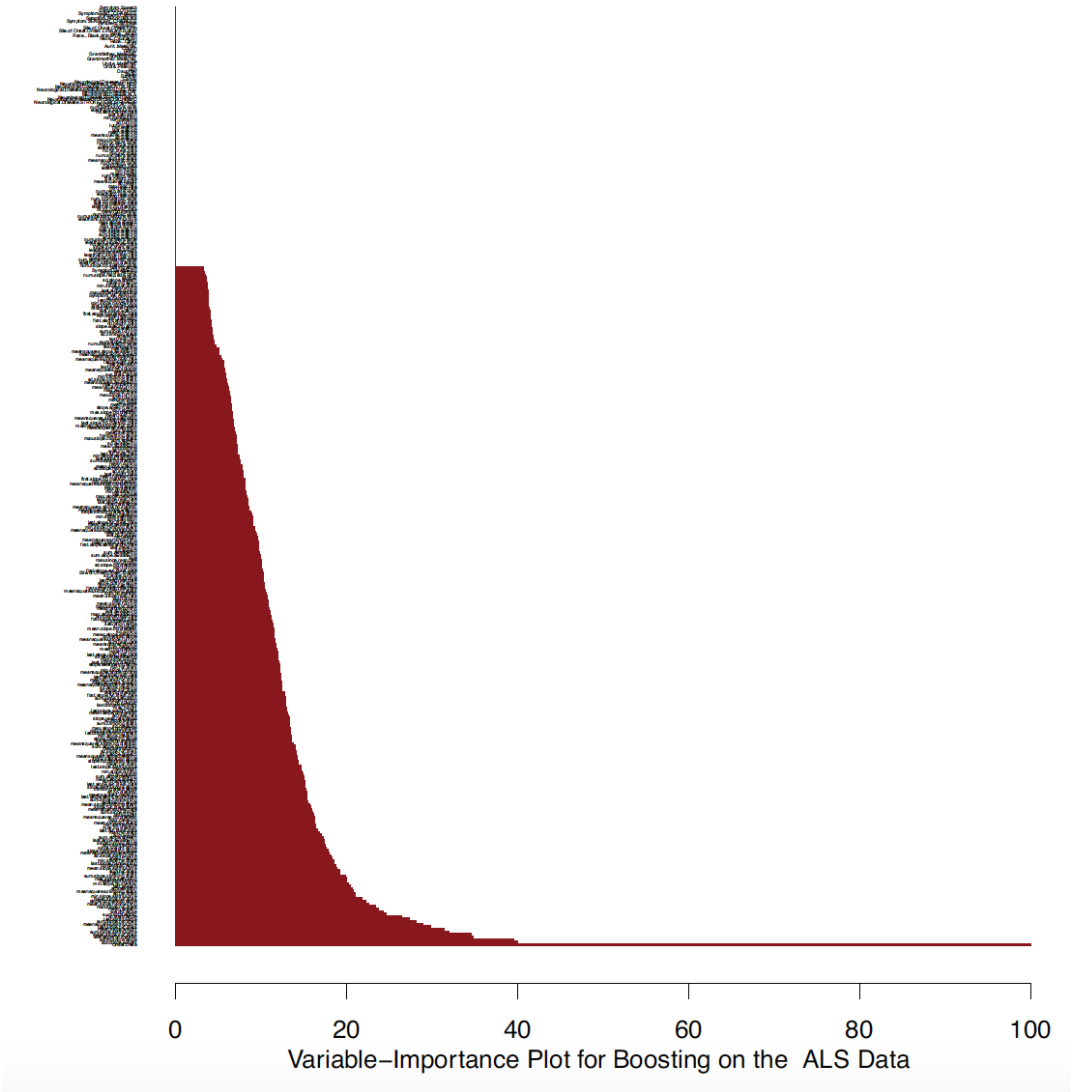
Source: Efron and Hastie (2016), Figure 17.10; ϵ in the legend is λ

Number of splits d



Source: Efron and Hastie (2016), Figure 17.8

Variable Importance



Source: Efron and Hastie (2016), Figure 17.7

Adaboost: the original boosting algorithm

- Boosting was originally proposed as a means for improving the performance of weak learners in binary classification problems
- The idea is to fit a sequence of classifiers to modified versions of the training data, where the modifications give more weight to misclassified observations
- The final classification is by weighted majority vote
- We present the AdaBoost.M1 algorithm (Freund and Schapire, 1997), which was developed for the response variable coded as $Y \in \{-1, 1\}$ and uses decision trees as weak learners

Adaboost algorithm for classification trees

1. Initialize the observation weights $w_i = 1/n, i = 1, \ldots, n$
2. For $b = 1, \ldots, B$ repeat the following steps
   - **(a)** Fit a classification tree $\hat{C}^b$ to the training data, using observation weights w_i
   - **(b)** Compute the weighted misclassification error for $\hat{C}^b$

$$\mathrm{Err}^b = \frac{\sum_{i=1}^n w_i I\{y_i \neq \hat{C}^b(x_i)\}}{\sum_{i=1}^n w_i}$$

   - **(c)** Compute

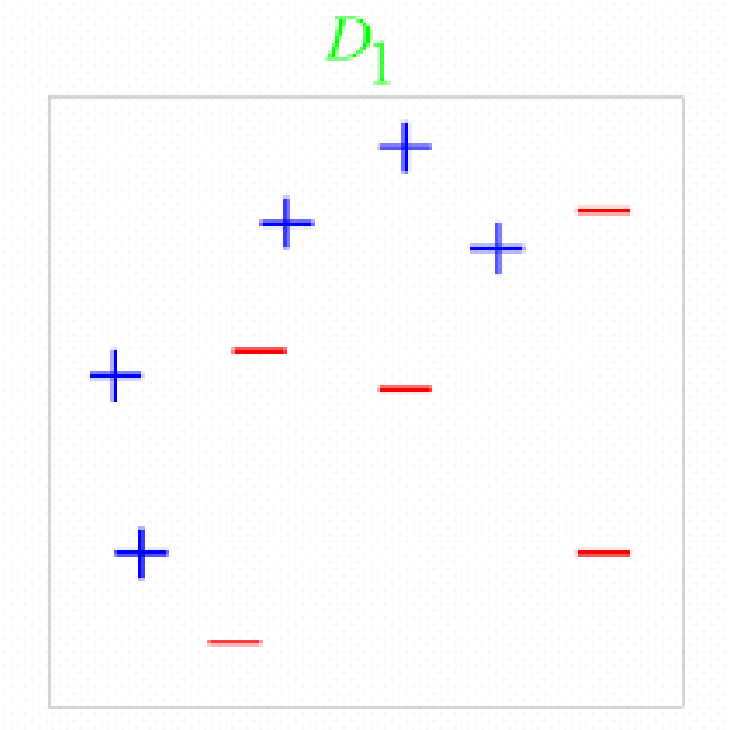
$$\alpha^b = \log[(1 - \mathrm{Err}^b)/\mathrm{Err}^b]$$

   - **(d)** Update the weights as

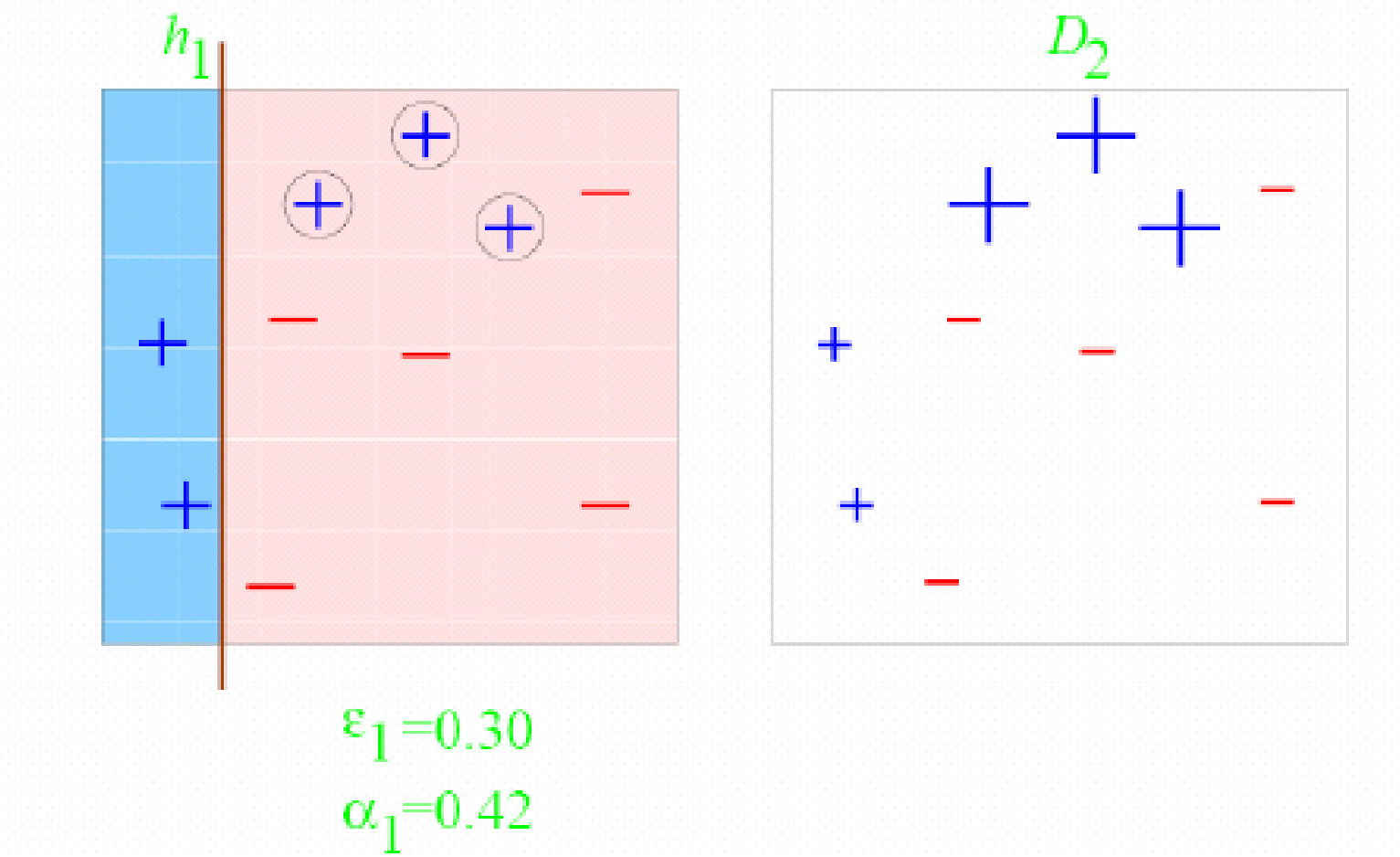
$$w_i \leftarrow w_i \cdot \exp(\alpha^b \cdot I\{y_i \neq \hat{C}^b(x_i)\}), \quad i = 1, \ldots, n$$

3. Output $\hat{C}^B(x) = \mathrm{sign}(\sum_{b=1}^B \alpha^b \hat{C}^b(x))$

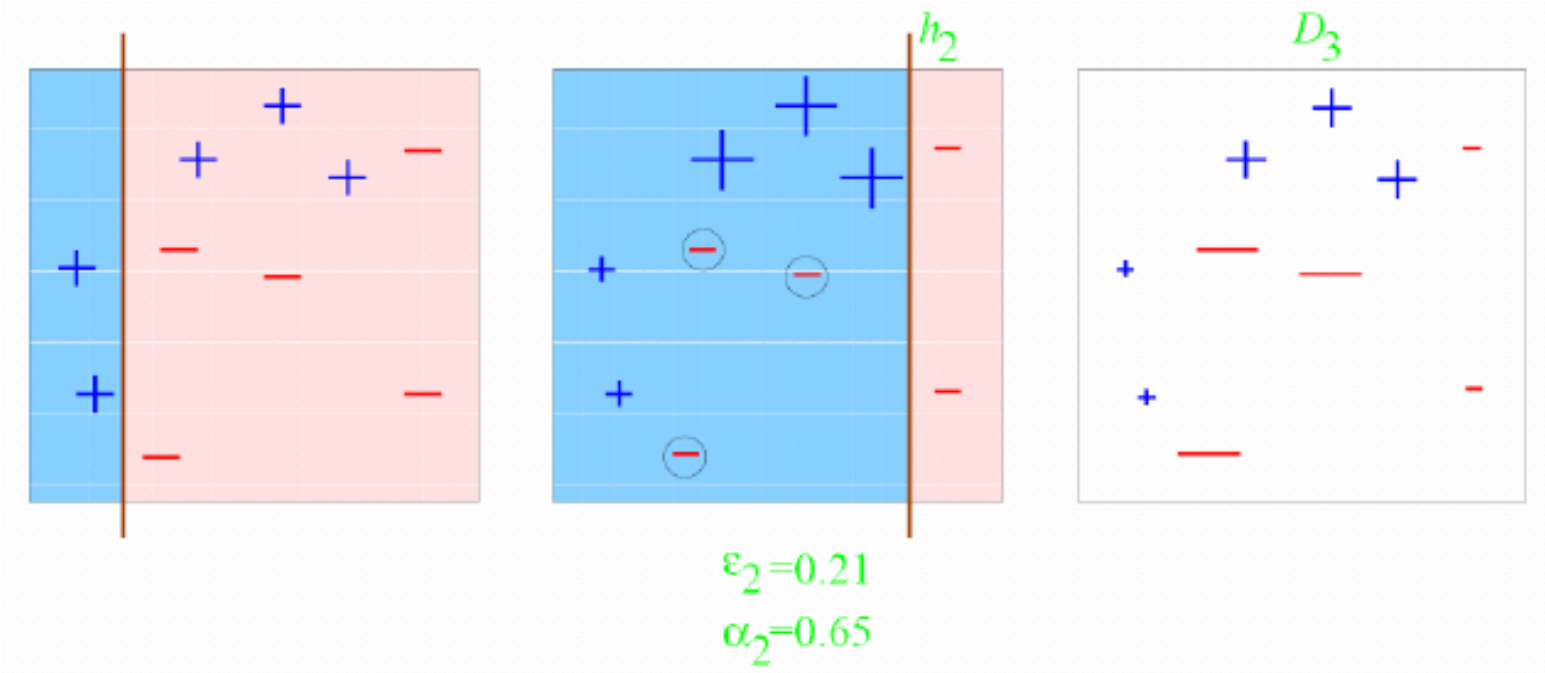
Adaboost example



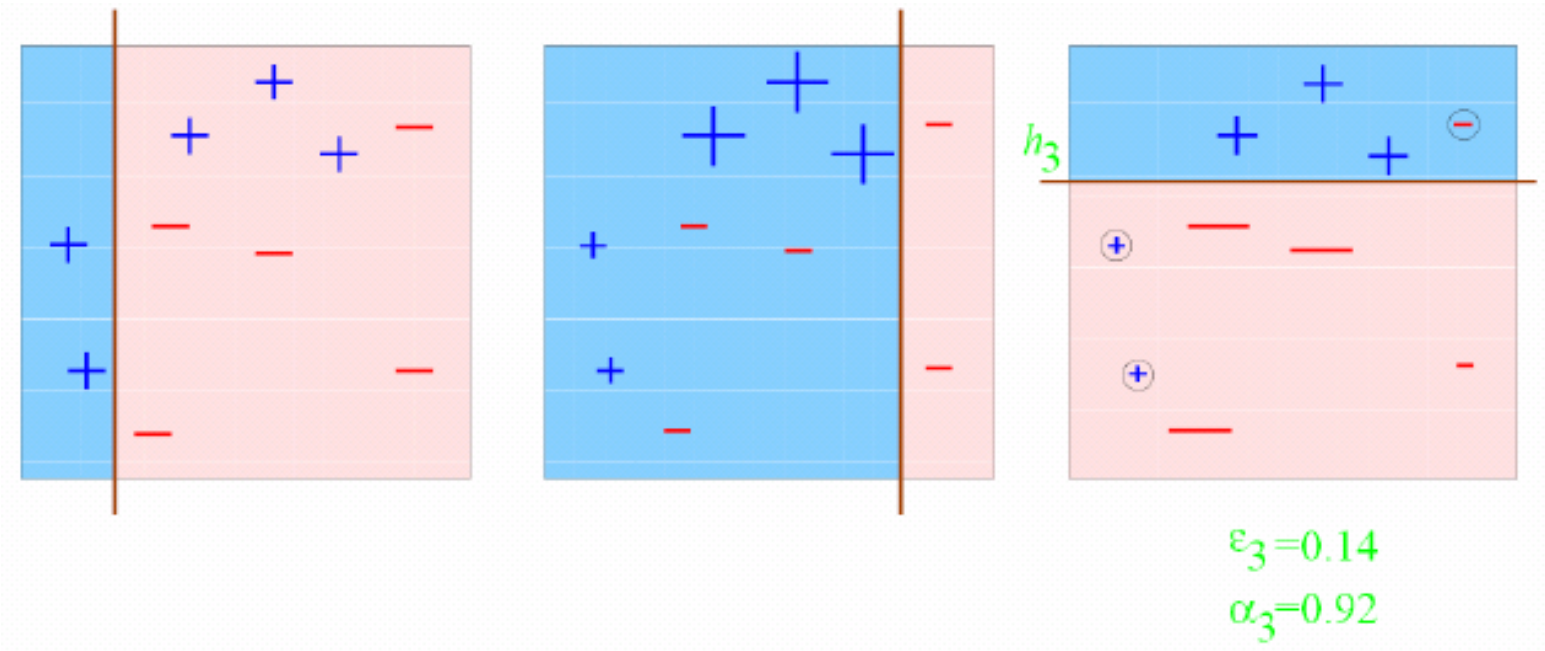
Adaboost example



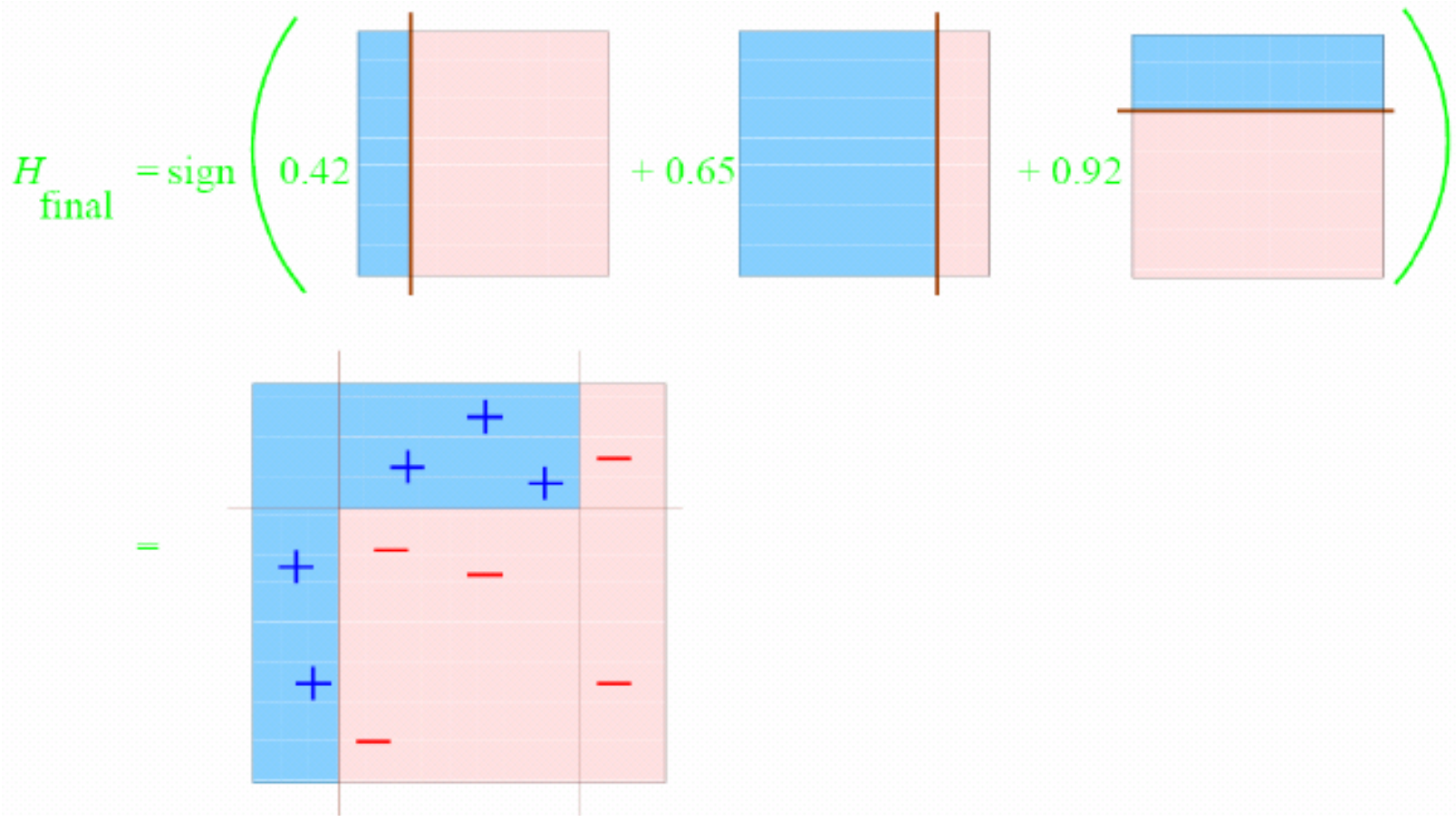
Adaboost example



Adaboost example



Adaboost example



Tuning of boosting

- As is often the case, boosting slightly outperforms a random forest here, but at a price
- Careful tuning of boosting requires considerable extra work, with time-costly rounds of cross-validation, whereas random forests are almost automatic

Outline

Model stacking

- Stacking is a general method to combine models
- Consider a library of L models $\hat{f}_1, \ldots, \hat{f}_L$ fitted on the training data

$$(x_1, y_1), \ldots, (x_n, y_n)$$

- Perhaps by combining their respective efforts, we could get even better prediction than using any particular one
- The issue then how we might combine them for predicting the test set $y_1^*, \ldots, y_m^*$
- A linear combination

$$\hat{y}_i^* = \sum_{l=1}^{L} w_l \hat{f}_l(x_i^*)$$

requires to define the the weights $w_1, \ldots, w_L$

Least squares

- The method of least squares provides the weights

$$\hat{w}_1, \ldots, \hat{w}_L = \underset{w_1,\ldots,w_L}{\arg\min} \sum_{i=1}^{n} \left[y_i - \sum_{l=1}^{L} w_l \hat{f}_l(x_i) \right]^2$$

- However, in this way we fail to take into account for model complexity: models with higher complexity get higher weights
- For example, consider L predictors and let $\hat{f}_l$ be the linear model formed the best subset of predictors of size l, $l = 1, \ldots, L$, where best is defined as having the smallest $\mathrm{MSE}_{\mathrm{Tr}}$
- Then all the weight goes on the largest model, that is, $\hat{w}_L = 1$ and $\hat{w}_l = 0$ for $l < L$

Stacked regression

- Wolpert (1992) presented an interesting idea, called *stacked generalizations*. This proposal was translated in statistical language by Breiman, in 1993
- If we exclude y_i in the fitting procedure of the models, then $\hat{f}_1^{-i}(x_i), \ldots, \hat{f}_L^{-i}(x_i)$ do not depend on y_i
- Stacked regression is a particular case of the model stacking algorithm (next) with $\hat{f}_{\text{stack}} =$ linear model and cross-validation with $K = n$
- Read *Guide to Model Stacking* by Ben Gorman: https://gormanalysis.com/guide-to-model-stacking-i-e-meta-ensembling/

Stacked regression

- Let $\hat{f}_l^{-i}(x_i)$ be the prediction at x_i using model l fitted to the training data with the ith training observation (x_i, y_i) removed
- Obtain the weights by least squares

$$\hat{w}_1, \ldots, \hat{w}_L = \underset{w_1,\ldots,w_L}{\arg\min} \sum_{i=1}^{n} \left[y_i - \sum_{l=1}^{L} w_l \hat{f}_l^{-i}(x_i) \right]^2$$

- Compute the predictions for the test data as

$$\hat{f}_{\text{stack}}(x_i^*) = \sum_{l=1}^{L} \hat{w}_l \hat{f}_l(x_i^*), \quad i = 1, \ldots, m$$

Model stacking algorithm

1. Partition the training data into K folds $\mathcal{F}_1, \ldots, \mathcal{F}_K$
2. For each test fold $\mathcal{F}_k$, $k = 1, \ldots, K$,combine the other $K - 1$ folds to be used as a training fold
   - **2.1** For $l = 1, \ldots, L$, fit the lth model to the training fold and make predictions on the test fold $\mathcal{F}_k$. Store these predictions

$$z_i = (\hat{f}_1^{-\mathcal{F}_k}(x_i), \ldots, \hat{f}_L^{-\mathcal{F}_k}(x_i)), \quad i \in \mathcal{F}_k$$

3. For $l = 1, \ldots, L$, fit the lth model to the full training data and make predictions on the test data. Store these predictions

$$z_i^* = (\hat{f}_1(x_i^*), \ldots, \hat{f}_L(x_i^*)), \quad i = 1, \ldots, m$$

4. Fit the stacking model $\hat{f}_{\text{stack}}$ using

$$(y_1, z_1), \ldots, (y_n, z_n)$$

5. Make final predictions $\hat{y}_i^* = \hat{f}_{\text{stack}}(z_i^*)$, $i = 1, \ldots, m$

caretEnsemble

Read *A Brief Introduction to caretEnsemble* by Zach Mayer: https://cran.r-project.org/web/packages/caretEnsemble/vignettes/caretEnsemble-intro.html

```
# K fold CV for regression
KCV <- trainControl(method="cv", number=K,
  savePredictions="final",index=createResample(train$y, K)
  )
# list of models fit1 and fit2
List <- caretList(y ~. , data=train,
  trControl=KCV,
  methodList=c("fit1","fit2")
  )
# ensemble fit
fit.ensemble <- caretEnsemble(List, metric="RMSE")
summary(fit.ensemble)
```