Statistical Learning for Data Science: Advanced techniques using R

Stefan Zohren

5 May 2016

Content of this lecture

- Course overview
- A review of bias-variance-tradeoff and cross-validation
  - Bias-variance-tradeoff
  - Training, cross-validation and test sets
  - The right way of doing cross-validation
  - N-fold cross-validation
- Decision trees (CART)
  - Regression trees
  - Classification trees
- Summary

Courses on statistical learning and data science with R

- Statistical learning for data science: Adv. techniques using R
  - Introduction (recap bias-variance-tradeoff and CV), decision trees;
  - Bagging, boosting and random forests;
  - Support vector machines;
  - Neural networks.
- Statistical learning for data science: Basic techniques using R
  - Introduction, Regression analysis: linear and polynomial, bias-variance-tradeoff;
  - Classification: logistic regression, discriminant analysis;
  - Regularisation (ridge regression and lasso) and cross-validation;
  - Unsupervised learning methods - principle component analysis (PCA) and clustering (k-means and hierarchical).
- Data Wrangling with R

Structure of each lecture

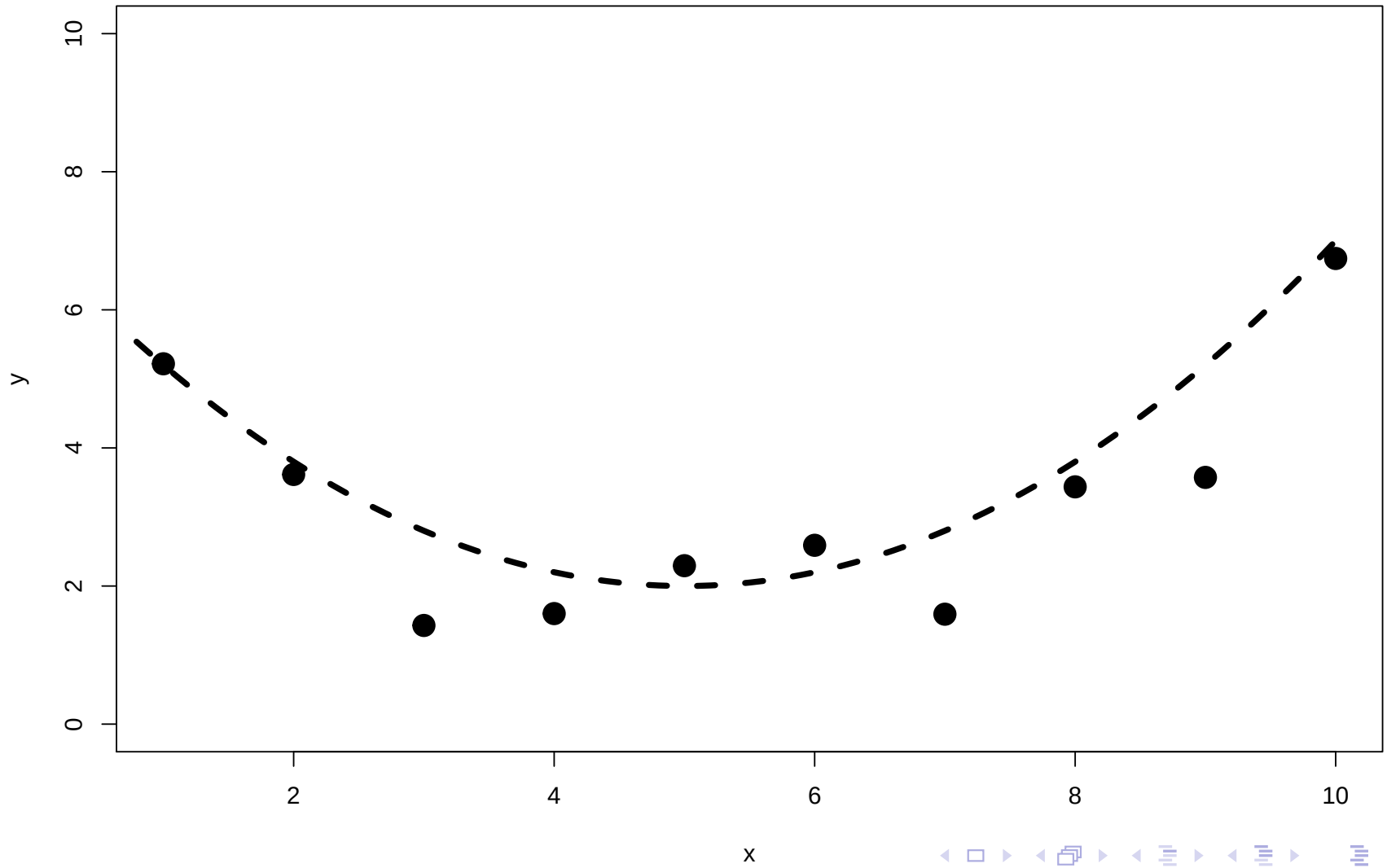
- These slides walk you through the basic examples in the handout provided to you on the course webpage.
- Subsequently, during the practicals, you work through the handout on your computer revising the examples, executing the commands and attempting the exercises.
- The general setup is:
  - 30 min presentation,
  - 30 min practicals,
  - 30 min presentation,
  - 30 min practicals,
  - time for general questions afterwards

Introduction

- The learning algorithms we discussed in "Statistical learning for data science: Basic Techniques using R" were **linear models** and variations thereof.
- The focus on this second module "Statistical learning for data science: Advanced Techniques using R" will be on **non-linear models**.
- A general difficulty in training non-linear models is the fact that the corresponding optimisation problems are often highly non-convex which makes them difficult to solve.

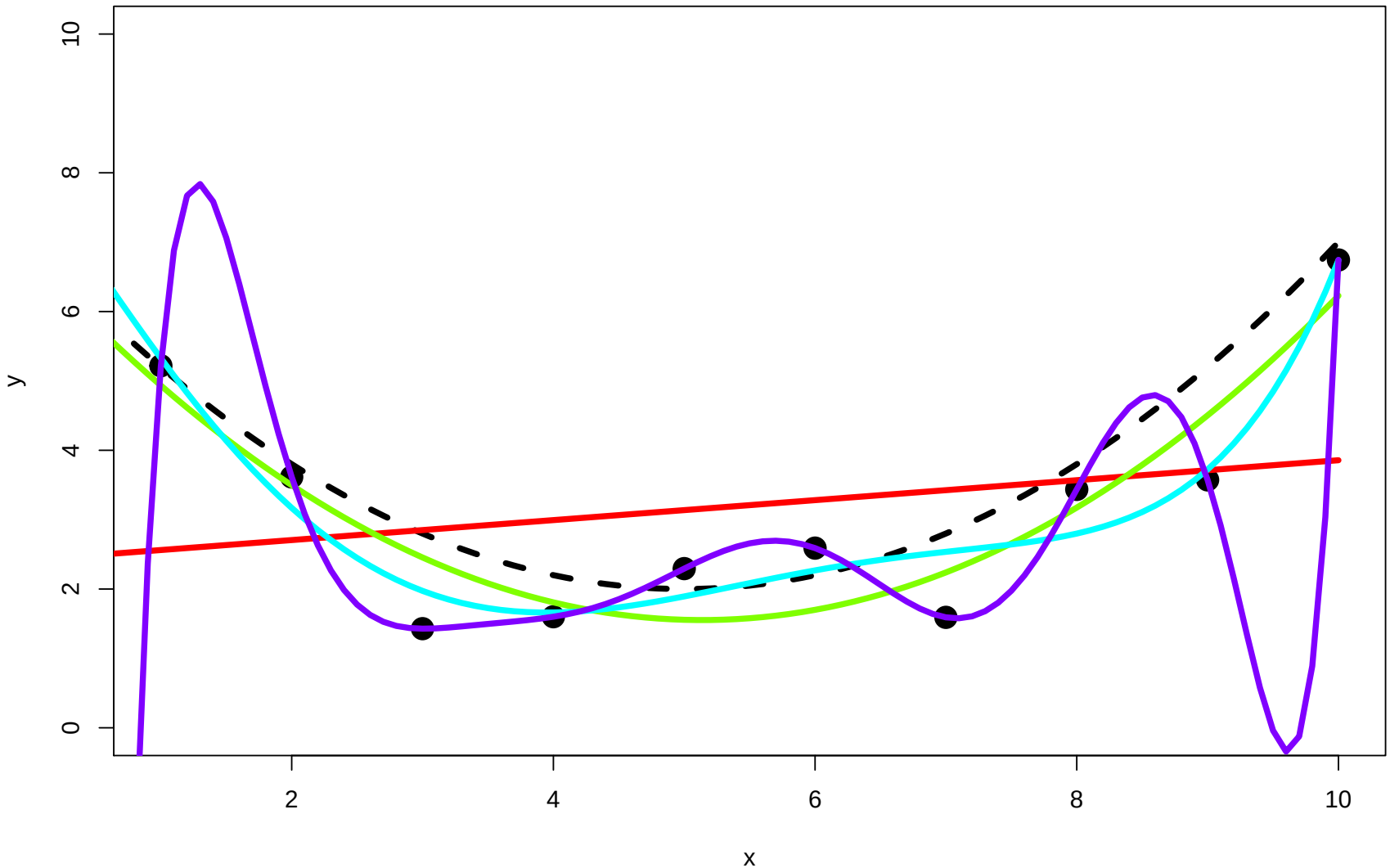
Review of bias-variance-tradeoff

We revise an example from Lecture 1 of the first module of simulated data from the model $y_i = f(x_i) + \epsilon_i$ where the true model is $f(x) = 2 + 0.2(x - 5)^2$ and $\epsilon_i \sim \mathcal{N}(0, 1)$.



Review of bias-variance-tradeoff

We fit polynomials of maximum degree 1, 2, 5 and 10 to the data:



Review of bias-variance-tradeoff

- We see that the RMSE decreases as we increase the flexibility of the model (degree of the polynomial)

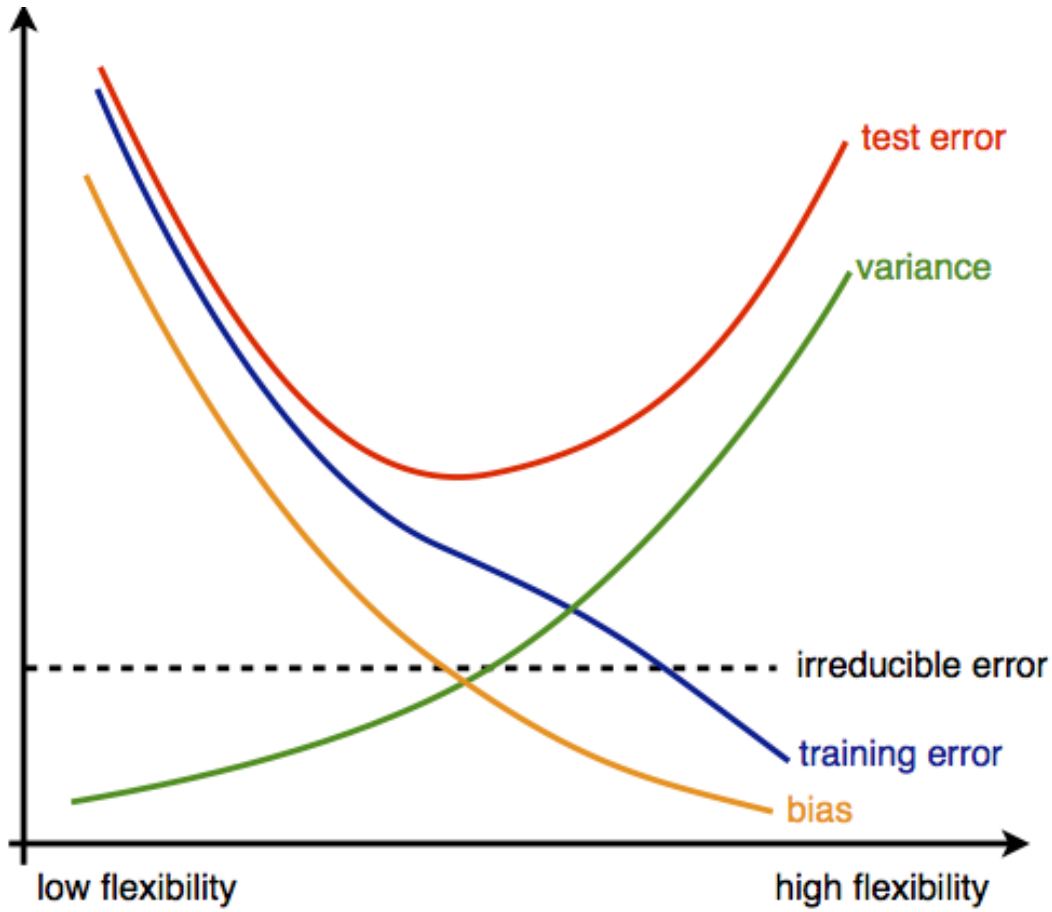
$$\mathrm{RMSE} = \left(\frac{1}{n} \sum_{i=1}^{n} \epsilon_i^2 \right)^{\frac{1}{2}} = \sqrt{\mathrm{mean}((\hat{\mathbf{y}} - \mathbf{y})^2)}$$

- However, for higher flexibility the model does not seem to be a good fit by looking at it
- How can be capture this in terms of errors we can calculate?
- The solution is to hold-out another data set and calculate the RMSE on this data set

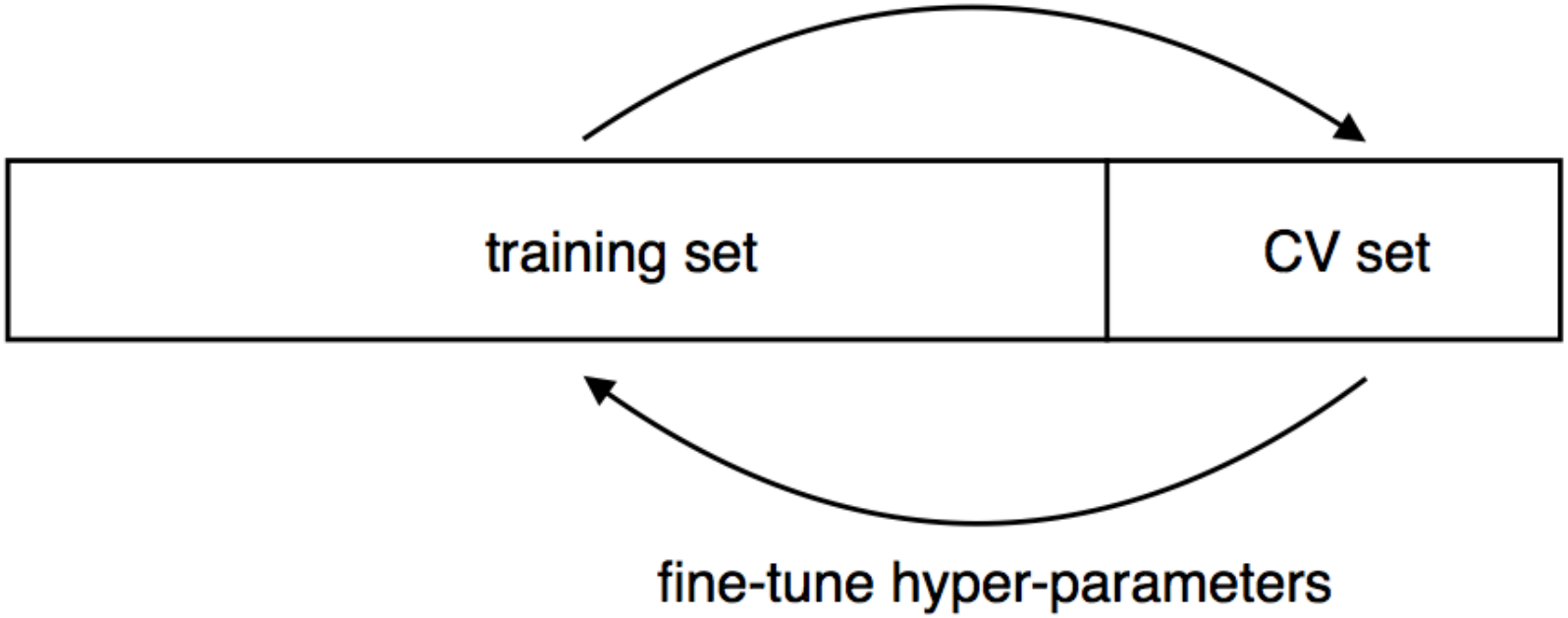
Review of bias-variance-tradeoff

This leads to the bias-variance-tradeoff

$$\mathrm{RMSE}_{test} = \mathrm{Variance} + \mathrm{Bias}^2 + \text{irreducible error}$$



Cross-validation



- We use a **cross-validation set** to **fine-tune hyper-parameters** such as a the shrinkage parameter in regularisation

Cross-validation

- A problem is that each time we look at the hold-out set to tune the parameters, we are effectively training on it. In this case, the error evaluated on the hold-out is under-estimating the true test error
- To circumvent this and to obtain an unbiased estimate of the test error it is thus advisable to hold-out yet another part of the training data to do cross-validation

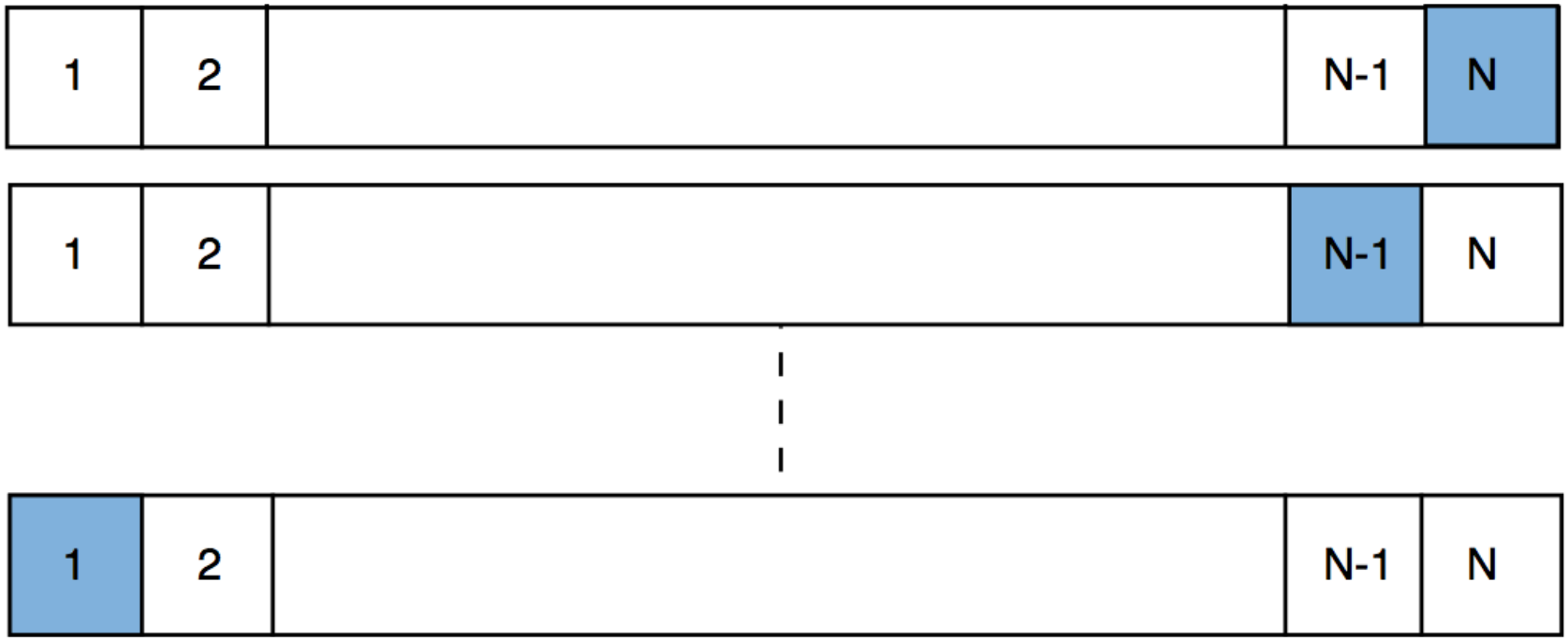
The right way of doing cross-validation

A common mistake in doing cross-validation (CV) is to involve the CV or test set in the training process.

This can be very subtle at times, e.g.

- when doing unsupervised learning, i.e. doing a PCA on the entire data set before splitting the data and regressing on the principle components
- when normalising data, this should be done on the training data only and then the test data should be rescaled with the mean and variance obtained from the training data

N-fold cross-validation



- Instead of doing a single split we can also divide the data in N parts, treating one as the CV set and training on the rest
- This gives rise to N estimates for the CV error from which we can obtain the mean

Regression trees: Basic ideas

Imagine we would like to fit a very simple model to data with response variable **y** and a single predictor **x**, which just consists of an intercept (i.e. a constant):

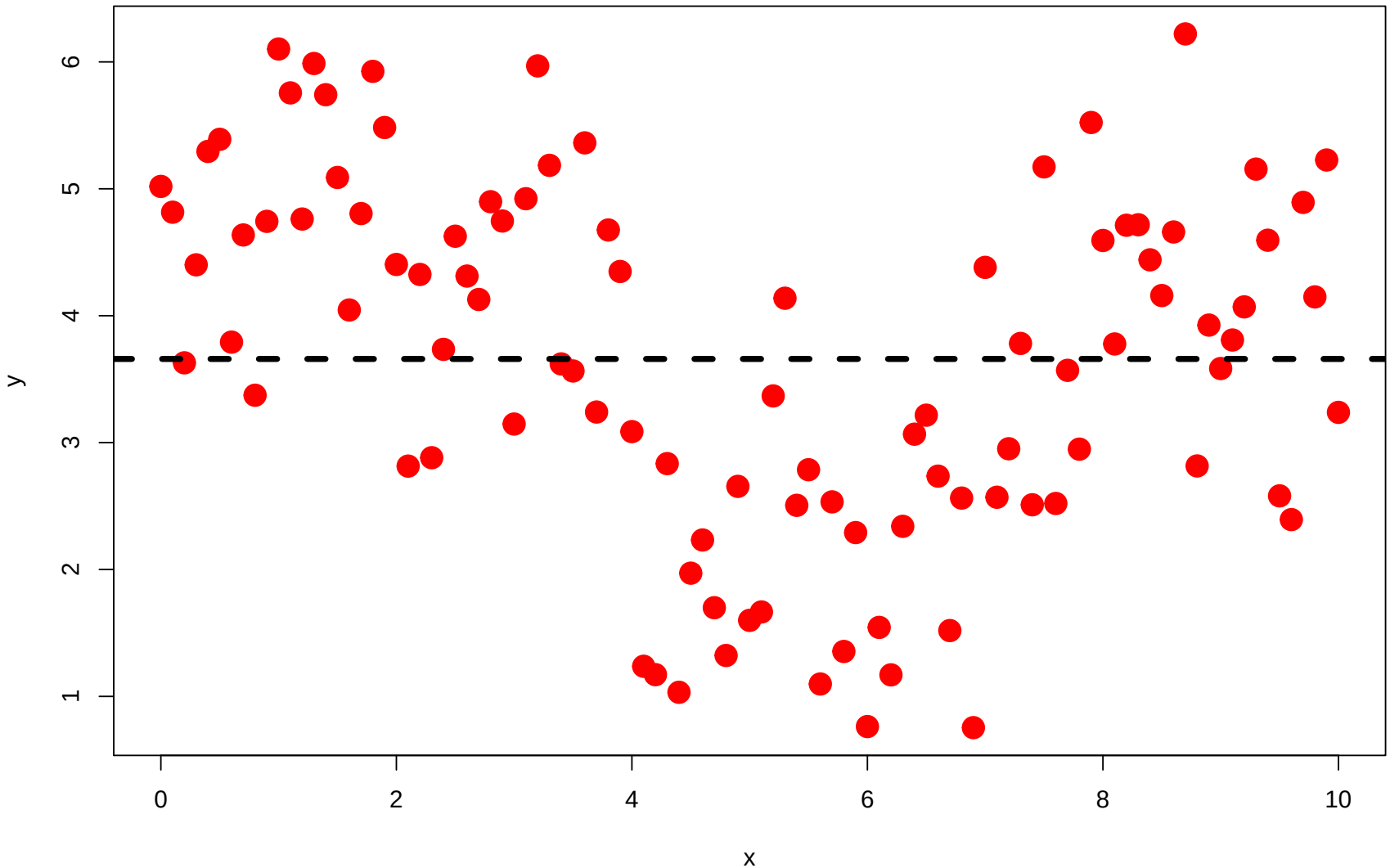
$$y_i = f(x_i) + \epsilon_i, \quad f(x) = a$$

It is intuitively clear that the least squared estimator $\hat{a}$ for a is

$$\hat{a} = \text{mean}(\mathbf{y})$$

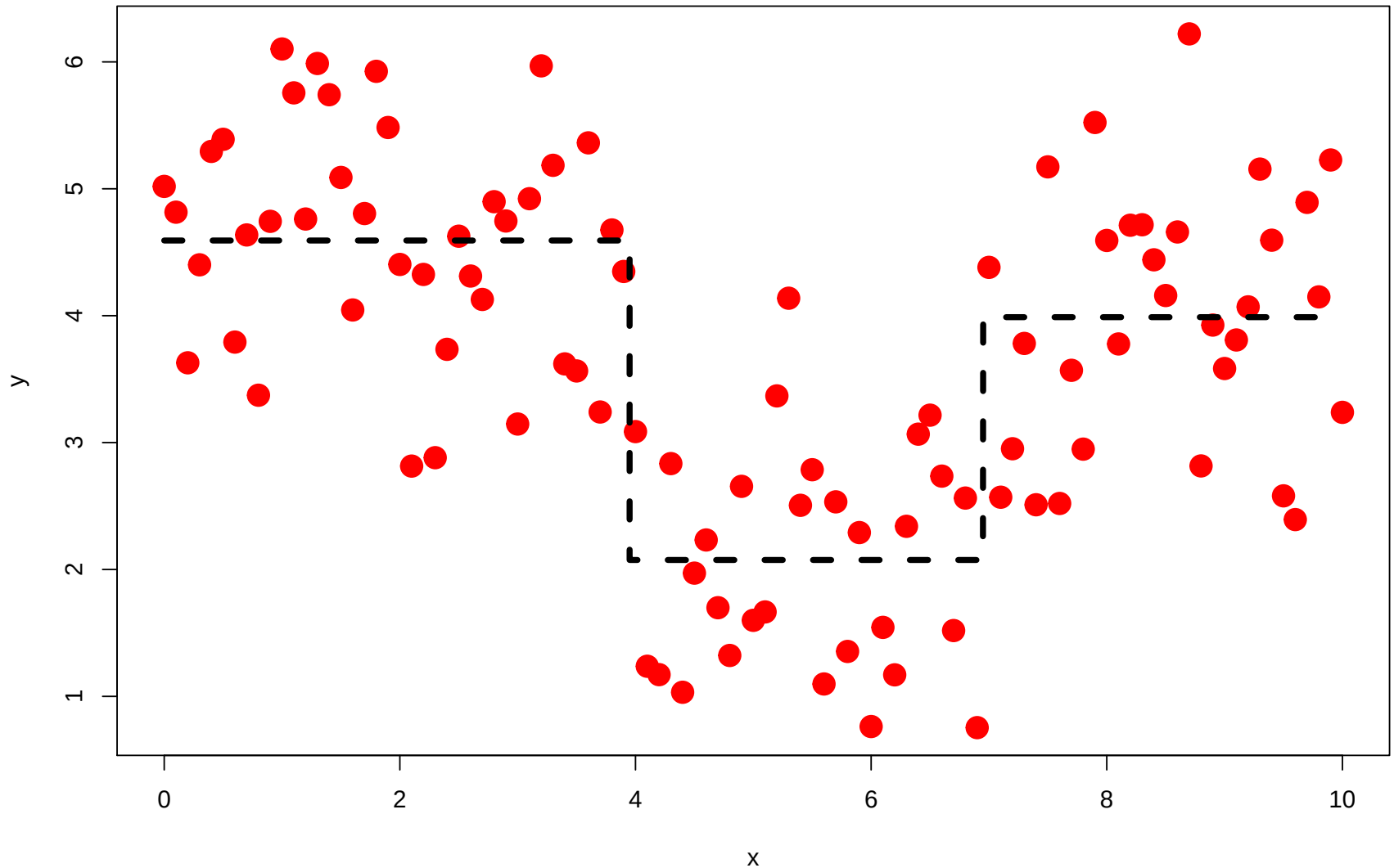
Regression trees: Basic ideas

Here is a toy example using simulated data. The constant estimator is the mean shown by the dashed line.



Regression trees: Basic ideas

We can make it piecewise constant to obtain a non-linear function which is a better fit:



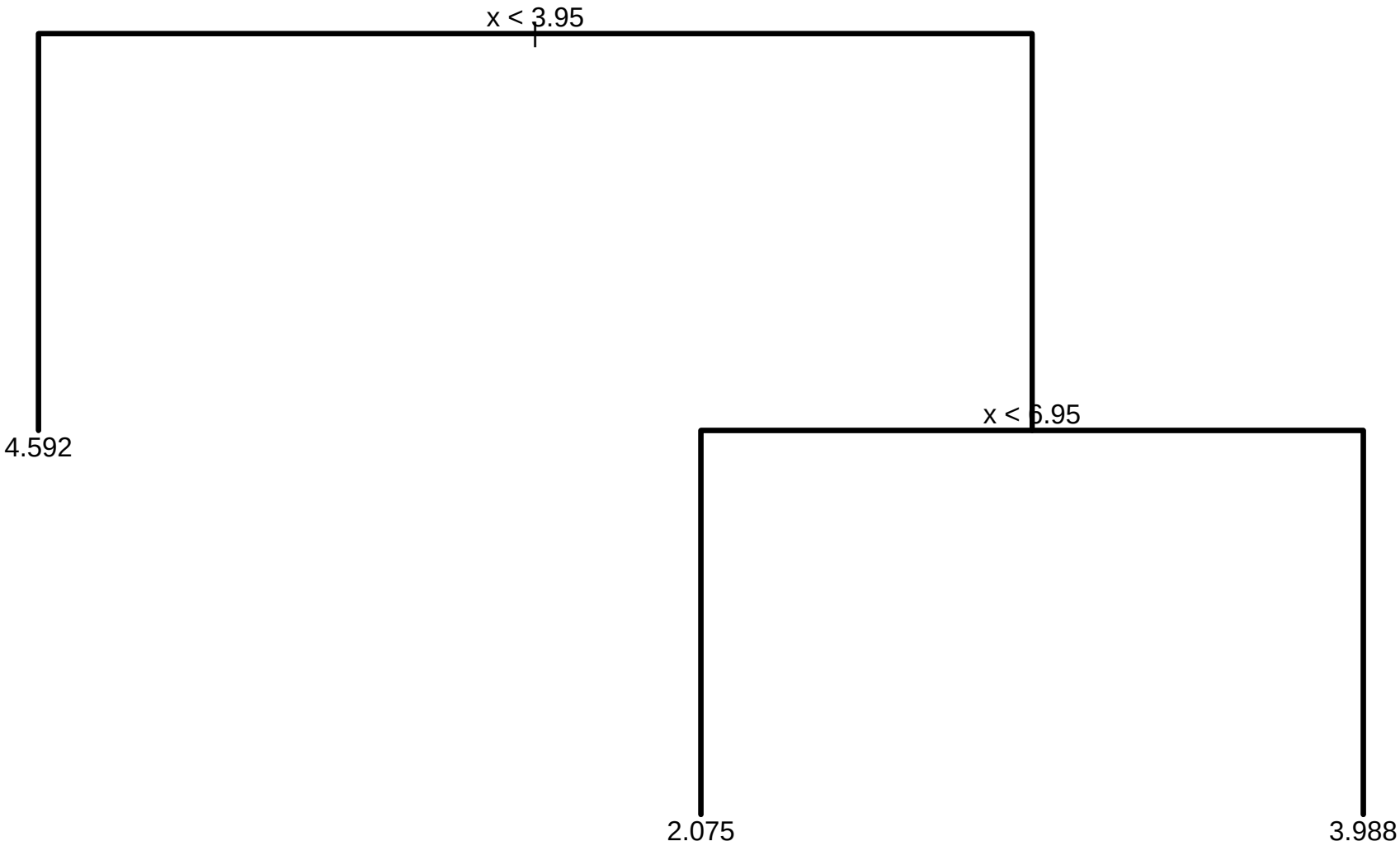
Regression trees: Basic ideas

The basic idea behind regression trees is to fit a piecewise constant function to the response variable by:

- looking for the best binary split amongst all predictors (according to some reduction in loss)
- given the previous split search for the next best split in a greedy manner etc.

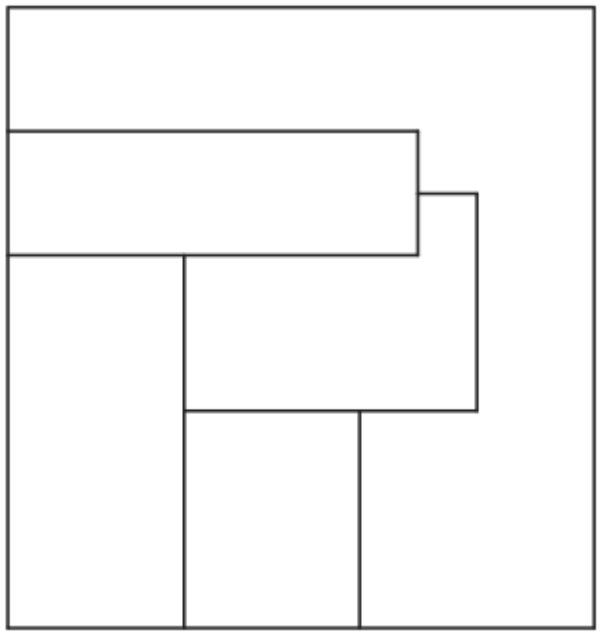
Regression trees: Tree representation

The process can be represented by a **binary decision tree**

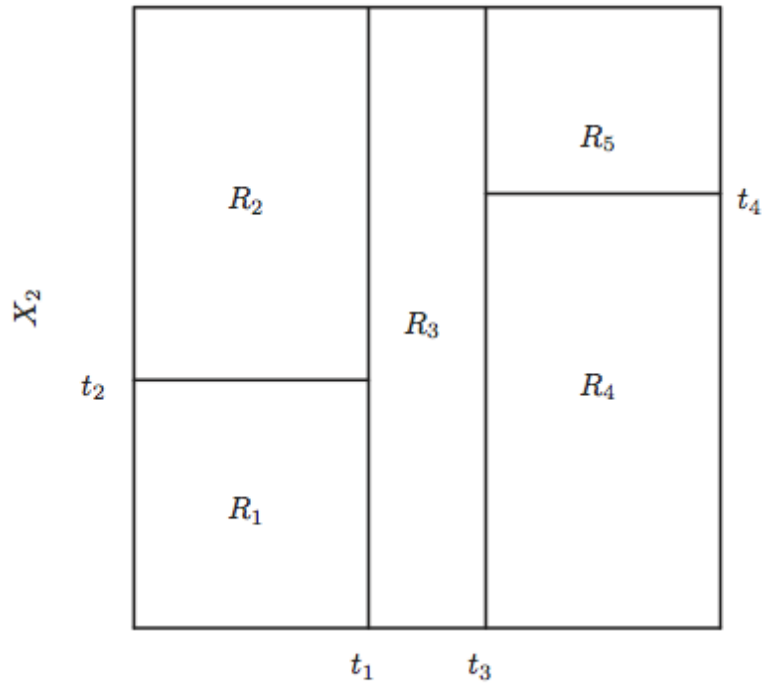


Regression trees: Tree representation

For two predictors the partitioning through splits looks as follows:



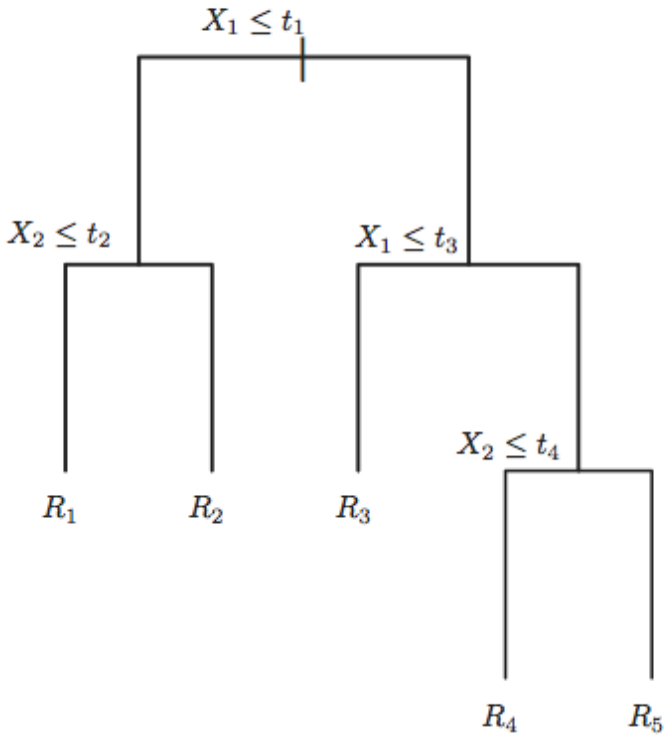
X_1

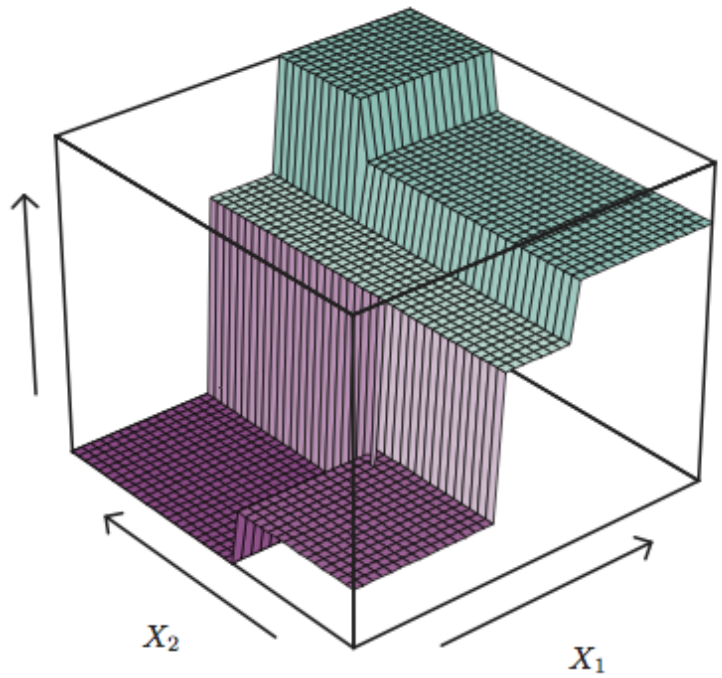


The partition on the left is incompatible with the greedy splitting strategy.

Regression trees: Tree representation

The corresponding tree and piecewise constant function look as follows:





Classification trees

- A nice aspect of decision trees is that we can easily also deal with categorial variables both in the predictors as well as in the response variable. If the response variable is a categorial variable we call the model a **classification tree**.
- As opposed to predicting a constant continuous value for the response variable in a certain partition of the predictor space, we now predict a constant value of the label.
- The loss function is then changed from RSS to a measure suitable for a categorial variable, such as so-called cross-entropy or Gini-index.

Regression trees: Implementation in R

- In `R` classification and regression and trees (CART) are implemented (for example) in the `tree` package
- Use e.g. `tree.fit <- tree(y ~ x, data = my_data)` to fit a classification or regression tree
- Trees are 'regularised' by pruning their size (the bigger the tree the more flexible the model). This is done in `R` using `prune.tree()` and `prune.misclass()`
- Cross-validation can be done automatically using `cv.tree()`

Summary

- We revised the bias-variance-tradeoff, one of the key paradigms of statistical learning
- We motivated the need for a separate training set, cross-validation set and test set
- Cross-validation and in particular N-fold cross-validation were introduced
- Regression and classification trees were introduced as simple non-linear models which are easy to train and easy to interpret
- In the next class we will see how to combine these tree models with powerful ensemble techniques leading to random forests, bagging and boosting
- To learn most, it is instructive to walk through the examples on your machine as well as to attempt all exercises!