Decision Trees

M2 D3S/EGR 2025-2026

Louis Olive

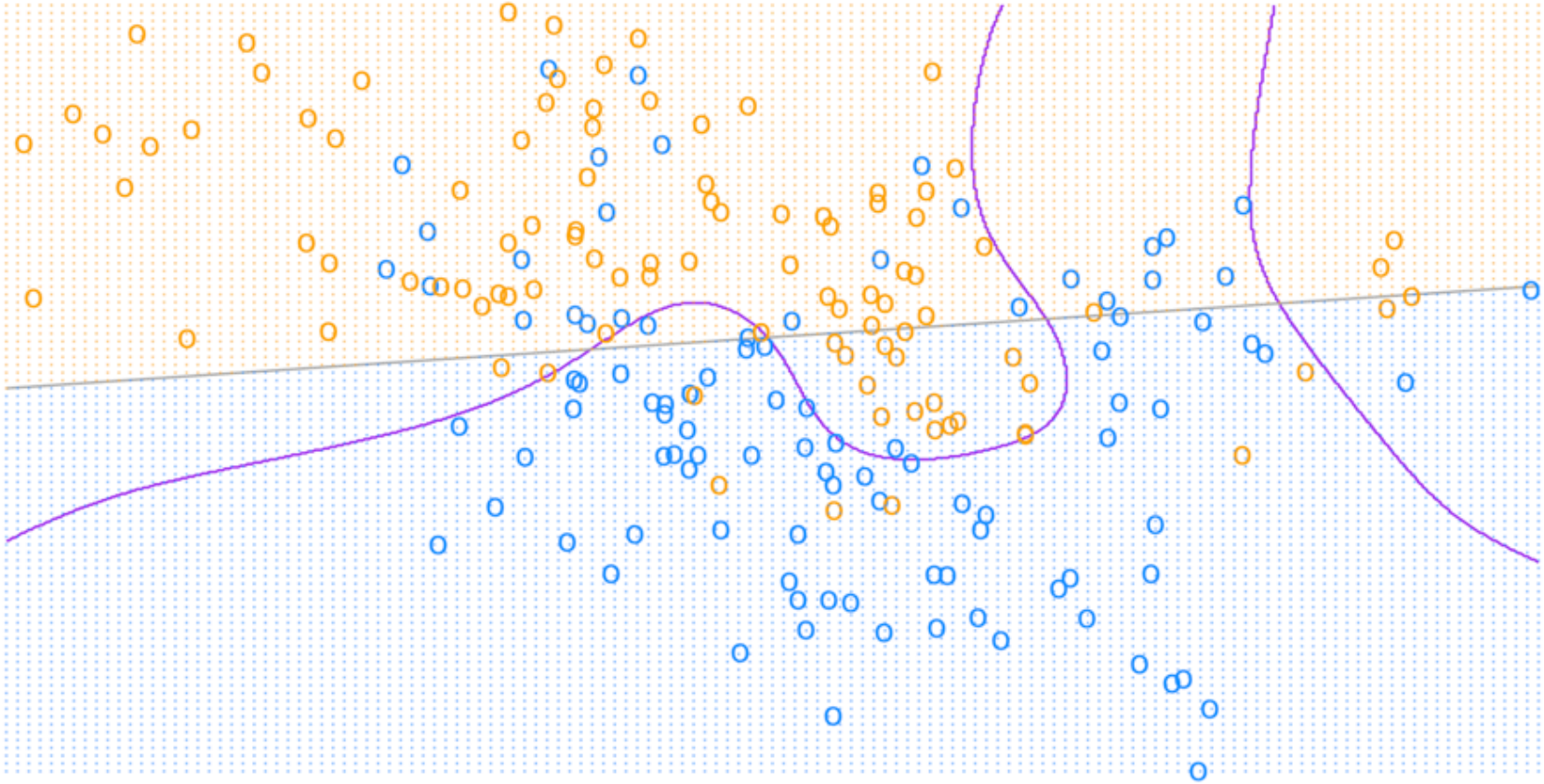
louis.olive@gmail.com / louis.olive@ut-capitole.fr

October 8, 2025

Decision Trees

Scoring

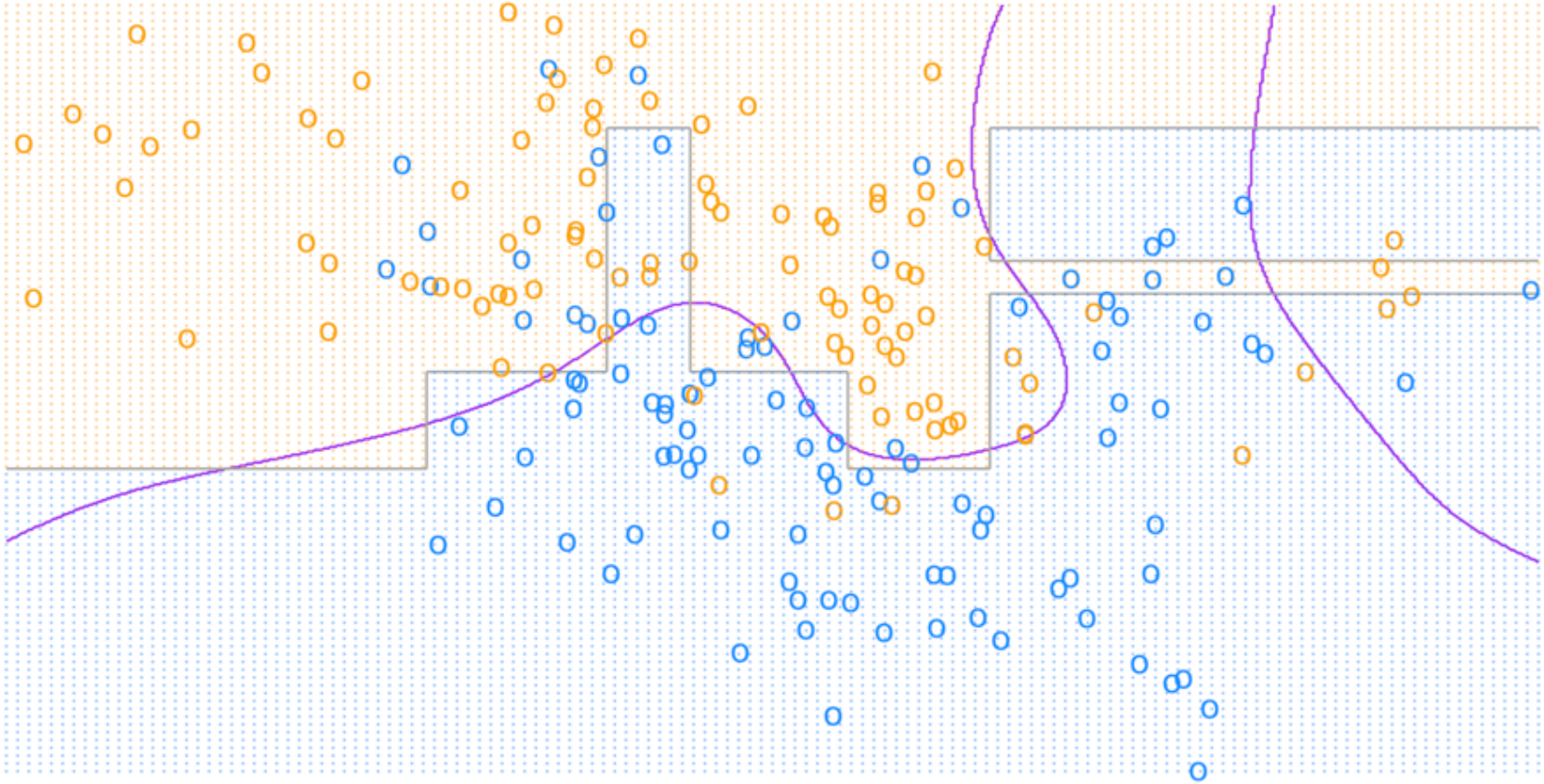
Mixture data set: Logistic Regression



The empirical risk on testing set is 0.291.

Scoring

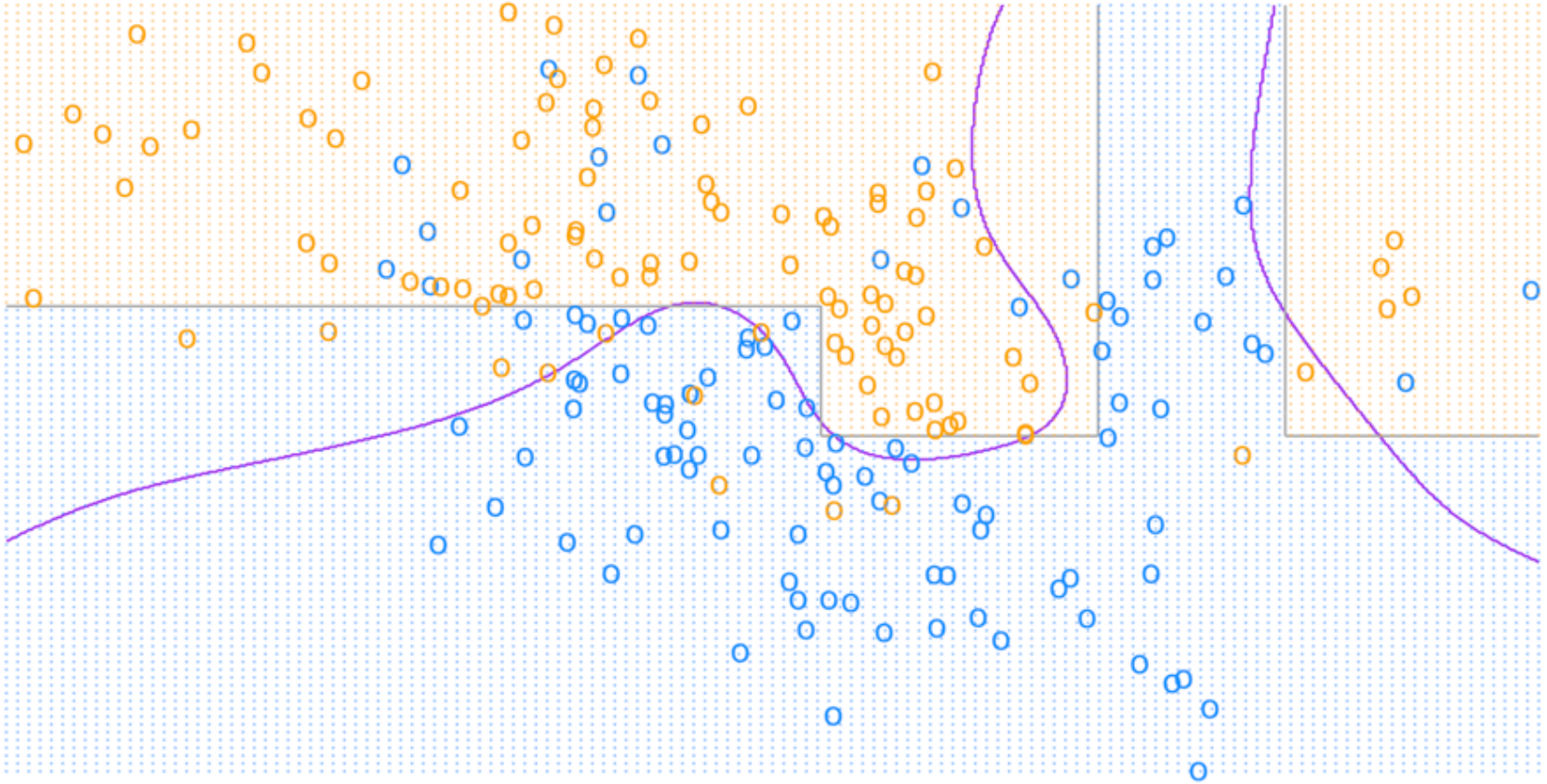
Toy example - Logistic Regression - binning



The empirical risk on testing set is 0.265.

Scoring

Toy example - Decision Trees

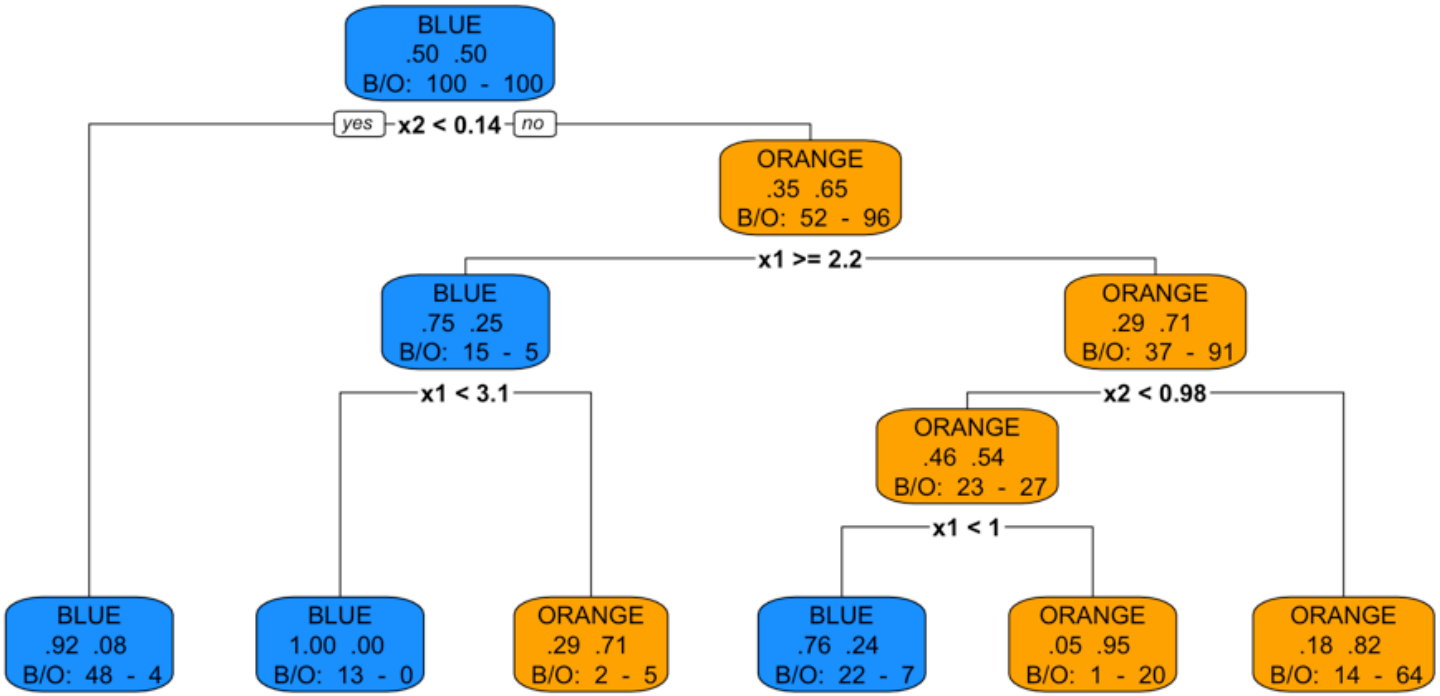


The empirical risk on testing set is 0.247.

Scoring

Toy example - Decision Trees

We show below the previously fitted Decision tree:

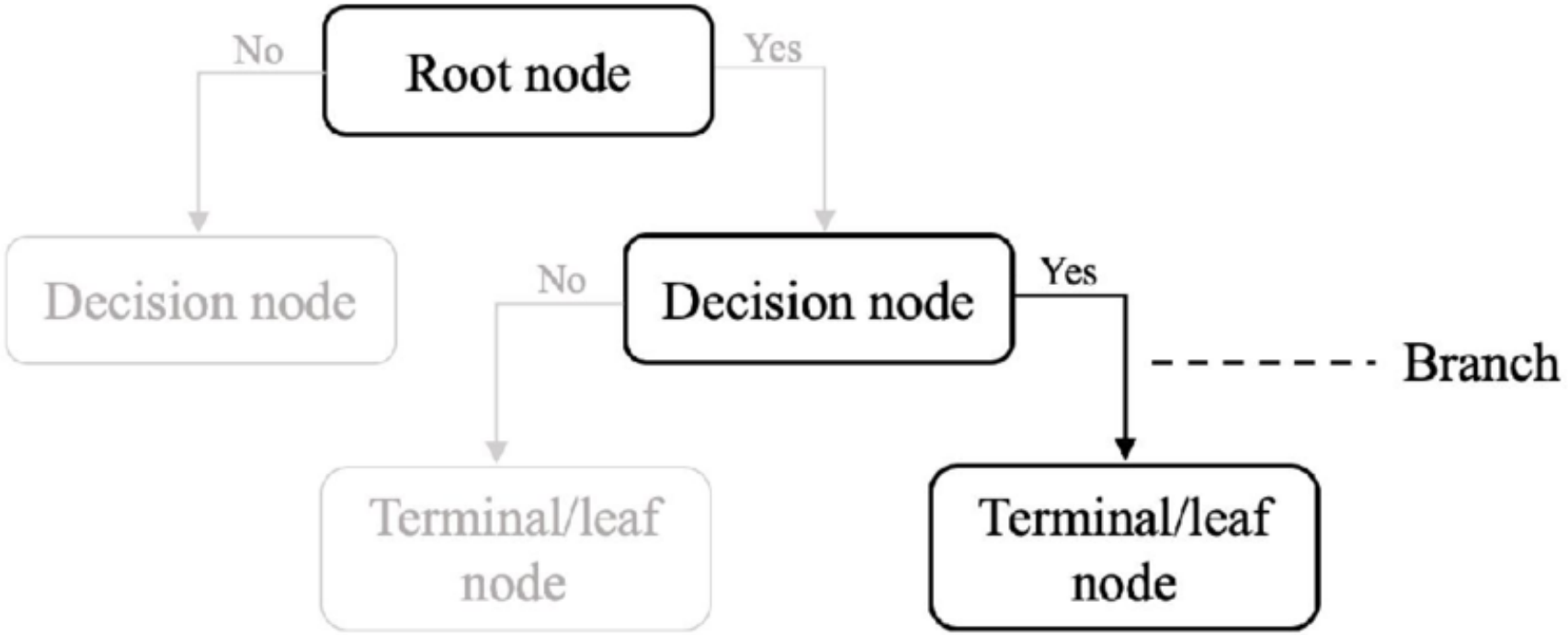


How to fit such a tree? What criterion is used?

Decision Trees

Terminology

Decision trees recursively partition the data by applying specific cutoff values to the features. This process creates various subsets of the data set, with each data point belonging to one of these subsets. The final subsets are known as Terminal or Leaf nodes, while the intermediate ones are referred to as Internal, Split or Decision nodes.



Decision Trees

CART

Classification And Regression Tree (CART) (Breiman et al. (1983)), is a recursive method:

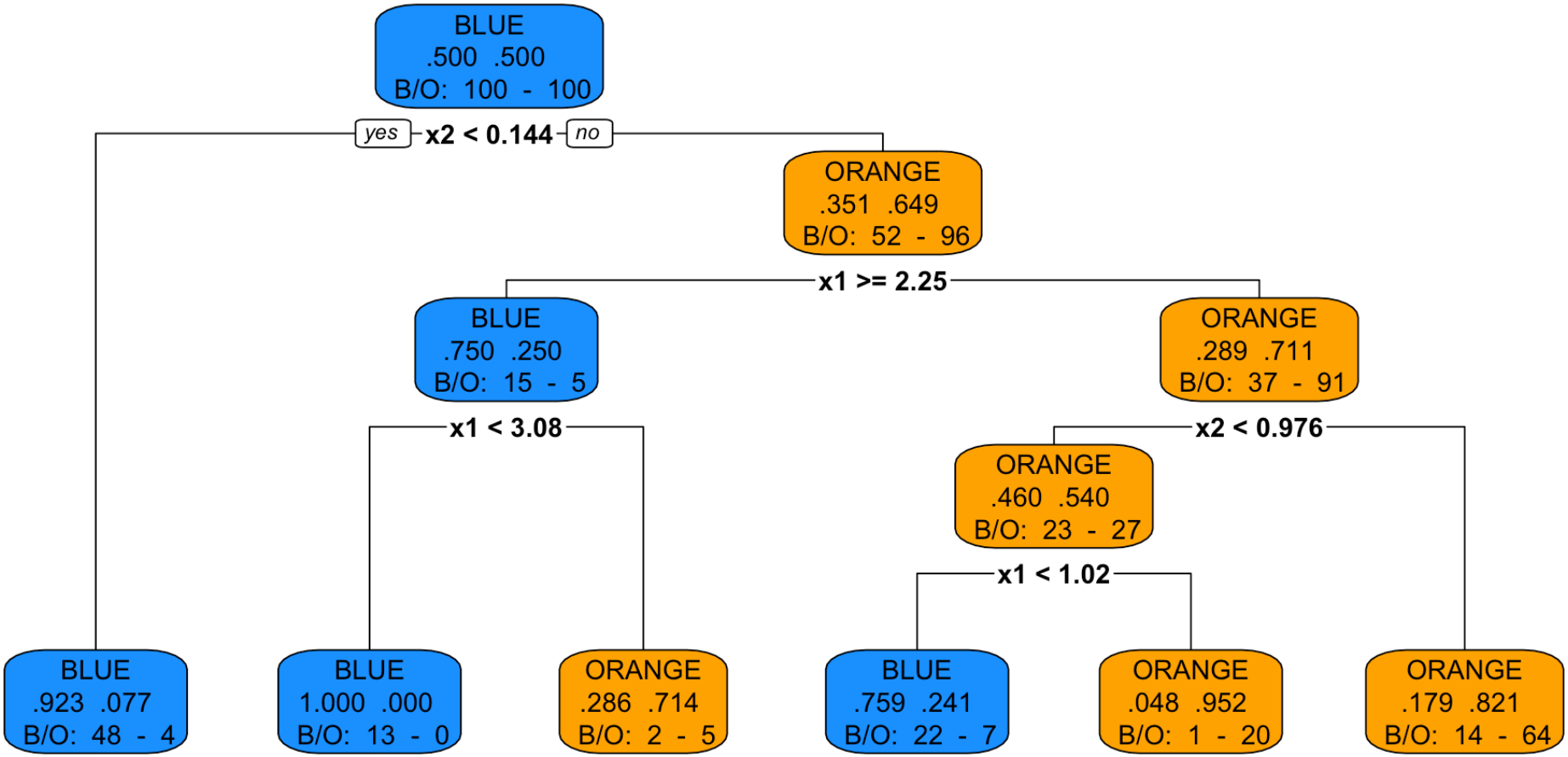
- At the root of the tree we find the entire sample.
- Each node of the tree divides the sample into 2 branches, according to a feature variable (discrete, continuous or ordinal variable (threshold) or a nominal variable (set of categories)).
- A terminal node is called a leaf. Usually the tree is represented upside down with its root at the top

The tree is built by the following process:

- First find the single variable which 'best' **splits** the data into two groups ('best' will be defined later).
- The data is separated, and then this process is applied separately to each sub-group, and so on recursively until a **stopping rule** occurs (either no improvement can be made or the subgroups reach a minimum size).

Decision Trees

CART for Mixture data set

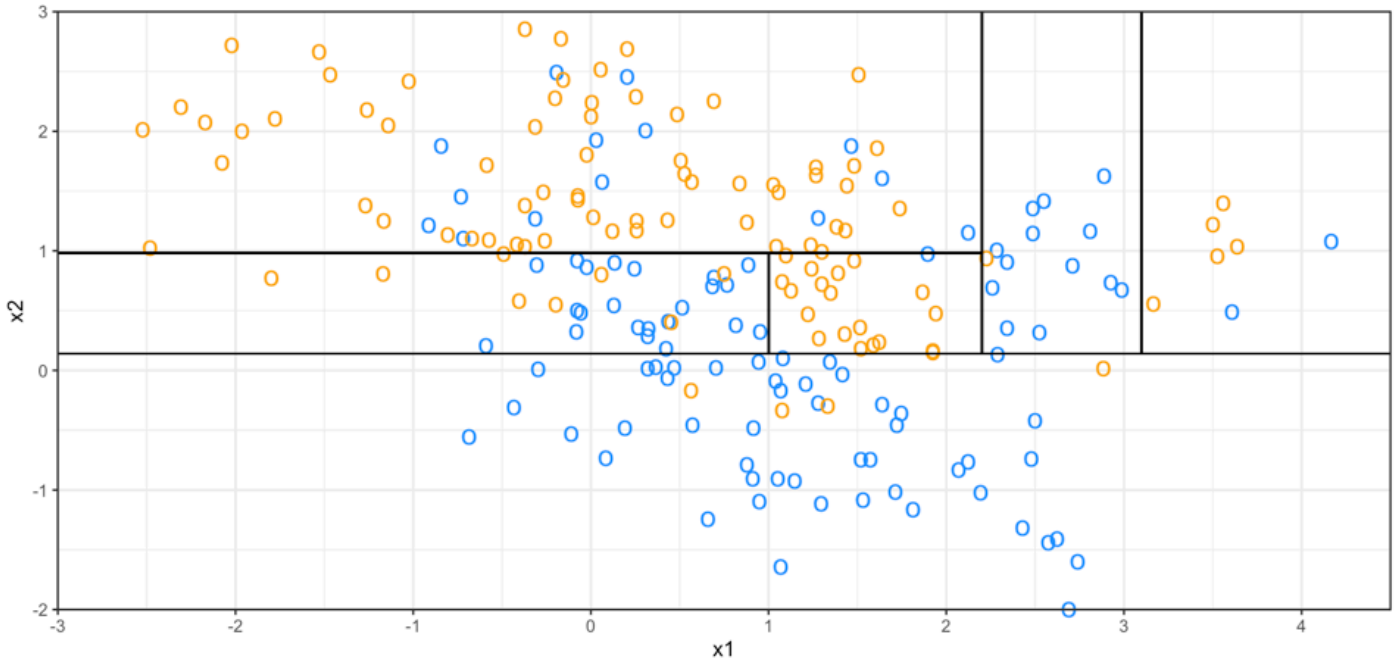


Decision Trees

CART for Mixture data set

Starting from the top of the tree and going down the `CART/rpart` algorithm splits at each node according to a binary decision.

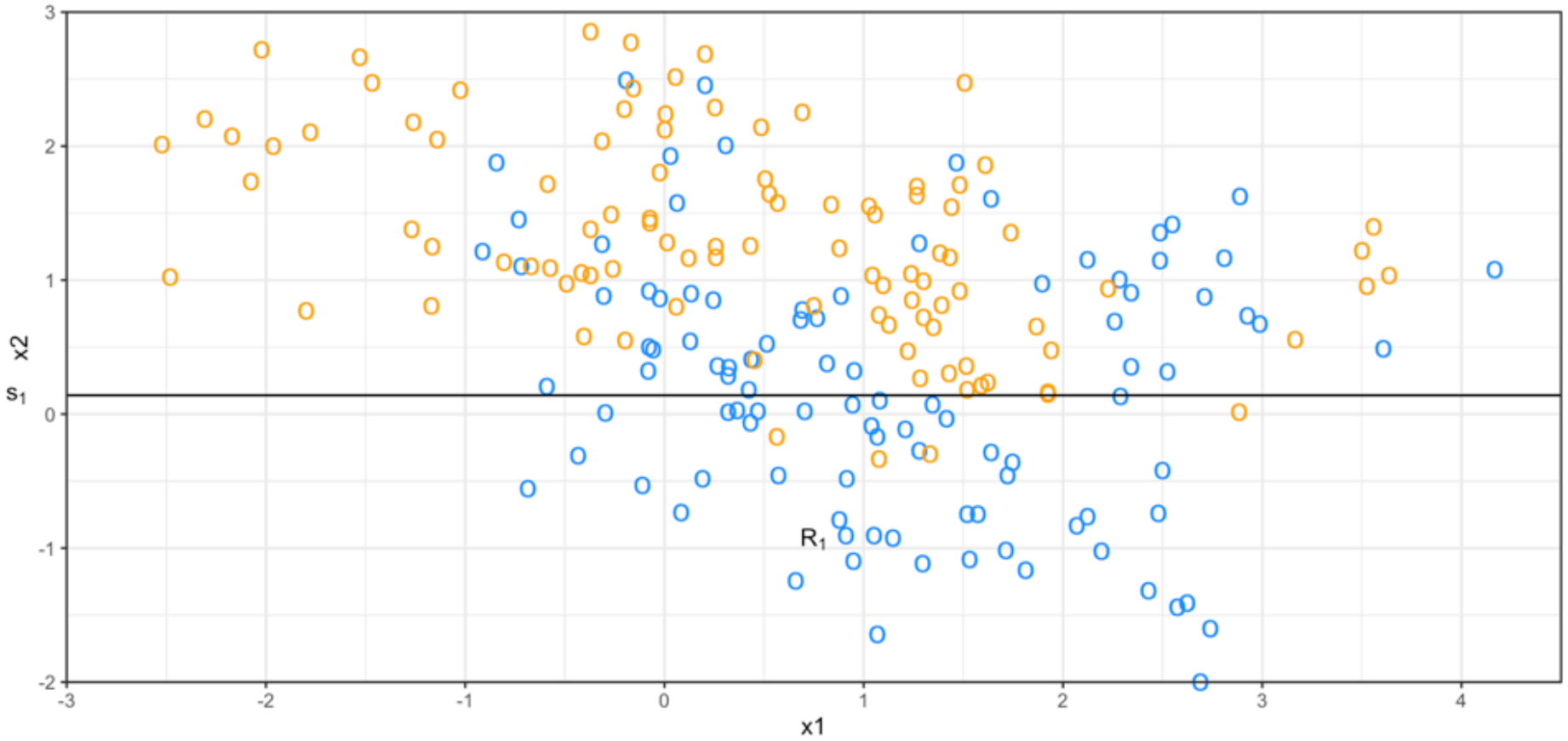
It ends up splitting the space into six regions, and then models the output by the mode/majority (classification) or proportion (scoring/probability) of Y in each region:



Decision Trees

CART for Mixture data set

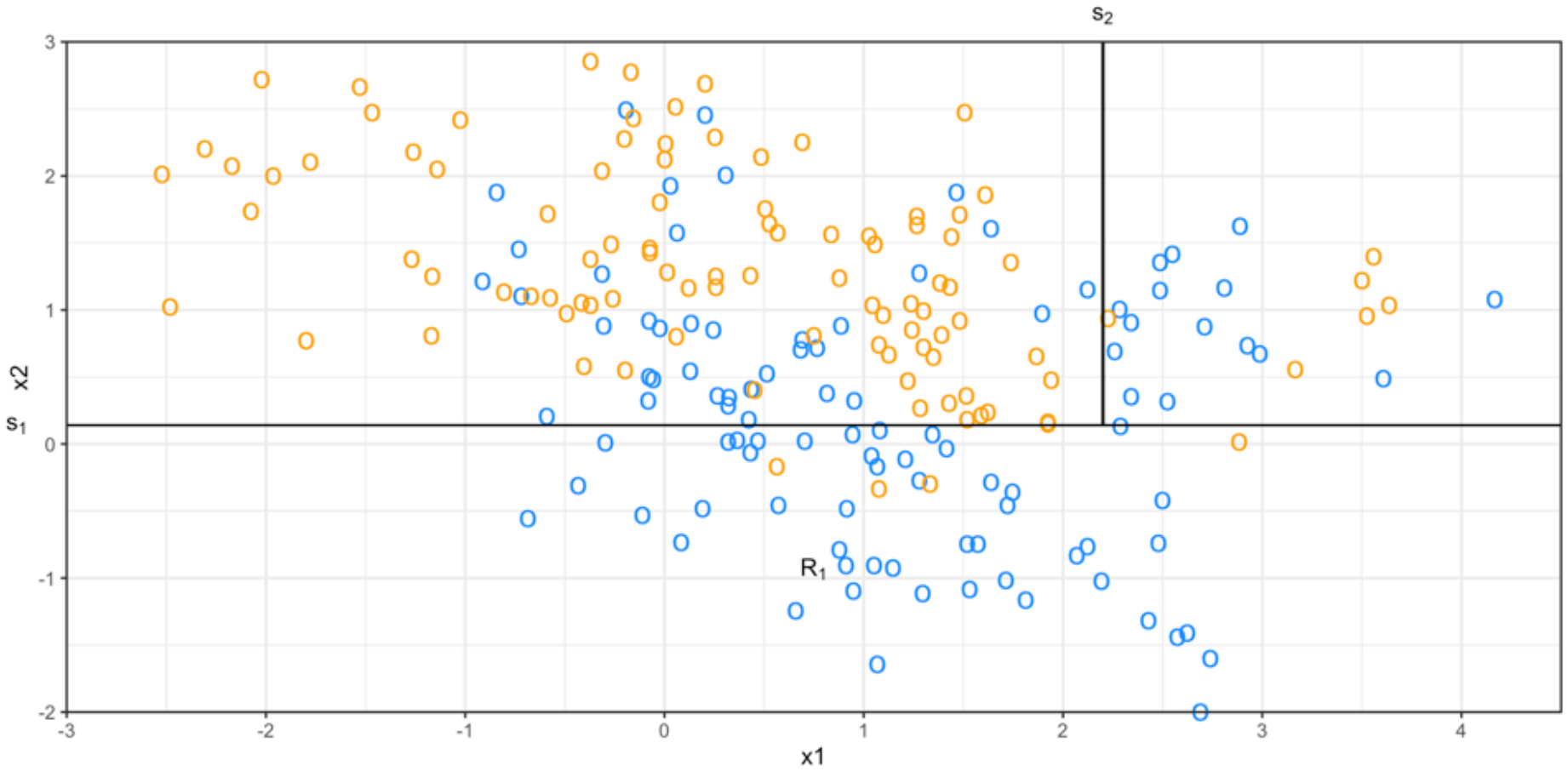
For example, with the Mixture data, `CART/rpart` first splits at $x_2 = s_1 = 0.14$:



Decision Trees

CART for Mixture data set

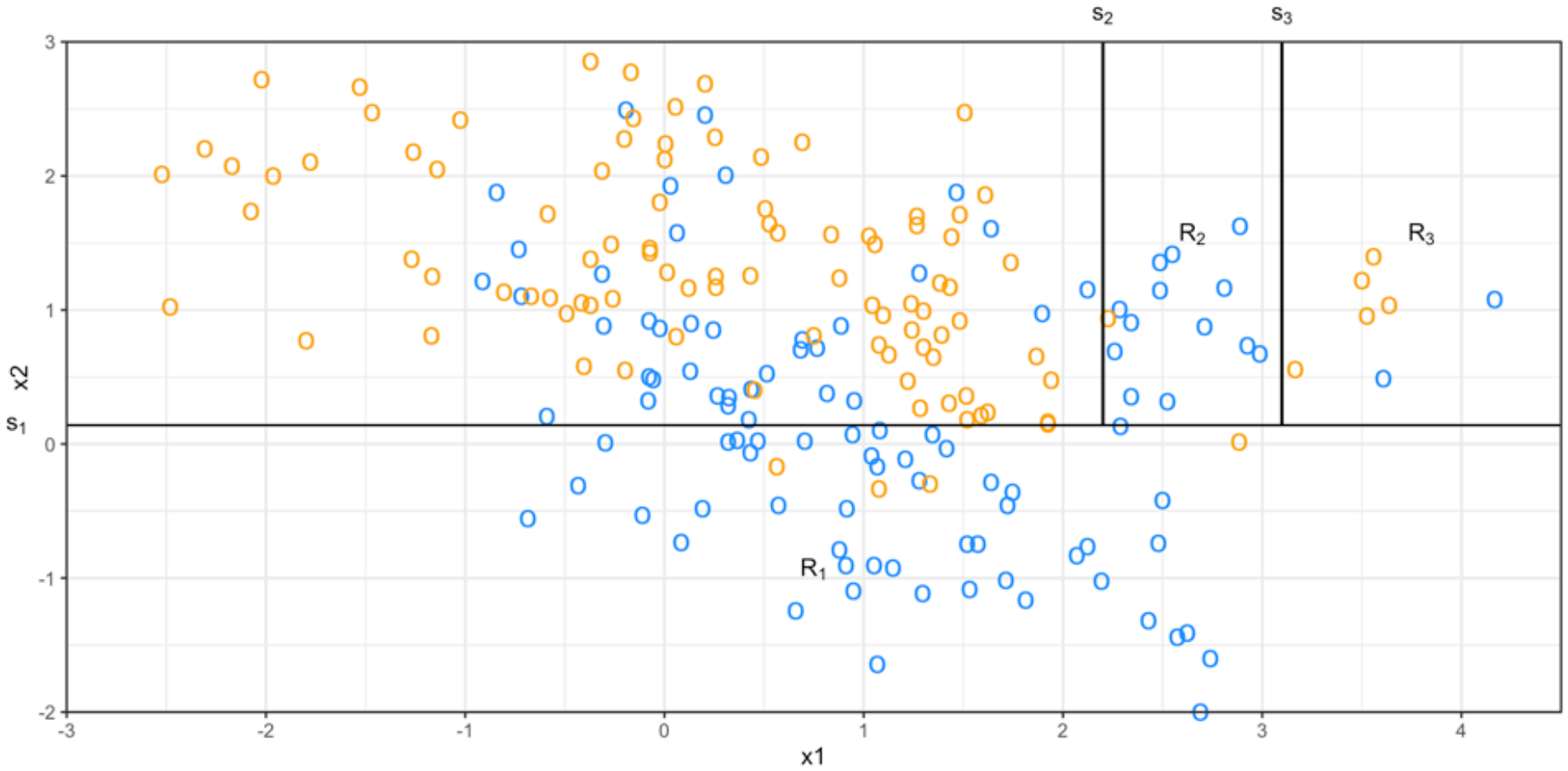
Then, the region $x_2 \geq s_1$ is split at $x_1 = s_2 = 2.2$:



Decision Trees

CART for Mixture data set

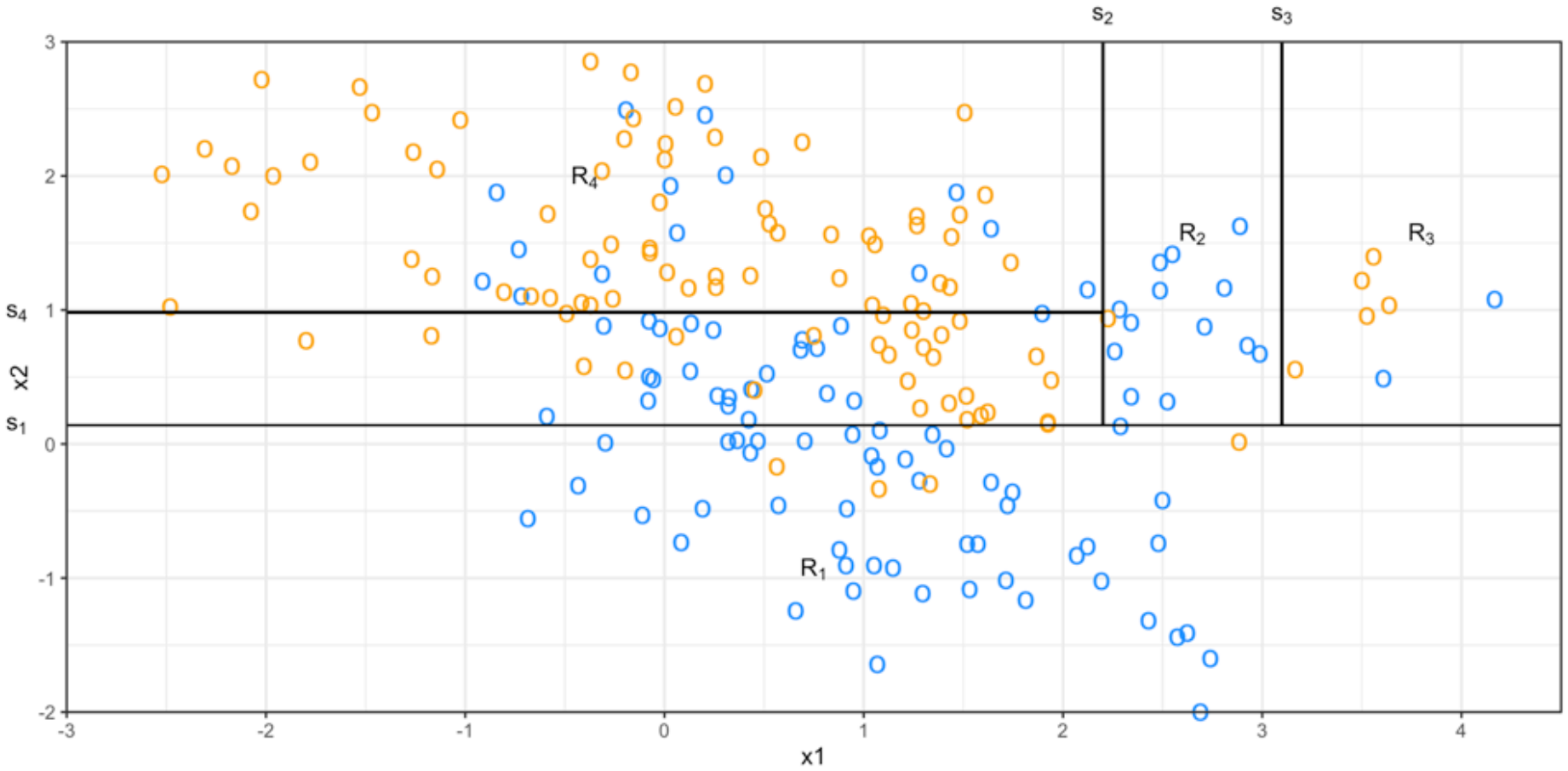
Then, the region $x_2 \geq s_1,\ x_1 > s_2$ is split at $x_1 = s_3 = 3.1$:



Decision Trees

CART for Mixture data set

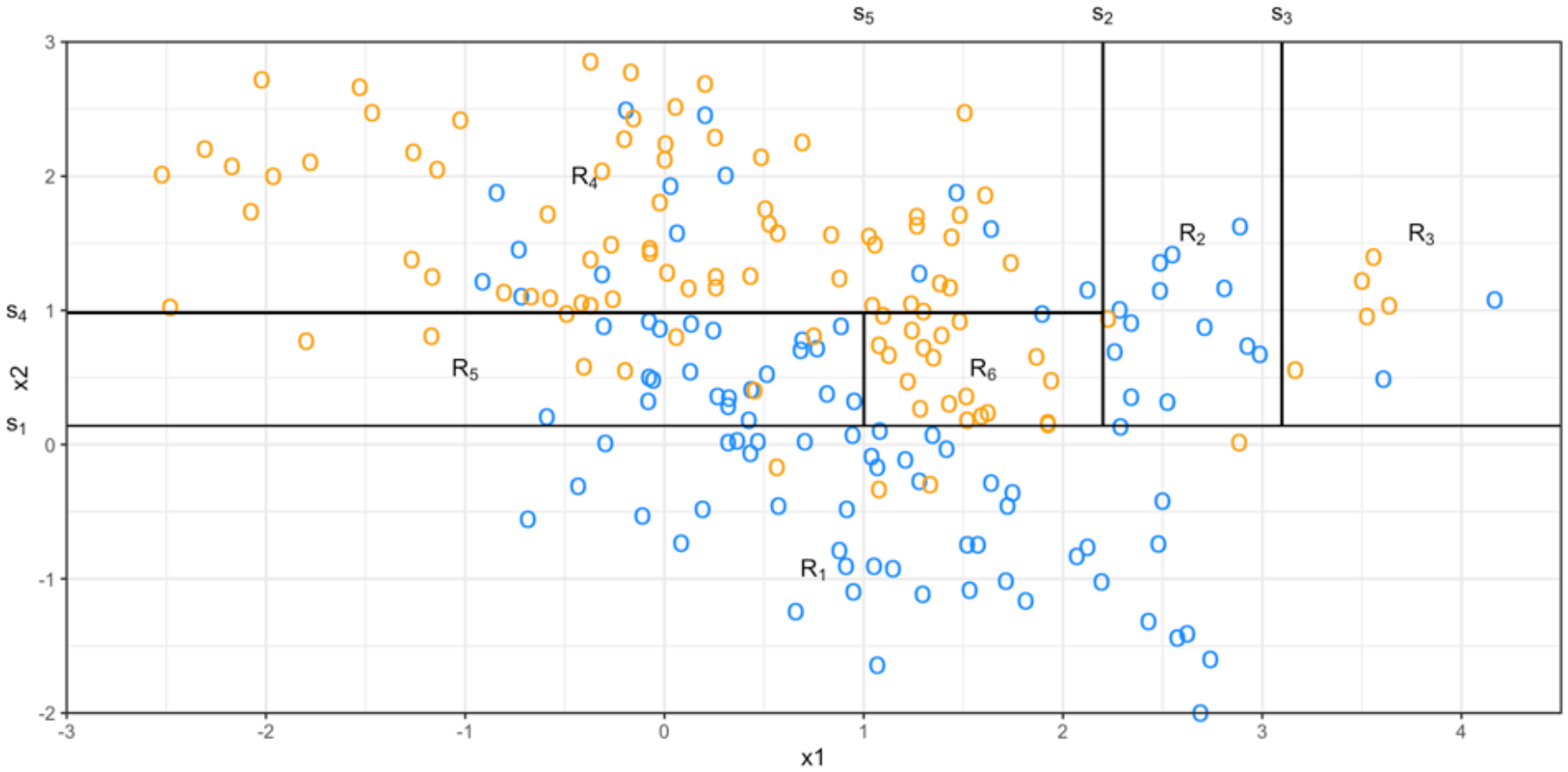
And the region $x_2 \geq s_1,\ x_1 \leq s_2$ is split at $x_2 = s_4 = 0.98$:



Decision Trees

CART for Mixture data set

Finally the region $x_2 \geq s_1,\ x_1 \leq s_2,\ x_2 < s_4$ is split at $x_1 = s_5 = 1$. Resulting in $R_1, R_2, \ldots, R_6$ shown below:



Decision Trees

Splitting criterion

In order to classify well the data, CART seeks as much as possible to obtain pure leaf nodes (i.e. high probability for one class).

At each step CART selects a predictor X_j and a split-point s such that splitting the current region $\mathcal{R}$ into the regions $\mathcal{R}_L(j,s) = \{X|Xj < s\}$ and $\mathcal{R}_R(j,s) = \{X|Xj \geq s\}$ leads to the greatest possible reduction in a well chosen measure of impurity.

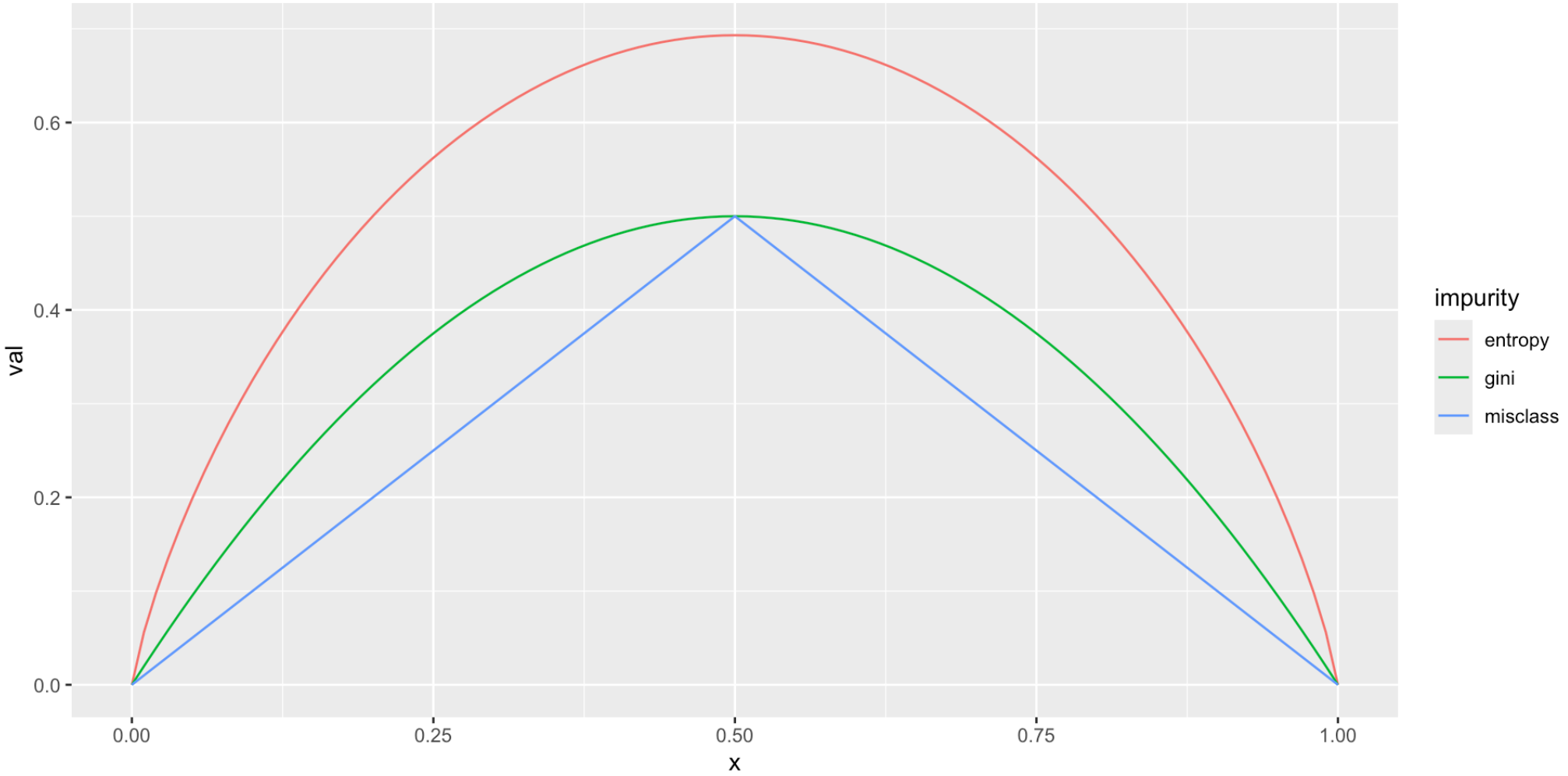
Given a leaf node m representing a region R_m containing n_m observations we denote $\mathcal{I}_m$ a measure of node impurity, three measures are usually retained:

- the misclassification error: $\mathcal{I}_m = \frac{1}{n_m} \sum_{x_i \in R_m} \mathbb{1}_{y_i \neq \hat{C}_m} = 1 - \hat{p}^m_{\hat{C}_m} = 1 - \max(\hat{p}^m, 1 - \hat{p}^m)$ the fraction of observations in the region that do not belong to the most common class
- the Gini index: $\mathcal{I}_m = \sum_k \hat{p}^m_k (1 - \hat{p}^m_k) = 2\hat{p}^m(1 - \hat{p}^m)$
- the cross-entropy or deviance: $\mathcal{I}_m = -\sum_k \hat{p}^m_k \log(1 - \hat{p}^m_k) = -\hat{p}^m \log(\hat{p}^m) - (1 - \hat{p}^m) \log(1 - \hat{p}^m)$

where we have denoted $\hat{p}^m = \hat{p}^m_1 = 1 - \hat{p}^m_0$

Decision Trees

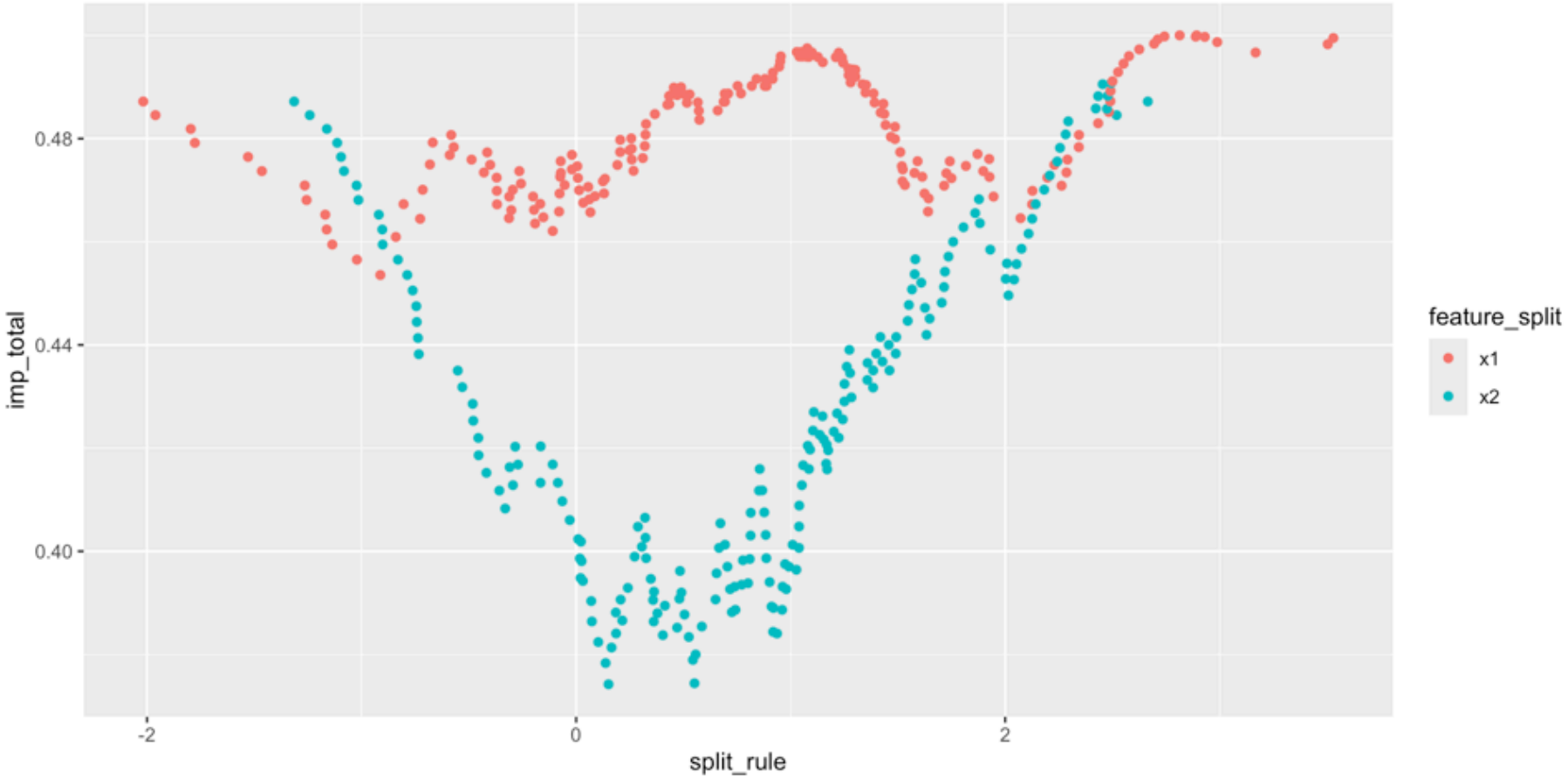
Impurity measures



Decision Trees

A split example

```
# A tibble: 1 × 6
  feature_split split_rule imp_left imp_right imp_total imp_node
  <chr>              <dbl>    <dbl>     <dbl>     <dbl>    <dbl>
1 x2                 0.151    0.142     0.456     0.374      0.5
```



Decision Trees

Exercise

- Using the Desbois data set, build and draw a decision tree using `rpart`.
- Compare a large tree and a smaller tree in terms of ROC/AUC/prediction on a testing set.
- Try to understand the output of the `printcp` function.
- Choose a terminal number of leafs and prune the tree using the `prune` function.
- Select the optimal `cp` for your tree (you can use a plot) and compare to the large and small models in terms of AUC.

Boosting

Boosting

AdaBoost - algorithm

Algorithm 10.1 *AdaBoost.M1.*

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 $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Boosting

AdaBoost - toy example

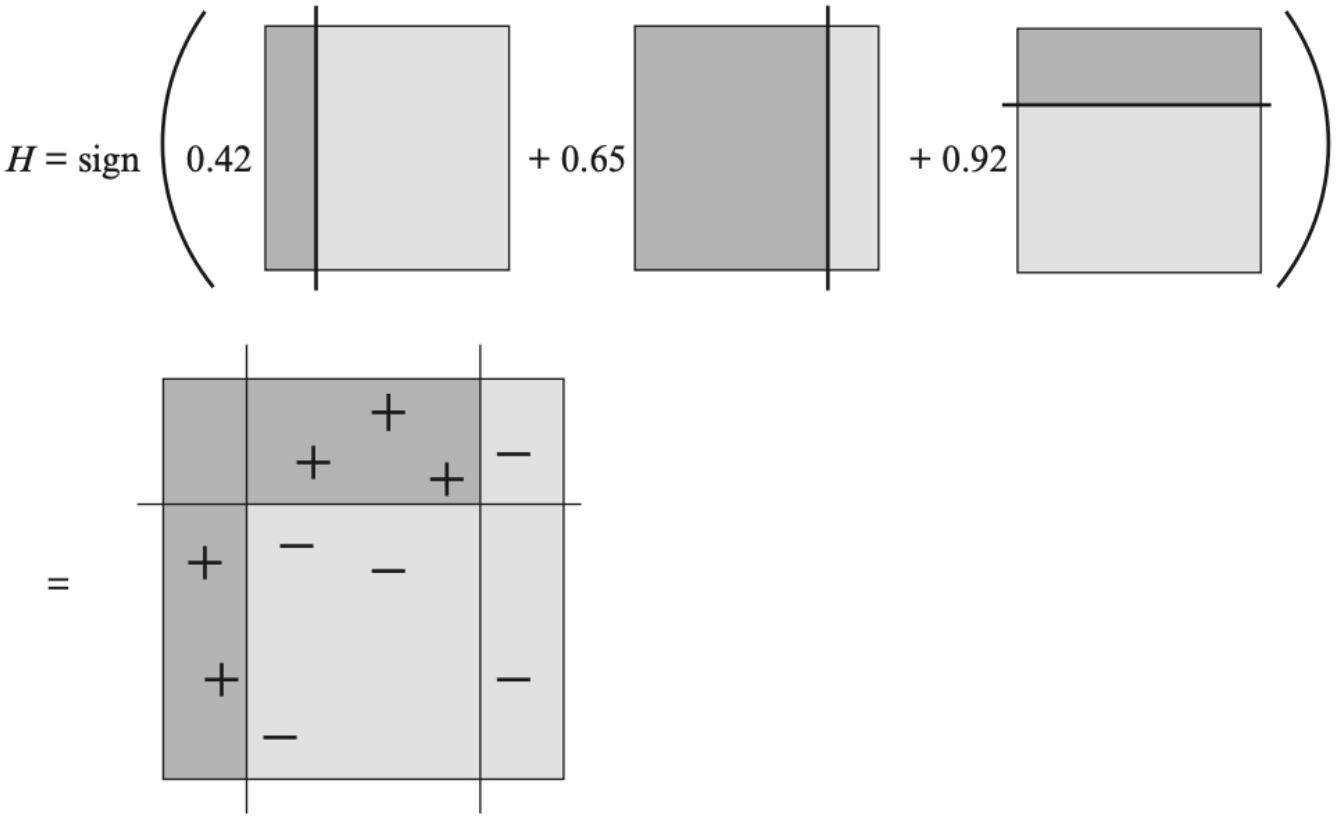
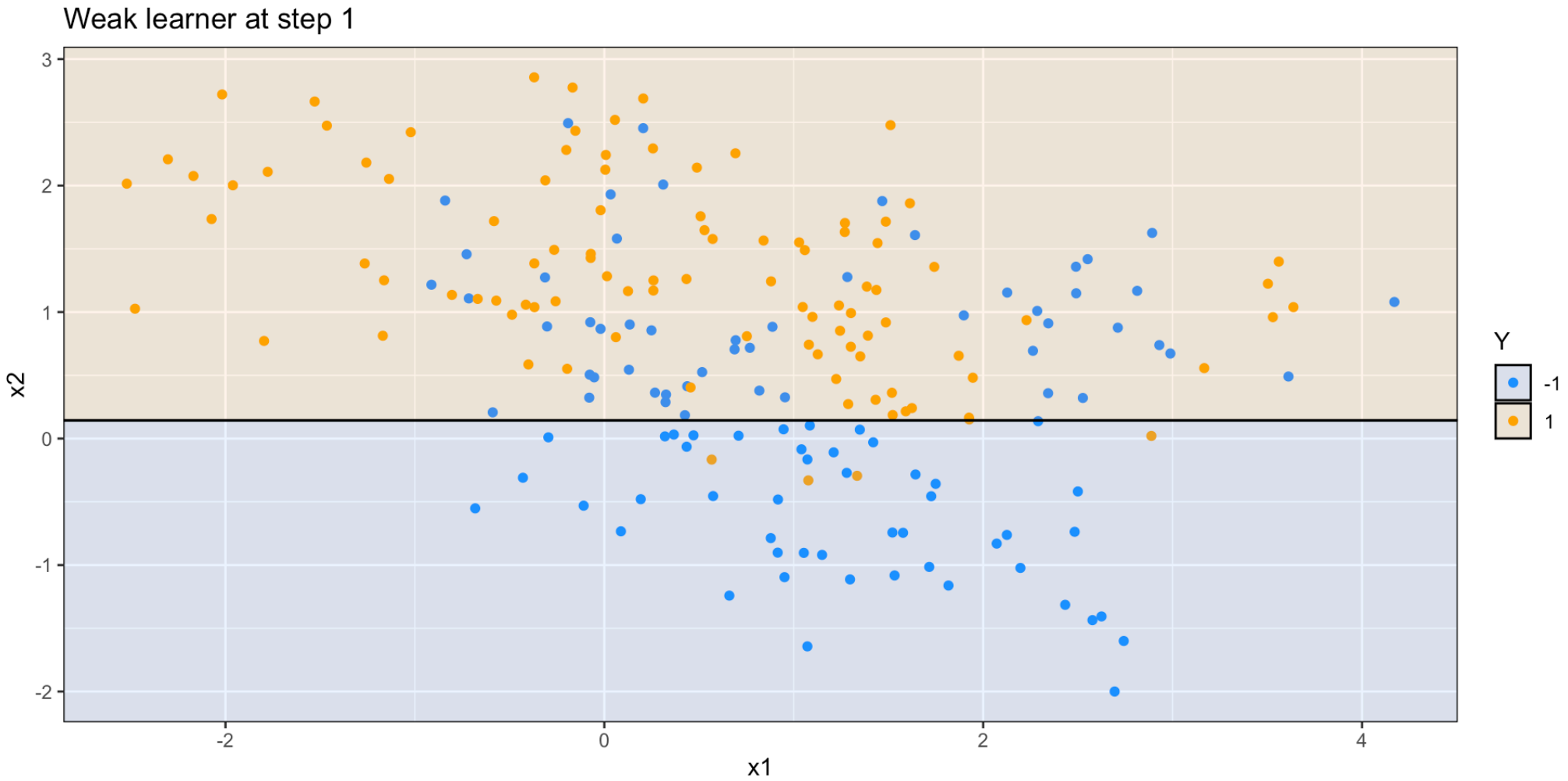


Figure 1.2
The combined classifier for the toy example of figure 1.1 is computed as the sign of the weighted sum of the three weak hypotheses, $\alpha_1 h_1 + \alpha_2 h_2 + \alpha_3 h_3$, as shown at the top. This is equivalent to the classifier shown at the bottom. (As in figure 1.1, the regions that a classifier predicts positive are indicated using darker shading.)

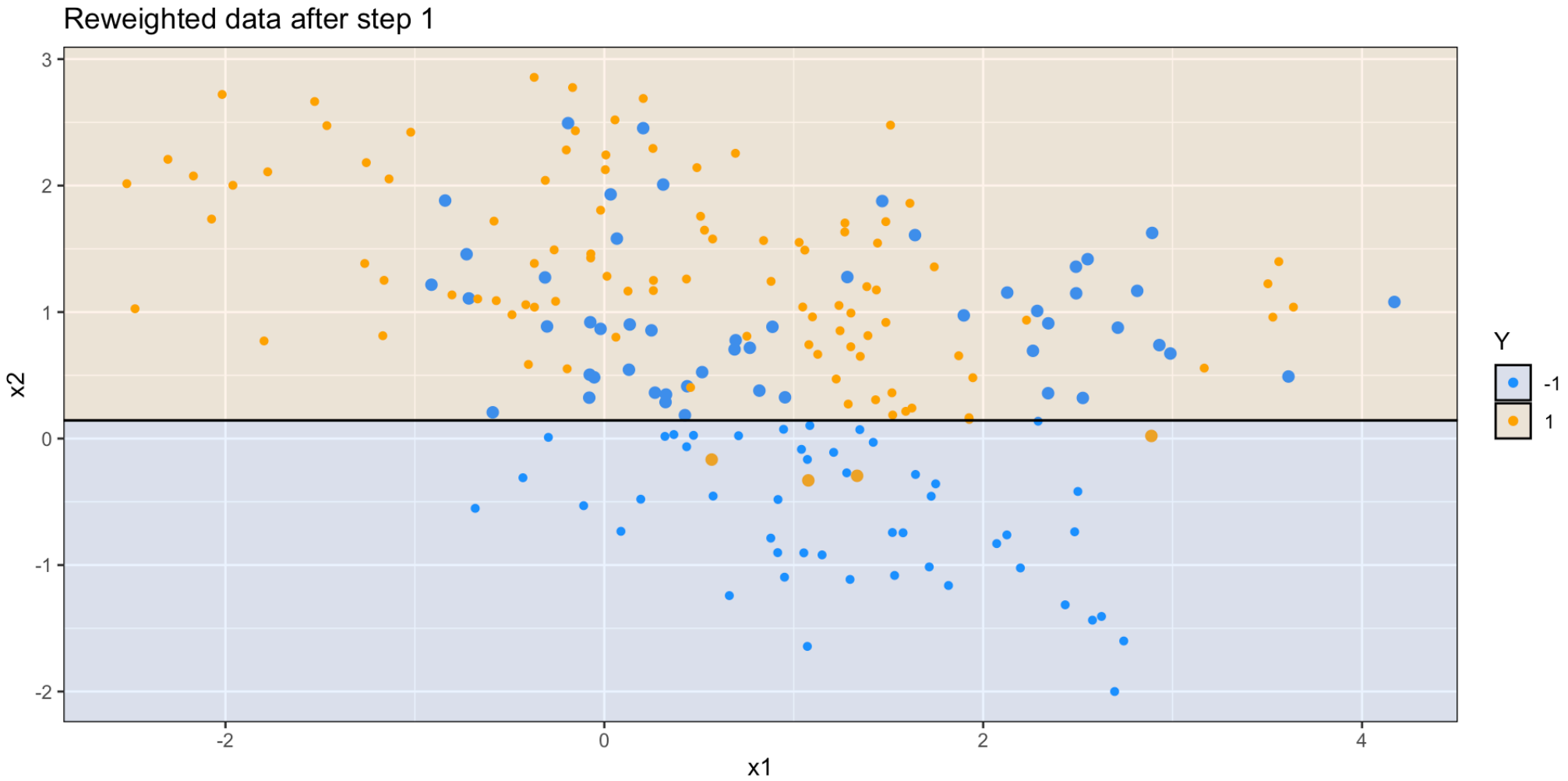
Boosting

AdaBoost - Step 1 (weak learner)



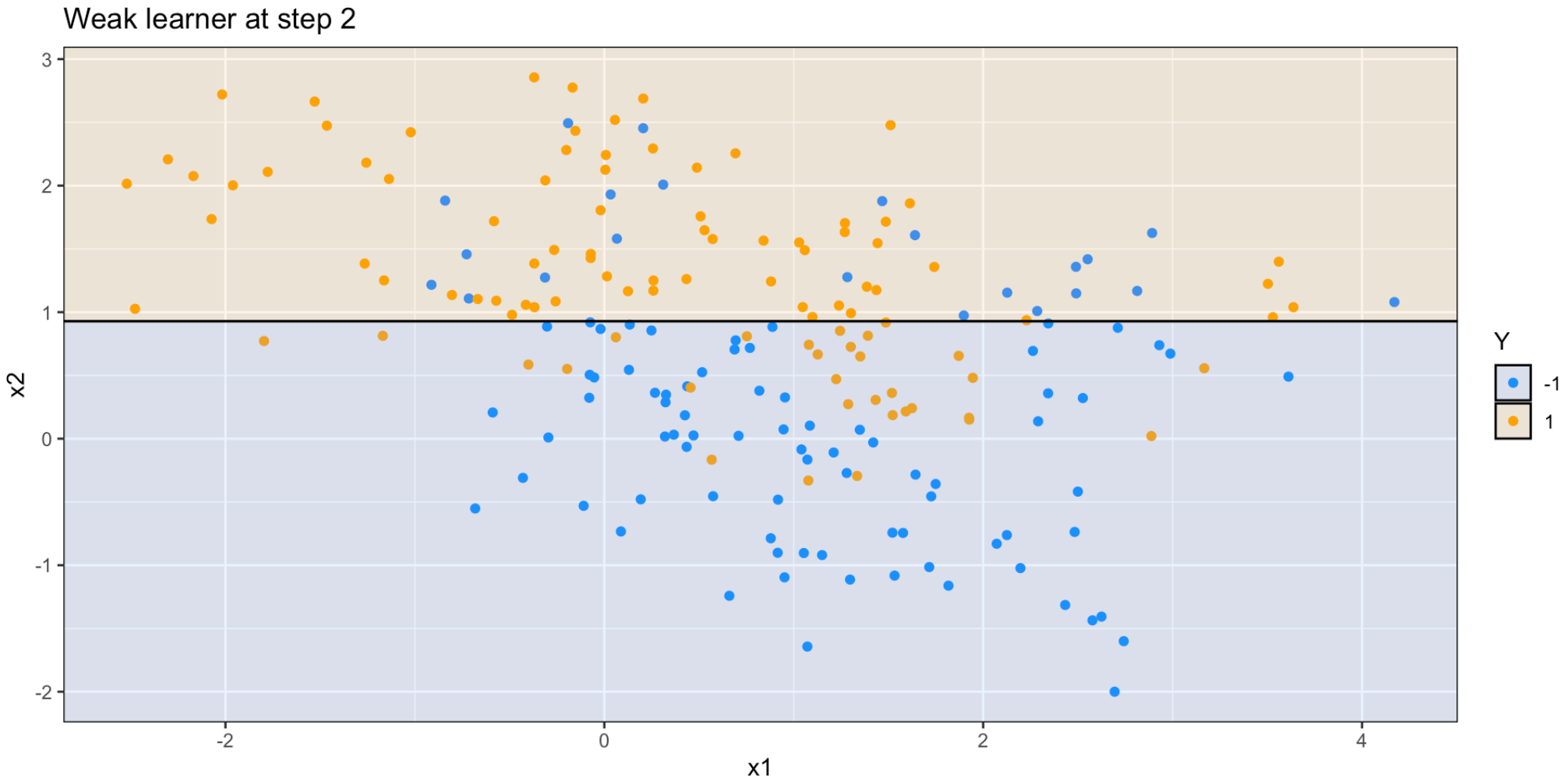
Boosting

AdaBoost - Step 1 (weighted data)



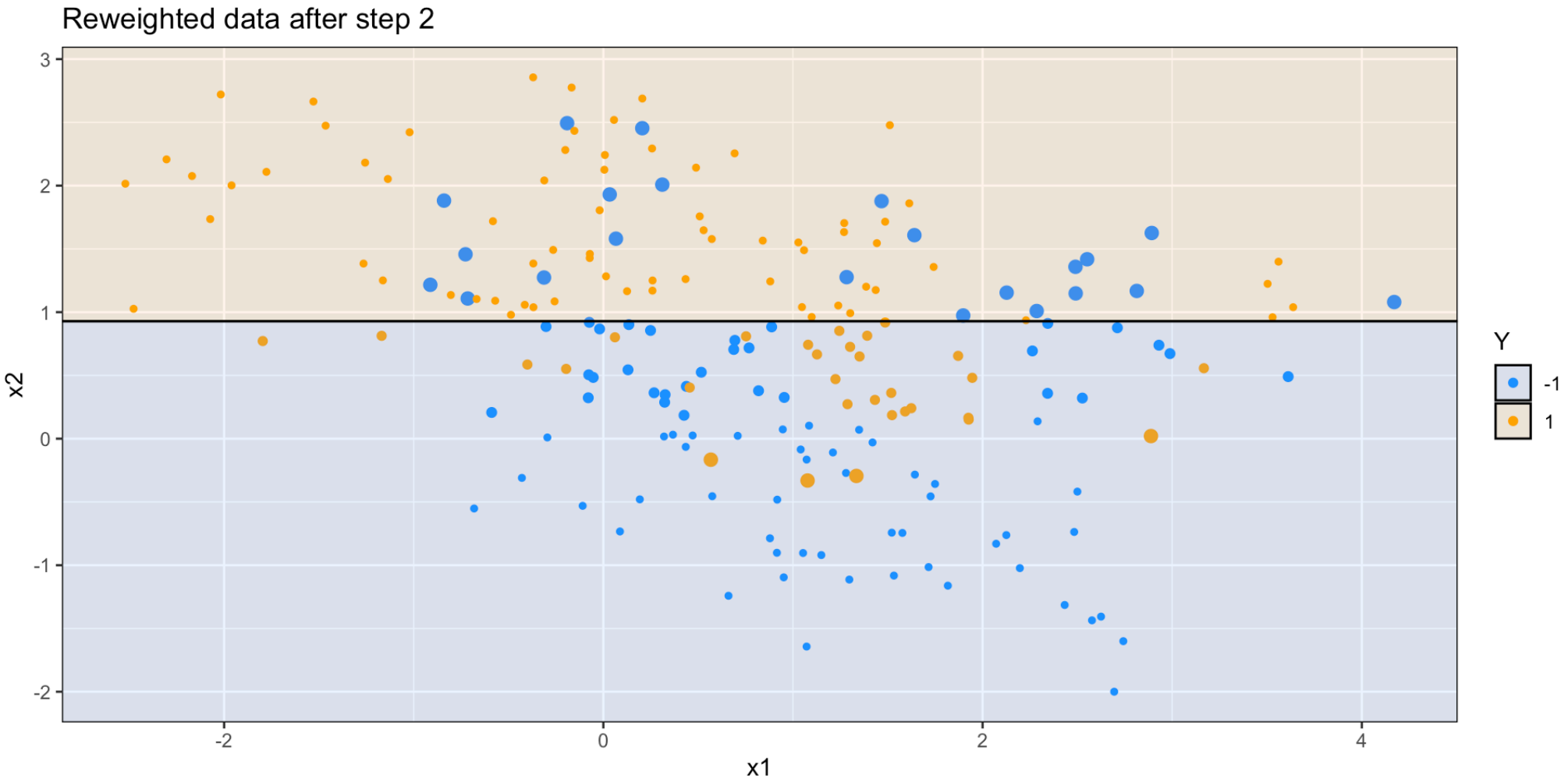
Boosting

AdaBoost - Step 2 (weak learner)



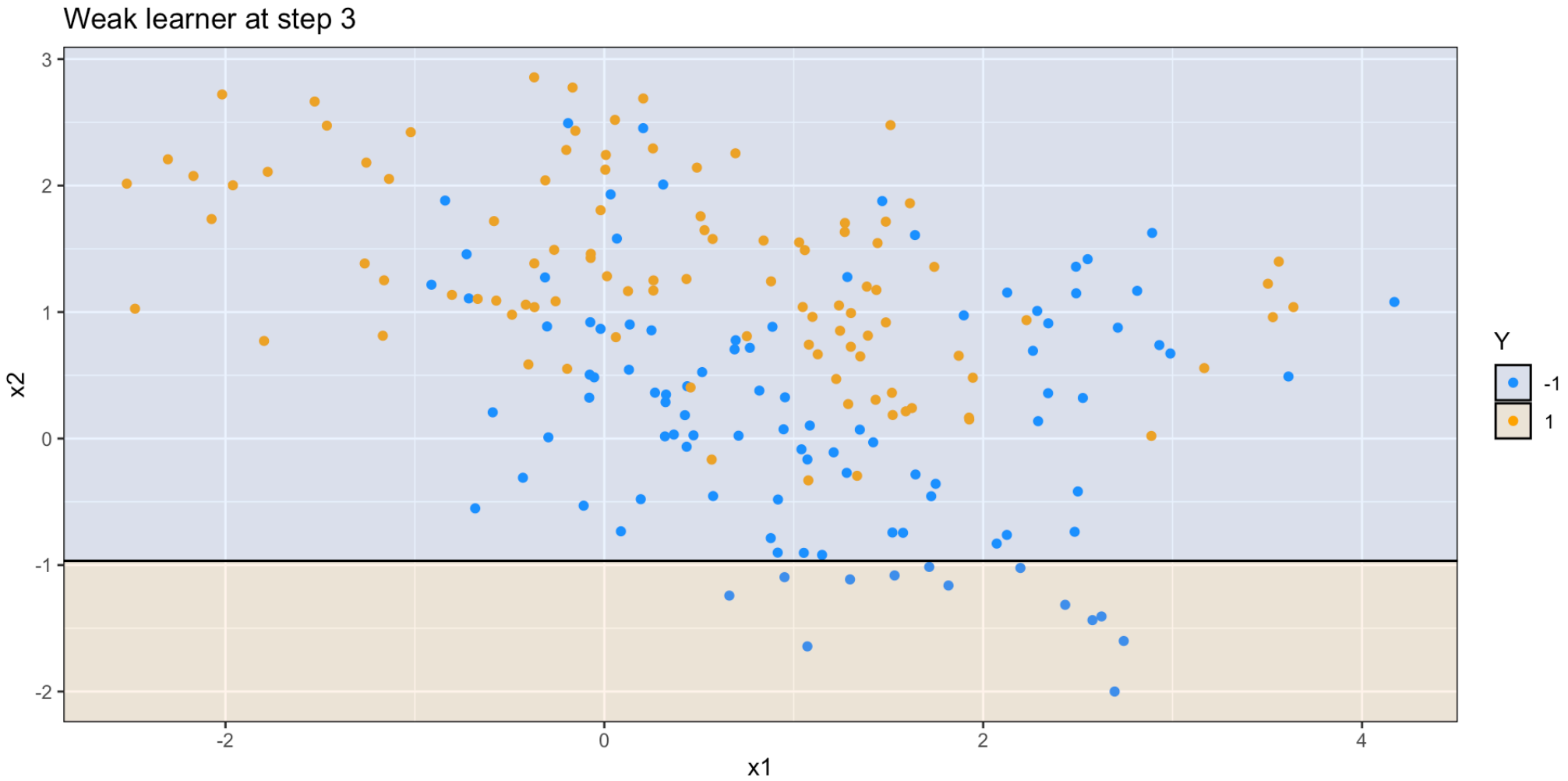
Boosting

AdaBoost - Step 2 (weighted data)



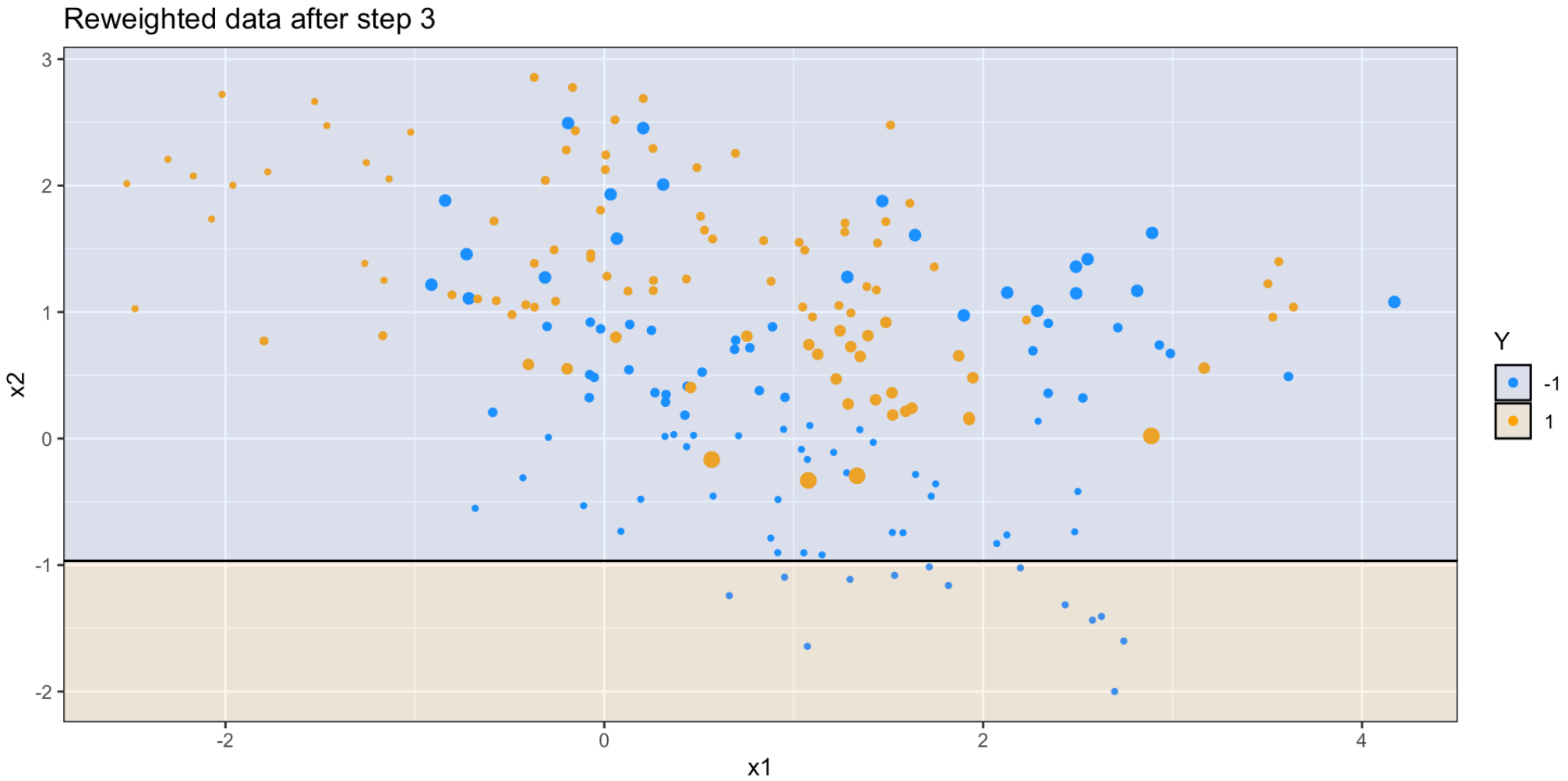
Boosting

AdaBoost - Step 3 (weak learner)



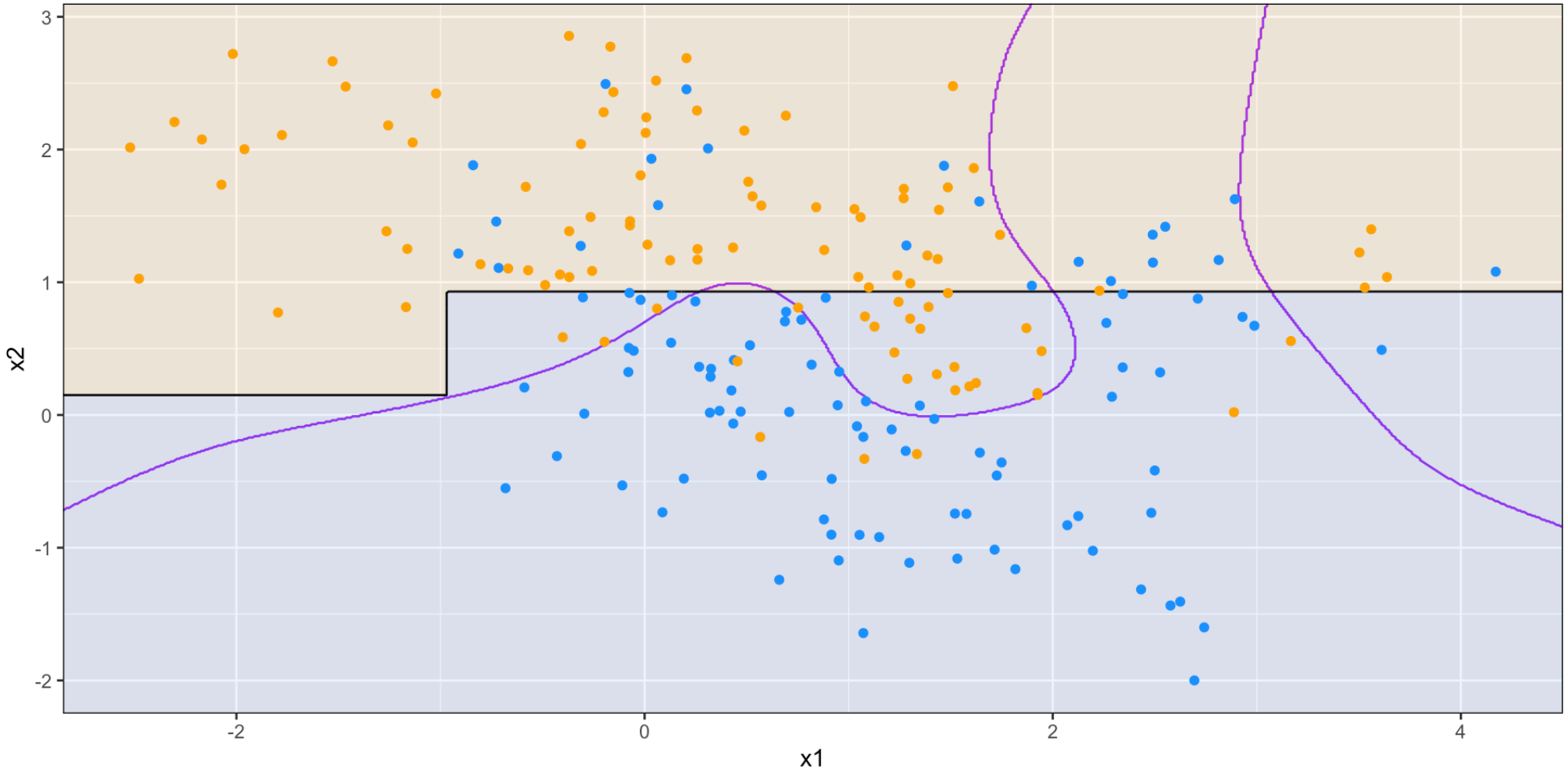
Boosting

AdaBoost - Step 3 (weighted data)



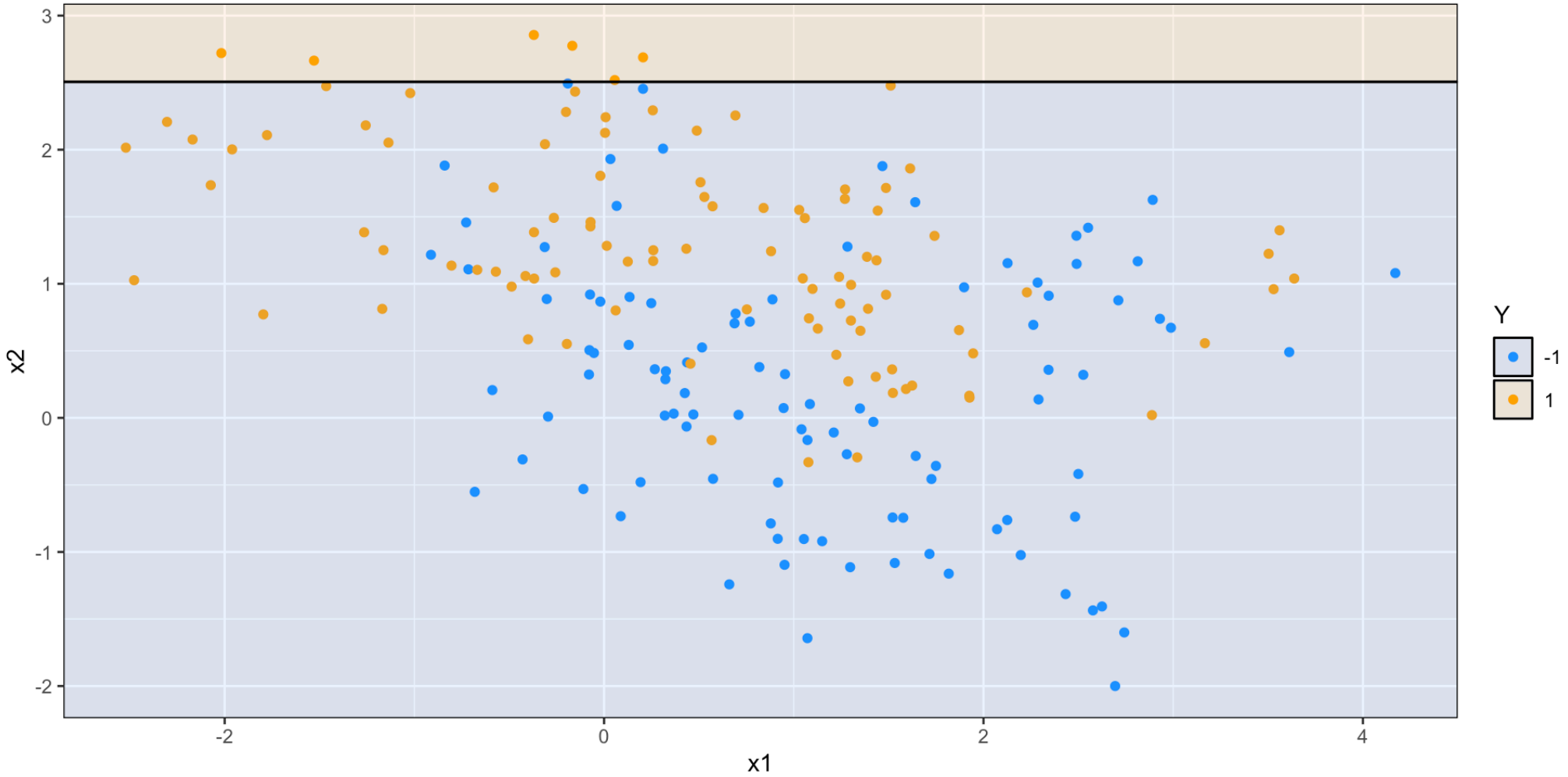
Boosting

AdaBoost - Step 3 (decision)



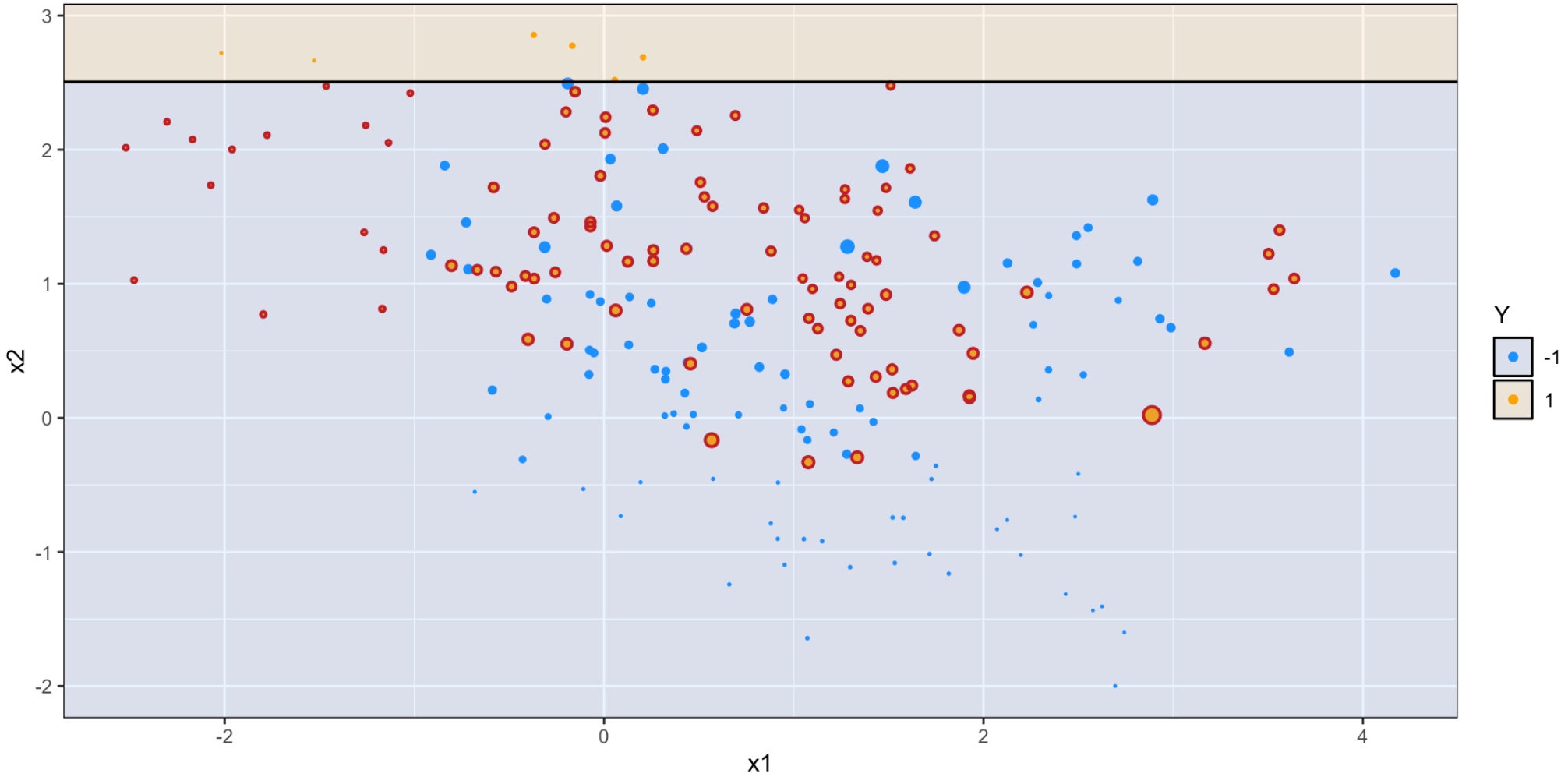
Boosting

AdaBoost - Step 100 (weak learner)



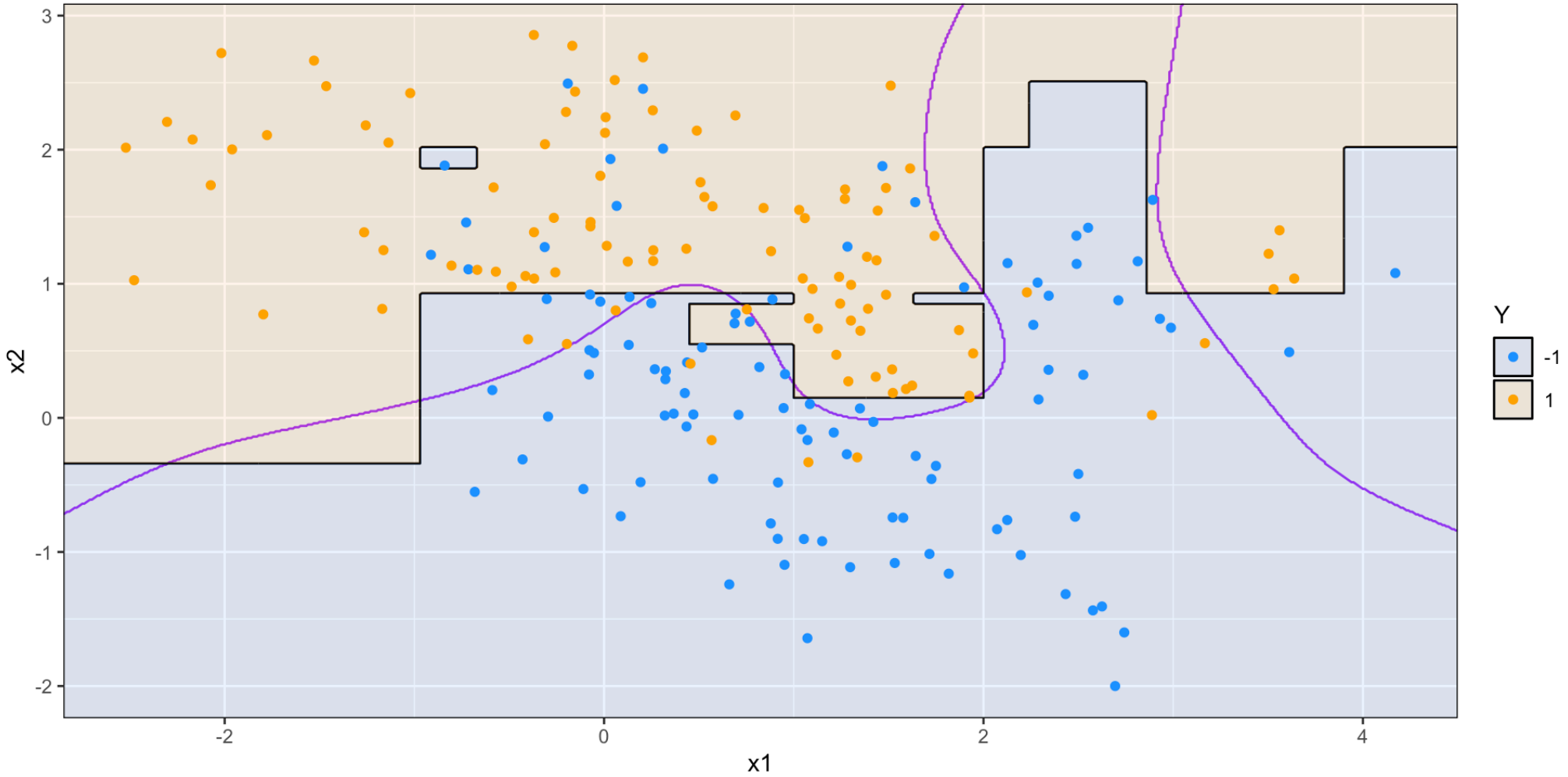
Boosting

AdaBoost - Step 100 (weighted data)



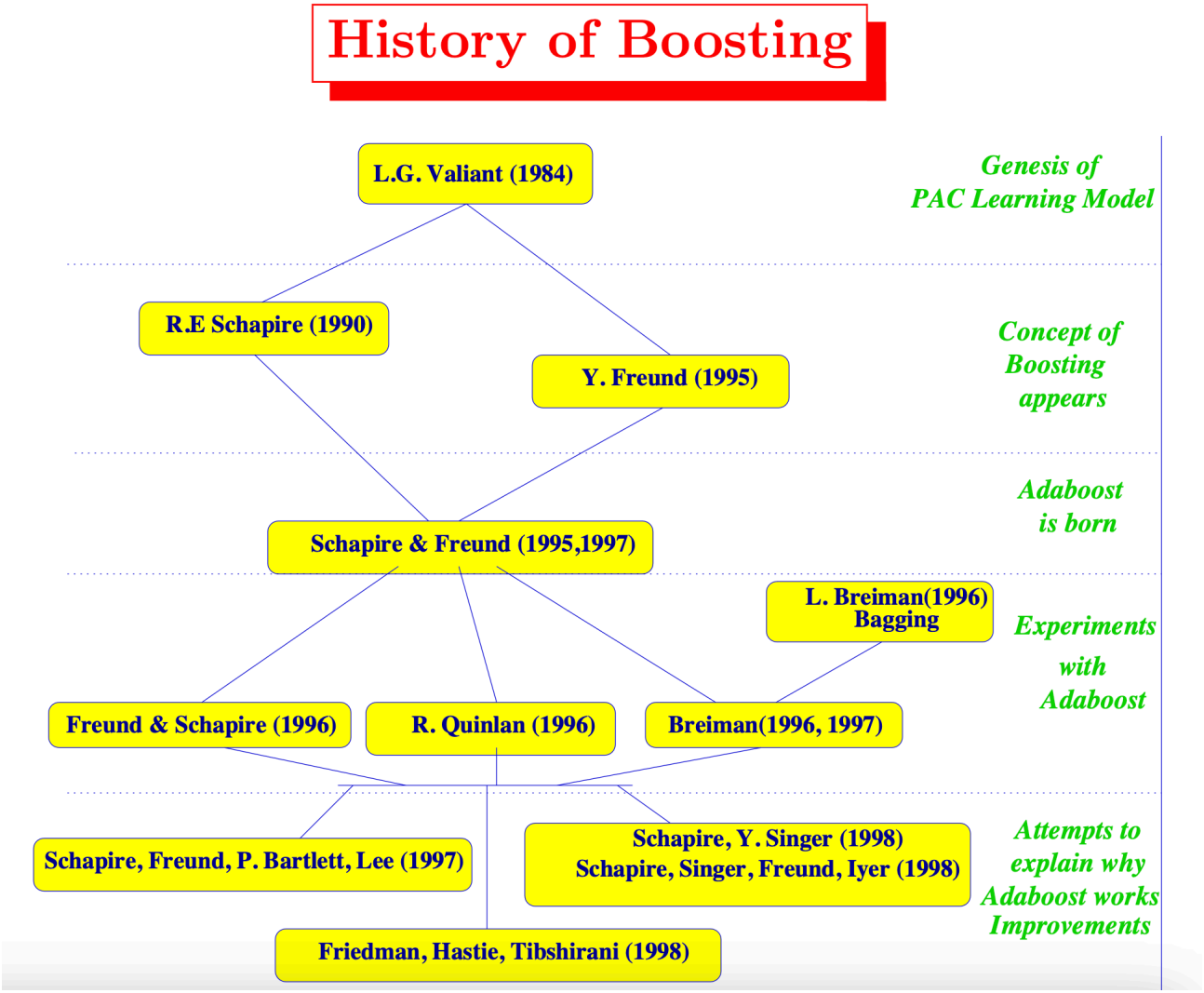
Boosting

AdaBoost - Step 100 (decision)



Boosting

Early history



Boosting

AdaBoost - Statistical view (Friedman et al. (2000))

- AdaBoost fits an additive logistic regression model (FSAM)

$$f^*(x) = \frac{1}{2}\log(\frac{\mathbb{P}[Y=1|X=x)]}{\mathbb{P}[Y=-1|X=x)]}) = \sum_{m=1}^{M} \alpha_m h(x, \gamma_m)$$

- by Forward Stagewise procedure (FSAM) using the exponential loss function:

$$L(y, f(x)) = \exp(-yf(x)) = \exp(-\text{margin})$$

See full details: Handout section “Derivation of AdaBoost: statistical view” or Friedman, J., Hastie, T., & Tibshirani, R. (2000). Additive logistic regression: A statistical view of boosting (Friedman et al. (2000))

Boosting

AdaBoost - FSAM (Friedman et al. (2000))

$$f_M(x) = \sum_{m=1}^{M} \alpha_m h(x, \gamma_m)$$

Additive model: α are the expansion coefficients or weights and $h(x, \gamma_m)$ are the "basis" functions or "learners", usually "simple" functions of x parameterized by γ, such as Decision Trees.

Fitted by minimizing a loss function of the training data $L(y, f(x))$:

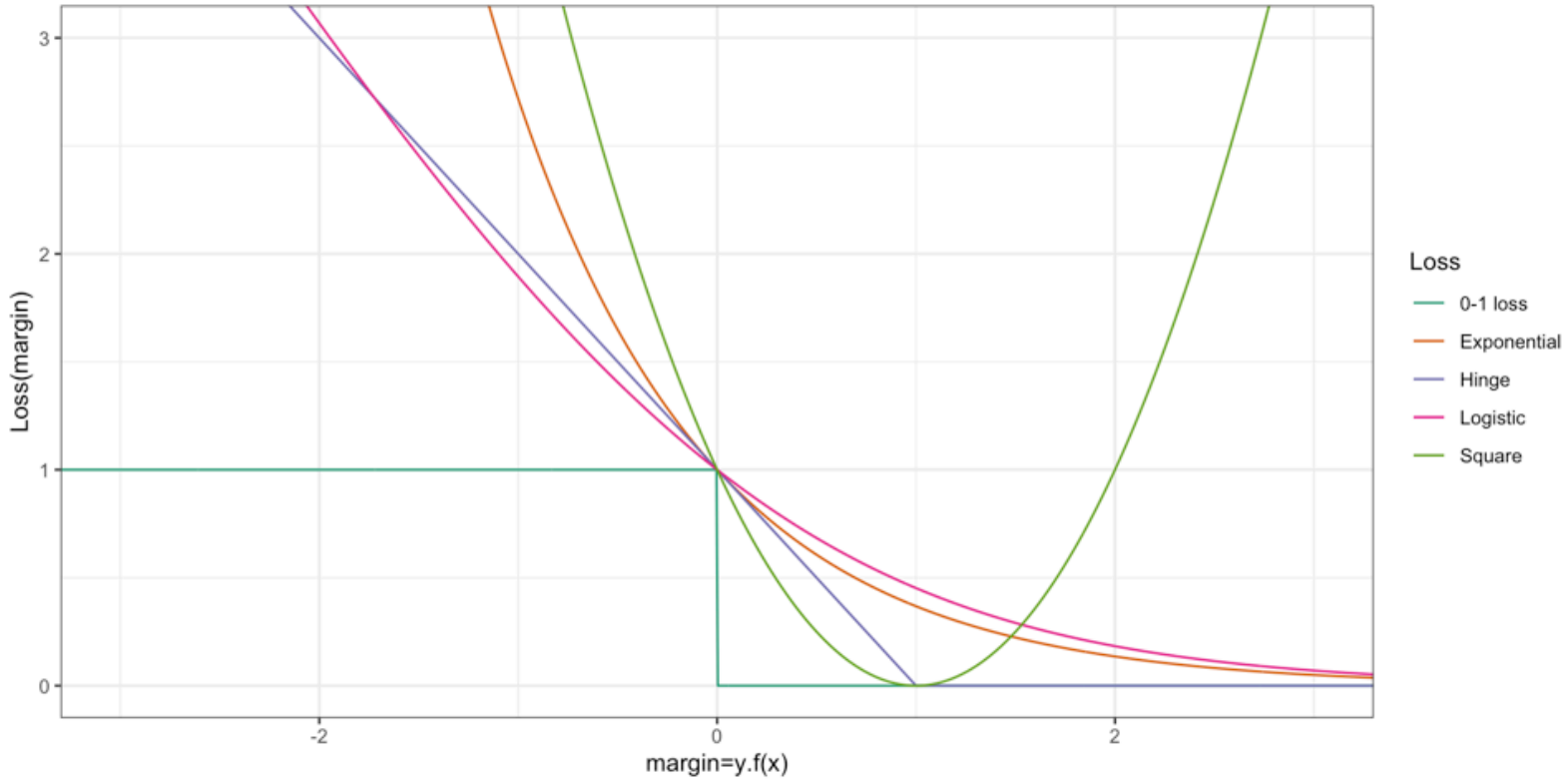
$$\min_{\{\alpha_m, \gamma_m\}_1^M} \sum_{i=1}^{n} L(y_i, \sum_{m=1}^{M} \alpha_m h(x_i, \gamma_m))$$

Heuristic: a "forward stagewise" or "greedy" approach to solve the optimization:

At a given iteration m, the expansion f_{m-1} is fixed and we only optimize the weights α_m and current "learner" parameters γ_m. f_m is obtained and the procedure repeats. Note that in the next step f_m is fixed and we only optimize the weights and parameters for learner $m + 1$.

Boosting

Loss functions (margin)



Replace exponential loss with logistic loss: LogitBoost

See full details: Handout section “Introducing LogitBoost algorithm”

Boosting

LogitBoost algorithm (Friedman et al. (2000))

We remind that we want to estimate an additive model of the form (for a better reading we have simplified the preceding notation $f_m(x) = \alpha_m h(x_i, \gamma_m)$):

$$f^M(x) = \sum_{m=1}^{M} f_m(x)$$

Such additive models are fitted by minimizing a loss function of the training data $\ell(y, f(x))$:

$$\min_{\{f_m\}_1^M} \sum_{i=1}^{n} \ell(y_i, \sum_{m=1}^{M} f_m(x_i))$$

For some losses (ie Mean Squared Error), the preceding expression simplifies and allows for direct solving.

Boosting

LogitBoost algorithm : Taylor expansion (Friedman et al. (2000))

In general to minimize $\mathcal{L}_\ell^t$ we expand the expression up to the second order in f_t:

$$\begin{aligned}\mathcal{L}_L^t &= \sum_{i=1}^n \ell(y_i, f^{t-1}(x_i) + f_t(x_i)) \\ &\approx \sum_{i=1}^n [\ell(y_i, f^{t-1}(x_i)) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] \\ &= \mathcal{L}_\ell^{t-1} + \sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)]\end{aligned}$$

where $g_i = \frac{\partial \ell(y_i, f_i)}{\partial f_i}\Big|_{f_i = f^{t-1}(x_i)}$ and $h_i = \frac{\partial^2 \ell(y_i, f_i)}{\partial^2 f_i}\Big|_{f_i = f^{t-1}(x_i)}$

Boosting

LogitBoost algorithm : Taylor expansion (Friedman et al. (2000))

Completing the square, this rewrites:

$$\begin{aligned}
\mathcal{L}_\ell^t &\approx \mathcal{L}_\ell^{t-1} + \sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] \\
&= \mathcal{L}_\ell^{t-1} + \sum_{i=1}^n \frac{1}{2} h_i [2 \frac{g_i f_t(x_i)}{h_i} + f_t^2(x_i)] \\
&= \mathcal{L}_\ell^{t-1} + \sum_{i=1}^n \frac{1}{2} h_i [\frac{g_i}{h_i} + f_t(x_i)]^2 - (\frac{g_i}{h_i})^2 \\
&= \mathcal{L}_\ell^{t-1} + \sum_{i=1}^n \frac{1}{2} h_i [-\frac{g_i}{h_i} - f_t(x_i)]^2 - (\frac{g_i}{h_i})^2
\end{aligned}$$

Boosting

LogitBoost algorithm : Taylor expansion (Friedman et al. (2000))

Having fixed f^{t-1}, minimizing $\mathcal{L}_\ell^t$ in f_t is equivalent to solve:

$$f_t^* = \underset{f_t}{\operatorname{argmin}} \sum_{i=1}^{n} \frac{1}{2} h_i [-\frac{g_i}{h_i} - f_t(x_i)]^2$$

where we recognize a weighted least square regression problem.

For the logistic loss $\ell_{logistic}(Y, f(x)) = \log(1 + e^{-2Yf(x)})$ we have

$g_i = \frac{\partial \ell(y_i, f_i)}{\partial f_i}\Big|_{f_i = f(x_i)} = 2(y_i - \frac{1}{1+exp(-2f(x_i))}) = 2(y_i - p(x_i))$ and
$h_i = \frac{\partial^2 \ell(y_i, f_i)}{\partial^2 f_i}\Big|_{f_i = f(x_i)} = 4 \times p(x_i)(1 - p(x_i))$ having denoted $p(x) = \frac{e^{f(x)}}{e^{f(x)} + e^{-f(x)}} = \frac{1}{1+e^{-2f(x)}}$

Boosting

LogitBoost algorithm (Friedman et al. (2000))

LogitBoost (two classes)

1. Start with weights $w_i = 1/N$ $i = 1, 2, \ldots, N$, $F(x) = 0$ and probability estimates $p(x_i) = \frac{1}{2}$.
2. Repeat for $m = 1, 2, \ldots, M$:

   (a) Compute the working response and weights

$$z_i = \frac{y_i^* - p(x_i)}{p(x_i)(1 - p(x_i))},$$

$$w_i = p(x_i)(1 - p(x_i)).$$

   (b) Fit the function $f_m(x)$ by a weighted least-squares regression of z_i to x_i using weights w_i.

   (c) Update $F(x) \leftarrow F(x) + \frac{1}{2} f_m(x)$ and $p(x) \leftarrow (e^{F(x)})/(e^{F(x)} + e^{-F(x)})$.

3. Output the classifier $\text{sign}[F(x)] = \text{sign}[\sum_{m=1}^{M} f_m(x)]$.

ALGORITHM 3. *An adaptive Newton algorithm for fitting an additive logistic regression model.*

Boosting

LogitBoost - Weighted least square / Newton method (Friedman et al. (2000))

RESULT 3. *The LogitBoost algorithm (two classes, population version) uses Newton steps for fitting an additive symmetric logistic model by maximum likelihood.*

DERIVATION. Consider the update $F(x) + f(x)$ and the expected log-likelihood

$$El(F+f) = E\left[2y^*(F(x)+f(x)) - \log[1+e^{2(F(x)+f(x))}\right]. \tag{31}$$

Conditioning on x, we compute the first and second derivative at $f(x) = 0$,

$$\begin{aligned} s(x) &= \frac{\partial El(F(x)+f(x))}{\partial f(x)}\bigg|_{f(x)=0} \\ &= 2E(y^* - p(x)|x), \end{aligned} \tag{32}$$

$$\begin{aligned} H(x) &= \frac{\partial^2 El(F(x)+f(x))}{\partial f(x)^2}\bigg|_{f(x)=0} \\ &= -4E(p(x)(1-p(x))|x), \end{aligned} \tag{33}$$

Newton step, hence the term (Gradient/Newton) Boosting.

Numerical issues due to dividing by $p(x_i)(1-p)(x_i)$

Boosting

(Robust)LogitBoost - Boosted trees (Ping (2010))

2.2 The Robust Logitboost Algorithm

Algorithm 2 *Robust logitboost*, which is very similar to Friedman's *mart* algorithm [7], except for Line 4.

1: $F_{i,k} = 0, p_{i,k} = \frac{1}{K}, k = 0 \text{ to } K-1, i = 1 \text{ to } N$
2: For $m = 1$ to M Do
3: For $k = 0$ to $K-1$ Do
4: $\{R_{j,k,m}\}_{j=1}^{J} = J$-terminal node regression tree from
: $\{r_{i,k} - p_{i,k},\ \mathbf{x}_i\}_{i=1}^{N}$, with weights $p_{i,k}(1-p_{i,k})$ as in (7)
5: $\beta_{j,k,m} = \frac{K-1}{K} \frac{\sum_{\mathbf{x}_i \in R_{j,k,m}} r_{i,k} - p_{i,k}}{\sum_{\mathbf{x}_i \in R_{j,k,m}} (1-p_{i,k}) p_{i,k}}$
6: $F_{i,k} = F_{i,k} + \nu \sum_{j=1}^{J} \beta_{j,k,m} 1_{\mathbf{x}_i \in R_{j,k,m}}$
7: End
8: $p_{i,k} = \exp(F_{i,k}) / \sum_{s=0}^{K-1} \exp(F_{i,s})$
9: End

Alg. 2 describes *robust logitboost* using the tree-split criterion (7). Note that after trees are constructed, the values of the terminal nodes are computed by

$$\frac{\sum_{node} z_{i,k} w_{i,k}}{\sum_{node} w_{i,k}} = \frac{\sum_{node} r_{i,k} - p_{i,k}}{\sum_{node} p_{i,k}(1-p_{i,k})}, \qquad (8)$$

which explains Line 5 of Alg. 2.

Boosting

(Robust)LogitBoost - specific Decision tree criterion (Ping (2010))

$$Gain(s) = \left[-\bar{z}\sum_{i=1}^{N} z_i w_i\right] - \left[-\bar{z}_L \sum_{i=1}^{s} z_i w_i\right] - \left[-\bar{z}_R \sum_{i=s+1}^{N} z_i w_i\right]$$
$$= \frac{\left[\sum_{i=1}^{s} z_i w_i\right]^2}{\sum_{i=1}^{s} w_i} + \frac{\left[\sum_{i=s+1}^{N} z_i w_i\right]^2}{\sum_{i=s+1}^{N} w_i} - \frac{\left[\sum_{i=1}^{N} z_i w_i\right]^2}{\sum_{i=1}^{N} w_i}$$

Plugging in $w_i = p_{i,k}(1-p_{i,k})$, $z_i = \frac{r_{i,k}-p_{i,k}}{p_{i,k}(1-p_{i,k})}$ **yields**

$$Gain(s) = \frac{\left[\sum_{i=1}^{s}(r_{i,k}-p_{i,k})\right]^2}{\sum_{i=1}^{s} p_{i,k}(1-p_{i,k})} + \frac{\left[\sum_{i=s+1}^{N}(r_{i,k}-p_{i,k})\right]^2}{\sum_{i=s+1}^{N} p_{i,k}(1-p_{i,k})}$$
$$- \frac{\left[\sum_{i=1}^{N}(r_{i,k}-p_{i,k})\right]^2}{\sum_{i=1}^{N} p_{i,k}(1-p_{i,k})}. \qquad (7)$$

Boosting

xgboost - equivalent criterion (Chen & Guestrin (2016))

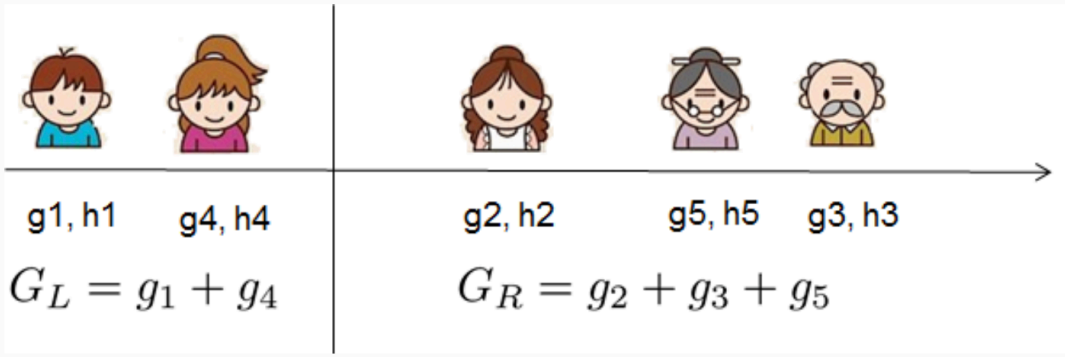
xgboost.readthedocs.io

Now that we have a way to measure how good a tree is, ideally we would enumerate all possible trees and pick the best one. In practice this is intractable, so we will try to optimize one level of the tree at a time. Specifically we try to split a leaf into two leaves, and the score it gains is

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

This formula can be decomposed as 1) the score on the new left leaf 2) the score on the new right leaf 3) The score on the original leaf 4) regularization on the additional leaf. We can see an important fact here: if the gain is smaller than γ, we would do better not to add that branch. This is exactly the **pruning** techniques in tree based models! By using the principles of supervised learning, we can naturally come up with the reason these techniques work :)

For real valued data, we usually want to search for an optimal split. To efficiently do so, we place all the instances in sorted order, like the following picture.



Boosting

More in the Handout document

References

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. and. (1983). *Classification and regression trees*. Wadsworth.

Chen, T., & Guestrin, C. (2016). *XGBoost: A scalable tree boosting system*. 785–794. https://doi.org/10.1145/2939672.2939785

Friedman, J., Hastie, T., & Tibshirani, R. (2000). Additive logistic regression: A statistical view of boosting (with discussion and a rejoinder by the authors). *The Annals of Statistics*, *28*(2), 337–407. https://doi.org/10.1214/aos/1016218223

Ping, L. (2010). *Robust logitboost and adaptive base class (ABC) logitboost*. 302–311. https://doi.org/10.5555/3023549.3023585