Artificial Intelligence

UNI FREIBURG

Albert-Ludwigs-Universität Freiburg

Thorsten Schmidt

Abteilung für Mathematische Stochastik

www.stochastik.uni-freiburg.de
thorsten.schmidt@stochastik.uni-freiburg.de
SS 2017

Our goal today

Support vector machines (continued)

- Multiple classes
- Virtual Support Vector Machines

Adaptive basis function models

- CART
- Hierarchical mixtures of experts

Literature (incomplete, but growing):

- I. Goodfellow, Y. Bengio und A. Courville (2016). **Deep Learning**. http://www.deeplearningbook.org. MIT Press
- D. Barber (2012). **Bayesian Reasoning and Machine Learning**. Cambridge University Press
- R. S. Sutton und A. G. Barto (1998). **Reinforcement Learning : An Introduction**. MIT Press
- G. James u. a. (2014). **An Introduction to Statistical Learning: With Applications in R**. Springer Publishing Company, Incorporated. ISBN: 1461471370, 9781461471370
- T. Hastie, R. Tibshirani und J. Friedman (2009). **The Elements of Statistical Learning**. Springer Series in Statistics. Springer New York Inc. URL: https://statweb.stanford.edu/~tibs/ElemStatLearn/
- K. P. Murphy (2012). **Machine Learning: A Probabilistic Perspective**. MIT Press
- CRAN Task View: Machine Learning, available at https://cran.r-project.org/web/views/MachineLearning.html
- UCI ML Repository: http://archive.ics.uci.edu/ml/ (371 datasets)

The multi-class problem

Actually, in the previous example we did have **10** classes and not only 1!

- The method is actually not designed to cope with more than one classes.
- Some heuristic methods have been proposed to overcome this deficiency: **one-vs-one** or **one-vs-all** comparison.

One-versus-one Classification

Think you have K classes and data points $(x_i, y_i)_{i=1}^n$. Then you divide in the $K(K-1)/2$ classification problems (x_i, z_i^{kl}),

$$z_i^k = \begin{cases} 1 & \text{if } y_i = k, \\ -1 & \text{if } y_i = l. \end{cases}$$

The decision rule is to assign those class which appears most often in the single SVMs.

One-versus-all Classification

Think you have K classes and data points $(x_i, y_i)_{i=1}^n$. Then you divide in the K classification problems (x_i, z_i^k),

$$z_i^k = \begin{cases} 1 & \text{if } y_i = k, \\ -1 & \text{otherwise.} \end{cases}$$

Denote the estimated vectors in these SVMs by $\beta_1, \ldots, beta_k$ the result of the fits. For a test observation x we choose the class k for which $\beta_k x$ is largest.

As is obvious, both methods have their difficulties in multi-class classification.

Both approaches try to find a measure for the quality of the estimation. One also would like to find something like a probability relating to the fit and a common approach is to compute

$$\sigma(a\hat{\beta}x+b)$$

where σ is the sigmoid function and a and b are estimated by maximum-likelihood. As commented in ebd., page 504, this has difficulties in general (there is no reason why $\hat{\beta}x$ leads to a probability).

Virtual Support Vector Machines

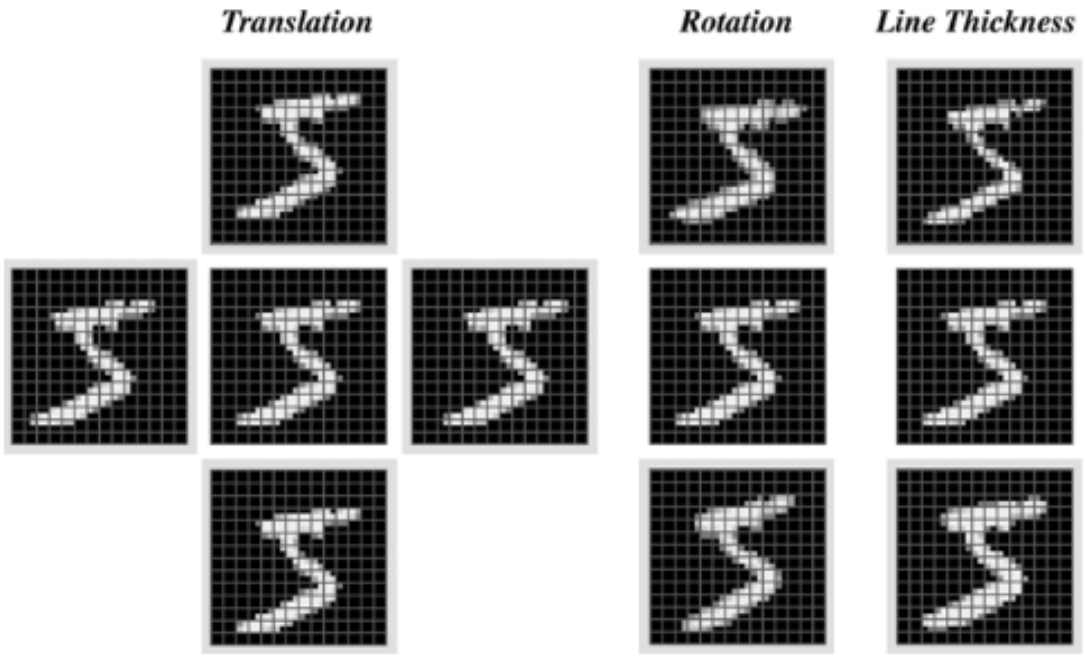
- The record-holding SVM on the MNIST database uses a **Virtual** SVM, see D. Decoste und B. Schölkopf (2002). „Training invariant support vector machines“. In: **Machine learning** 46.1, S. 161–190.
- We cite: ”Practical experience has shown that in order to obtain the best possible performance, prior knowledge about invariances of a classification problem at hand ought to be incorporated into the training procedure”
- Indeed, currently we did not take this into account at all, but simply trained the data. So how could this be done ?

- One could try to find optimal kernel functions just tailor-made to the problem.
- One could generate **virtual** examples from the training set by applying some standard transformations.
- A combination of these approaches.

For the MNIST digits one can use for example translation, rotation or in-/decrease in line thickness.

The consequence is that much more support vectors are found, suggesting that the methods improves the estimation. The optimal choice with error rate of **0.56%** uses a polynomial kernel of degree 9

$$K(x,y) = \frac{1}{512}(x \cdot y + 1)^9.$$



Source: Decoste & Schölkopf (2002)

Table 1. Comparison of Support Vector sets and performance for training on the original database and training on the generated Virtual Support Vectors. In both training runs, we used polynomial classifier of degree 3.

Classifir trained on	Size	Av. no. of SVs	Test error
Full training set	7291	274	4.0%
Overall SV set	1677	268	4.1%
Virtual SV set	8385	686	3.2%
Virtual patterns from full DB	36455	719	3.4%

Virtual Support Vectors were generated by simply shifting the images by one pixel in the four principal directions. Adding the unchanged Support Vectors, this leads to a training set of the second classifier which has five times the size of the first classifier's overall Support Vector set (i.e. the union of the 10 Support Vector sets of the binary classifiers, of size 1677—note that due to some overlap, this is smaller than the sum of the ten support set sizes). Note that training on virtual patterns generated from *all* training examples does not lead to better results han in the Virtual SV case; moreover, although the training set in this case is much larger, it hardly leads to more SVs.

Source: Decoste & Schölkopf (2002)

Recall that number 9 was misclassified (6 instead of 5)

```
test=attr(pred,"decision.values")[9,]
sign(test)

5/0 5/4 5/1 5/9 5/2 5/3 5/6 5/7 5/8 (...)
 -1   1   1   1  -1   1  -1   1   1  (...)
```

While 6 gets all ones.

We try an alternative statistic (now for 20.000 data):

```
testnorm=test/max(test)
labels1=c("5/0","5/1","5/2","5/3","5/4","5/6","5/7","5/8","5/9")
labels2=c("0/6","1/6","2/6","3/6","4/6","5/6","6/7","6/8","9/6")

> testnorm[labels1]
       5/0         5/1         5/2         5/3         5/4         5/6         5/7         5/8         5/9
 0.04257473  0.21475787  0.01911578  0.77776091  0.03457997 -0.28087561  0.08408199  0.25468632  0.15987355
> factors*testnorm[labels2]
       0/6        1/6        2/6        3/6        4/6        5/6        6/7        6/8        9/6
0.08472909 0.26445613 0.37370402 0.19366219 0.16418205 0.28087561 0.15263579 0.29240588 0.06712401
```

Adaptive basis function models

- As we saw in the previous chapter, a common classification approach is to consider a prediction of the form

$$f(x) = \beta\phi(x)$$

where ϕ is determined via a kernel, such that $\phi(v) = (\kappa(v,x_1),\dots(\kappa(v,x_n))$ with data points $x_1,\dots,x_n$.

- The question is how to obtain an optimal kernel and how to estimate the associated parameters.
- An alternative is to learn ϕ directly from the data ! This is done in the so-called ABMs (adaptive basis function models): the starting point is to study

$$f(x) = w_0 + \sum_{m=1}^{M} w_m\phi_m(x)$$

with weights $w_0,\dots,w_M$ and **basis functions** $\phi_1,\dots,\phi_M$.

- A typical approach is a parametric specification of the kernel function, i.e.

$$\phi_m(x) = \phi(x, v_m)$$

where $v_1, \ldots, v_M$ are the parameters of the kernel. The entire parameter set is denoted by $\theta := (w_0, \ldots, w_M, v_1, \ldots, v_M)$.

CART

- The first example will be **classification and regression trees** (CART).
- A **classification tree** has as an output classes and a **regression tree** gives real numbers instead.
- The idea is to partition the data suitably. Most commonly are half-planes i.e. we use as classification rules

$$\mathbb{1}_{\{x \leq t\}}$$

and various combinations of thereof. The classficiation in this case is the partition given by

$$\{x \in \mathbb{R} : x \leq t\} \quad \text{versus} \quad \{x \in \mathbb{R} : x > t\}$$

and leads to a **binomial tree**.

We study the famous example of Titanic passenger data. In fact this dataset is included in the R package `rpart.plot` and contains 1046 datapoints with observations on the passenger class, survival, sex, age, sibsp (number of spouses or siblings aboard), parch (number of parents or children aboard).

```
> data(ptitanic)
> summary(ptitanic)
 pclass        survived       sex           age              sibsp           parch
 1st:323   died    :809   female:466   Min.   : 0.1667   Min.   :0.0000   Min.   :0.000
 2nd:277   survived:500   male  :843   1st Qu.:21.0000   1st Qu.:0.0000   1st Qu.:0.000
 3rd:709                              Median :28.0000   Median :0.0000   Median :0.000
                                      Mean   :29.8811   Mean   :0.4989   Mean   :0.385
                                      3rd Qu.:39.0000   3rd Qu.:1.0000   3rd Qu.:0.000
                                      Max.   :80.0000   Max.   :8.0000   Max.   :9.000
                                      NA's   :263
```

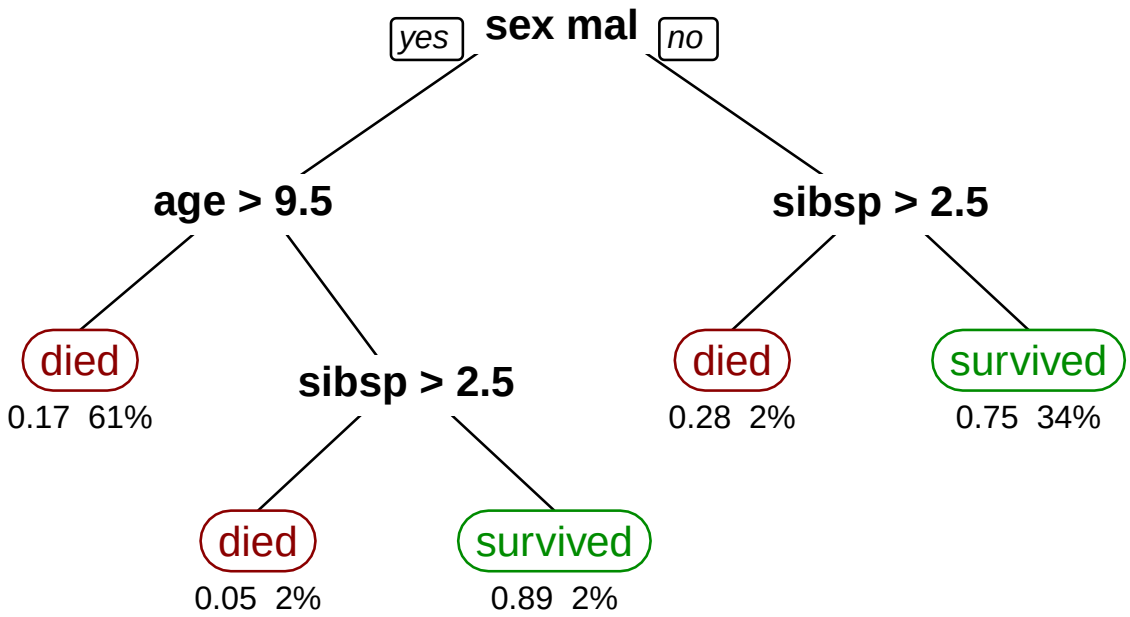
A classification tree can be obtained as follows:

```
library(rpart)
library(rpart.plot)
data(ptitanic)

fit = rpart(survived ~  sex + age + sibsp, data=ptitanic, method="class")
cols <- c("darkred", "green4")[fit$frame$yval]
prp(fit,tweak=1.4 , extra=106, under=TRUE, ge=" > ", eq=" ", col=cols)
```

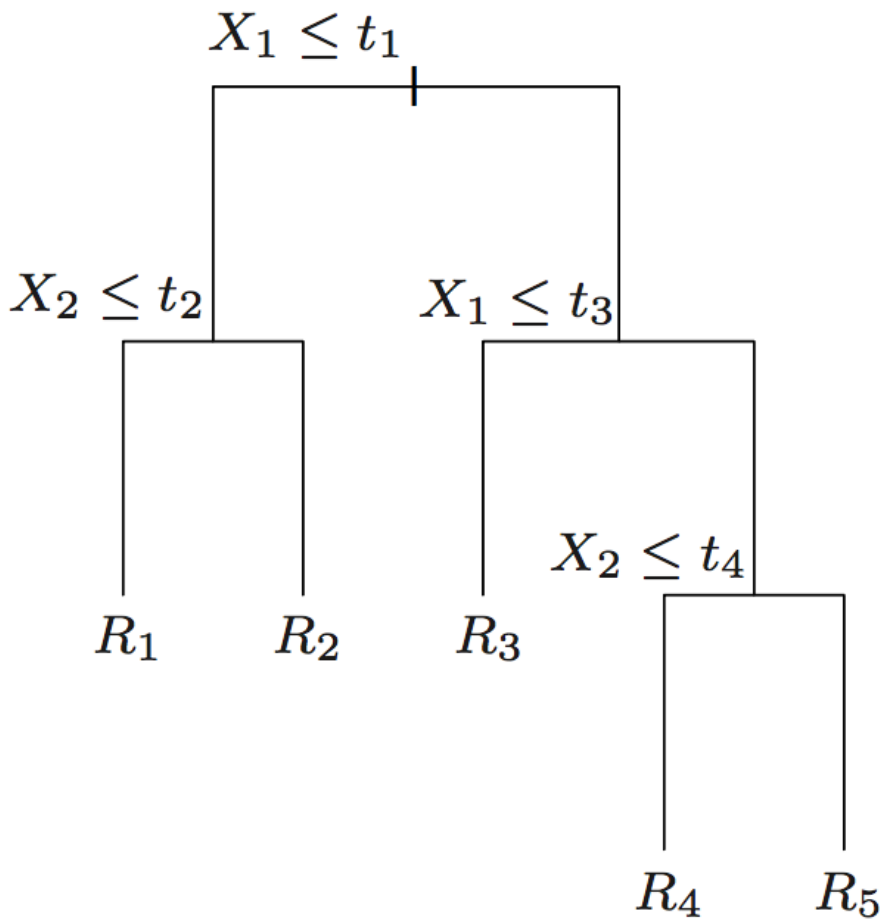
The plot is inspired by the graphic from Stephen Milborrow[1].
The titanic dataset is also part of a Kaggle competition and there is a very nice blog by Trevor Stephens[2] one how to fit this dataset. In particular, it is interesting how important some simple engineering techniques are (exploiting what really is in your data)! This is, however, not applicable here because no names are provided in the ptitantic dataset.

[1] See https://commons.wikimedia.org/wiki/File:CART_tree_titanic_survivors.png
[2] http://trevorstephens.com/kaggle-titanic-tutorial/getting-started-with-r/

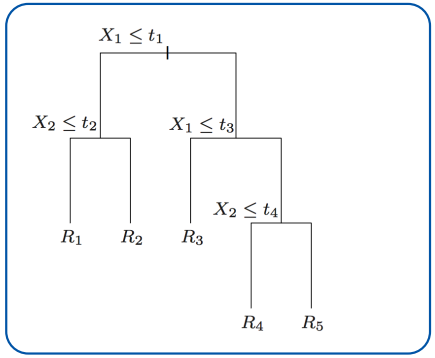


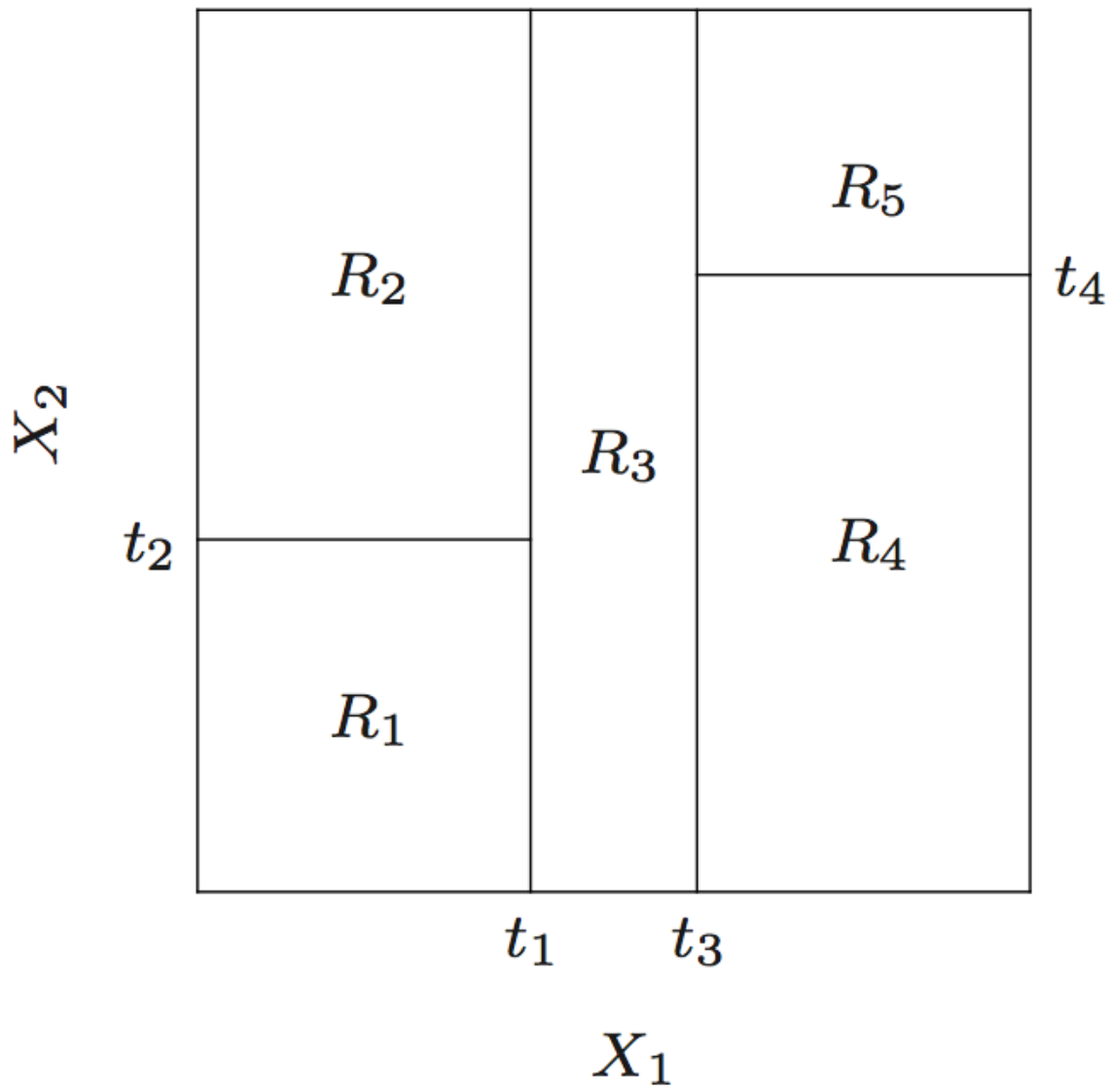
This is an astonishing example of ”women and children first”, as clearly spotted on the data.

Let us get back to theory and study a precise 2d example of an **regression tree**, Figures taken from Hastie et.al. (2009).

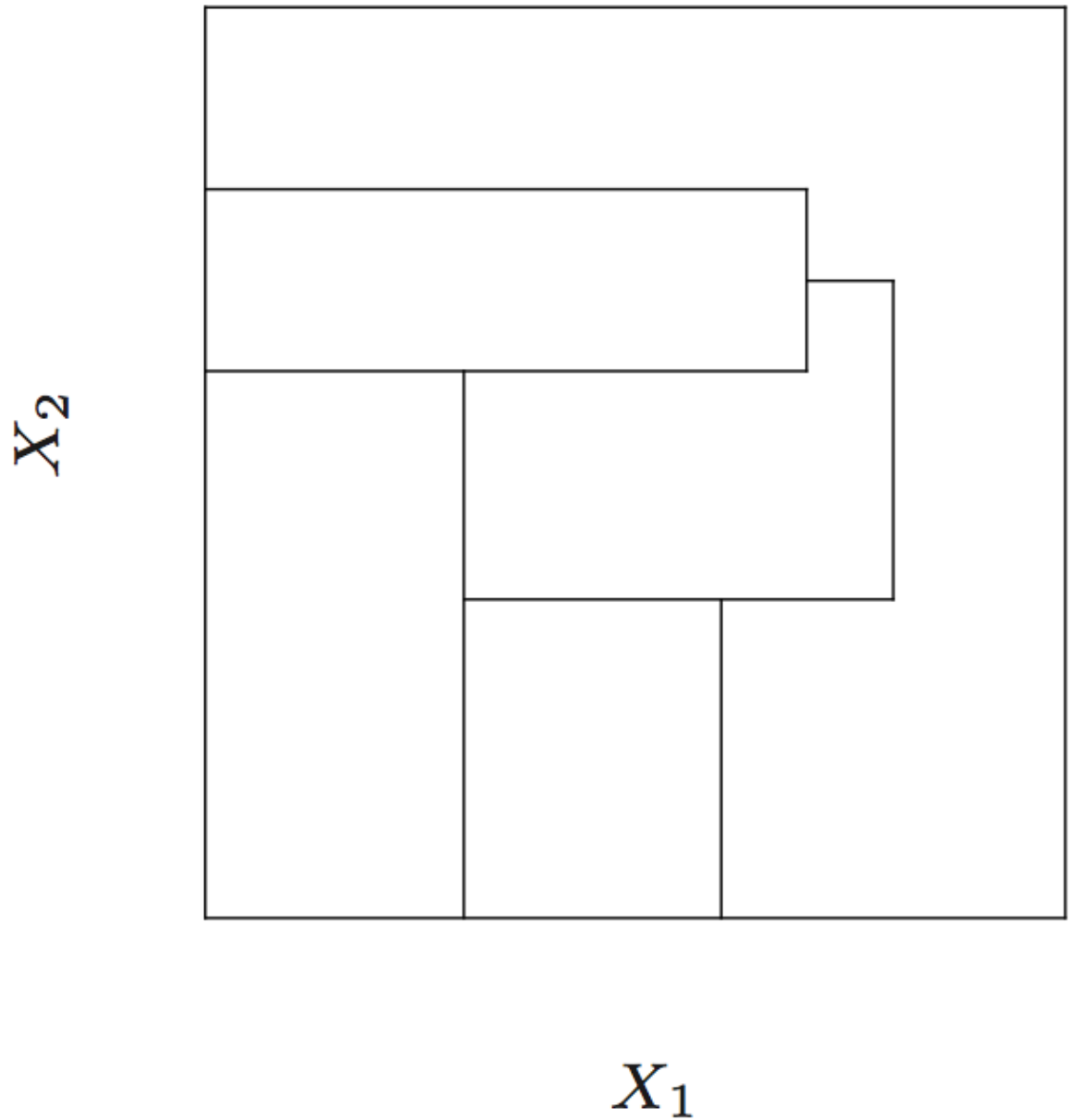


The associated partition is





Not all rectangular partitions can be reached by such a graph.



- Summarizing, we arrive at a model of the form

$$f(x) = \sum_{m=1}^{M} w_m \mathbb{1}_{\{x \in R_m\}}$$

  which fits in the ABM framework with $\phi(x, v_m) = \mathbb{1}_{\{x \in R_m\}}$.
- Note that the binary tree is recursive, which leads to easy and fast implementations. It moreover hast a high degree of interpretability.
- If we have data points $x_1, \ldots, x_n$ it is clear that a possible split will always be done at a data point (and not in between - why?). Thus the data automatically implies a maximum of possible regions $R_1, \ldots, R_M$. The main point is how to choose in a clever way a good performing sub-partition. We denote the possible split points in dimension j by $\mathcal{T}_j$, $j = 1, \ldots, d$.

- E.g. if we have data points $1, 3, -1, 3$ we obtain $\mathscr{T}_1 = \{-1, 1, 3\}$.
- Finding the pest partition is NP-complete, see Hyafil and Rivest (1976)[3]. Hence one needs to find efficient heuristics to construct nearly-optimal trees.
- We will study a **locally optimal approach**, but there are also other attemps, for example the evolutionary algorithm used in the R-package 'evtree'.
- Recall that we have (x_i, y_i), $i = 1, \dots, N$ data points at hand with x_i being d-dimensional, say. Suppose we have a partition into regions $R_1, \dots, R_M$ and we model

$$f(x) = \sum_{m=1}^{M} w_m \mathbb{1}_{\{x \in R_m\}}$$

(as above) and choose to minimize $\sum (y_i - f(x_i))^2$. Then clearly, the optimal $\hat{w}_m$ are given by the local averages

$$\hat{c}_m \propto \sum_{i=1}^{N} \mathbb{1}_{\{x_i \in R_m\}}.$$

[3] L. Hyafil und R. L. Rivest (1976). „Constructing optimal binary decision trees is NP-complete". In: **Information processing letters** 5.1, S. 15–17.

- The next step is the splitting. We split one dimension in two pieces, i.e. consider the partition

$$R_1(j,s) := \{x \in \mathbb{R}^D : x_j \leq s\}, \quad R_2(j,s) := \{x \in \mathbb{R}^D : x_j > s\}$$

of $\mathbb{R}^D$, leading to the minimization problem

$$\min_{j,s} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right].$$

- As already remarked, $\hat{c}_i$, $i = 1,2$ can easily be found by averaging over the entries in R_1 (and R_2, respectively). Moreover, s can also be find very fast by browsing through $\mathscr{T}_j$, rendering the determination of (j^*, s^*) feasible.
- It remains to find a good **stopping criterion**.

- On first sight, one could guess that we run the algorithm until a new split does not increase the fitting quality substantially. However, the problem with trees is more complicated and a split at one step might only increase the fit in a small step but later will lead to just the right classification.
- Hence, on typically grows a **large** tree T_0 first, which is achieved by stopping at a certain minimal node size.
- Then, the large tree is **pruned**, and we describe **cost-complexity pruning**: denote the nodes of the tree T_0 by $R_1, \dots, R_M$ and let

$$N_m = |\{x_i \in R_m\}|$$

$$\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i$$

$$Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2.$$

The cost-complexity criterion is given by

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T|.$$

- The tuning parameter α assesses the tradeoff between tree size and goodness of fit. It is typically estimated via crossvalidation leading to the optimal parameter $\hat{\alpha}$.
- For each α, there is a unique smallest subtree T_α minimizing $C_\alpha(T)$. This tree can be found by succesively collapsing the internal nodes producing the smalles per-node incresase in the cost-complexity creiterion and proceeding until we obtain the single-node tree. In this finite sequence of subtrees T_α is contained (see Hastie et. al. p. 308 for comments and links to the literature).

- **Classification trees** can be treated similarily, but we would typically use a different cost function: consider the case of K classes $1,\dots,K$ and denote by

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} \mathbb{1}_{\{y_i = k\}}$$

  the fraction of observations in node R_m which have class k.
- The class estimator in node R_m is given by the class which has most members in this node, i.e.

$$\hat{k}_m = \arg\max_k \hat{p}_{mk}.$$

- Typical measures are misclassification error, the Gini index, or even entropy.

Be reminded that while being easy to interprete, trees do not come without disadvantages: they are unstable, the solution is typically not globally optimal and they are not smooth. For further discussions we refer to Hastie e.a. (2009)

Hierarchical mixtures of experts

- An alternative to the hard split is to put probabilities on each nodes. It is common to call the terminal nodes experts and we therefore arrive at a mixture of experts. Total probabilities are computed with Bayes' formula, compare Section 9.5 in Hastie e.a. (2009)

In the following lecture we will consider random forests, bagging and boosting of CARTs.