11

Regression Trees

Global models

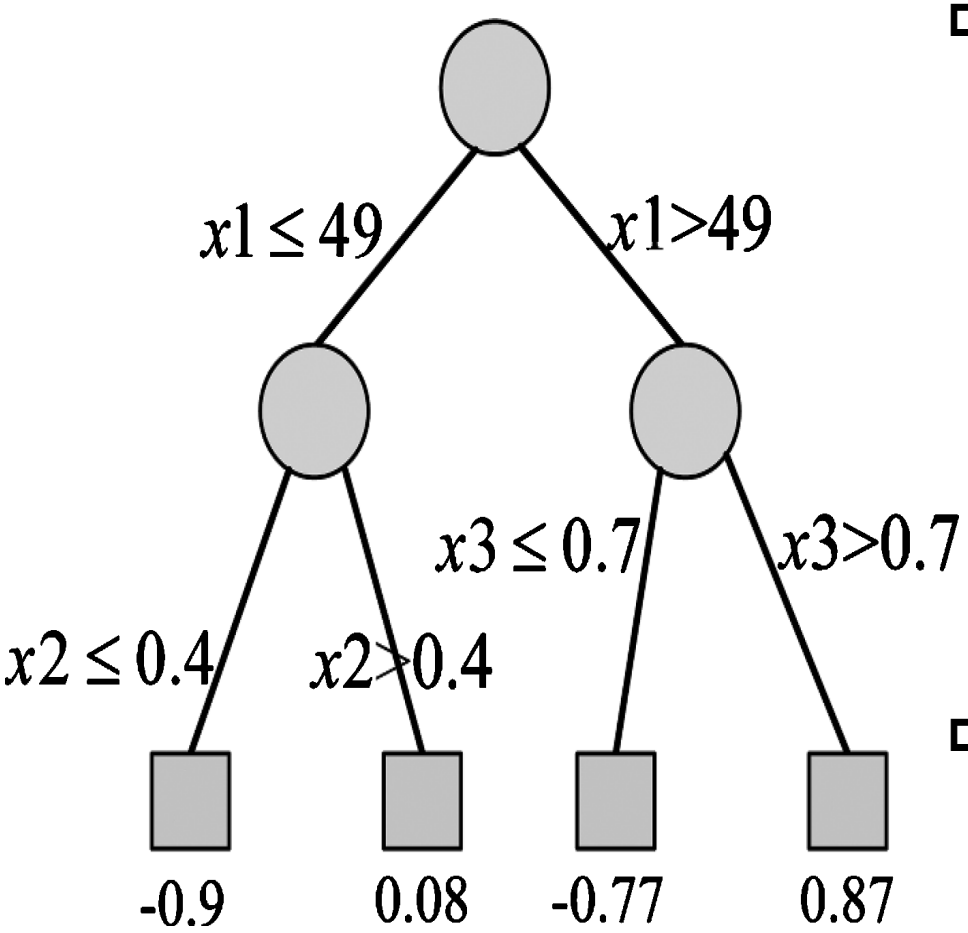
- Linear regression (a.o.) models are "global", i.e., one equation is used for the entire data domain
  - If the data has many features with complex interactions and complex effects of the output, building a good global model can be very difficult, even impossible
  - *A very simple example:* A shop owner wants to predict customer amount at any time in day (e.g., to dynamically decide on the number of salespersons at shop during the day)
    - But the data (and therefore – regression models) will be different on working days, weekends, holidays etc. Maybe each type of day should have its own model.
    - In more complex problems this can't be known/done manually.

Recursive partitioning

- Alternative to the global models: recursively divide the data in ever smaller subsets until the relationships in each subset are so simple that they can be accurately described using very simple models (even constants)

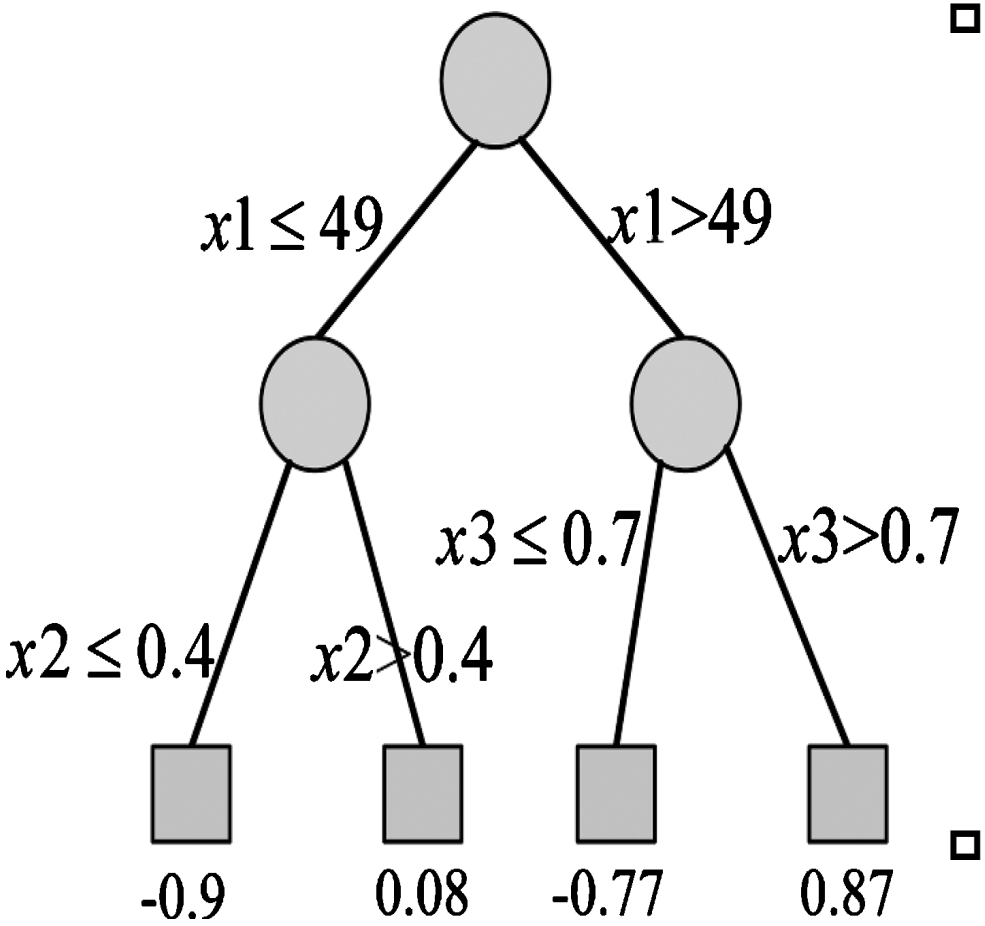
  This is called Recursive Partitioning

Regression Tree



- Regression tree is a representation of recursive partitioning of the data
  - Splits (decision nodes) = partitioning rules that use one of the input variables
  - Leaves (response nodes) = simple models. Usually just the mean of y of the corresponding data subset
- Data partitioning usually is binary (we work with binary trees), i.e., for each partitioning the data is always divided in two subsets

Usage of regression tree



- To use such tree for prediction of y:
  - Start with the root node
  - Answer its question about the value of the feature
  - Depending on the answer follow to the next node down
    - (A tree might be drawn also so the condition for an inner node is shown only once. In that case answer True usually means going to the left subtree and answer False usually means going the the right subtree.)
- Repeat this process until a leaf node is reached. That node contains your prediction of $\hat{y}$.

Advantages of trees

- Tree growing algorithms are relatively fast
- Prediction process is fast – there are no complex computations – just follow a path to a leaf node and retrieve the constant
- Relatively simple interpretation of the model – it's relatively easy to see which input variables are important for prediction and how exactly they influence the result
- Although trees don't give “smooth” predictions, you can still approximate smooth surfaces near enough by either using deeper trees or using ensembles of trees
- Tree growing already incorporates feature subset selection because irrelevant features won't be used in the split nodes or will be used only rarely (but there is no guarantee)

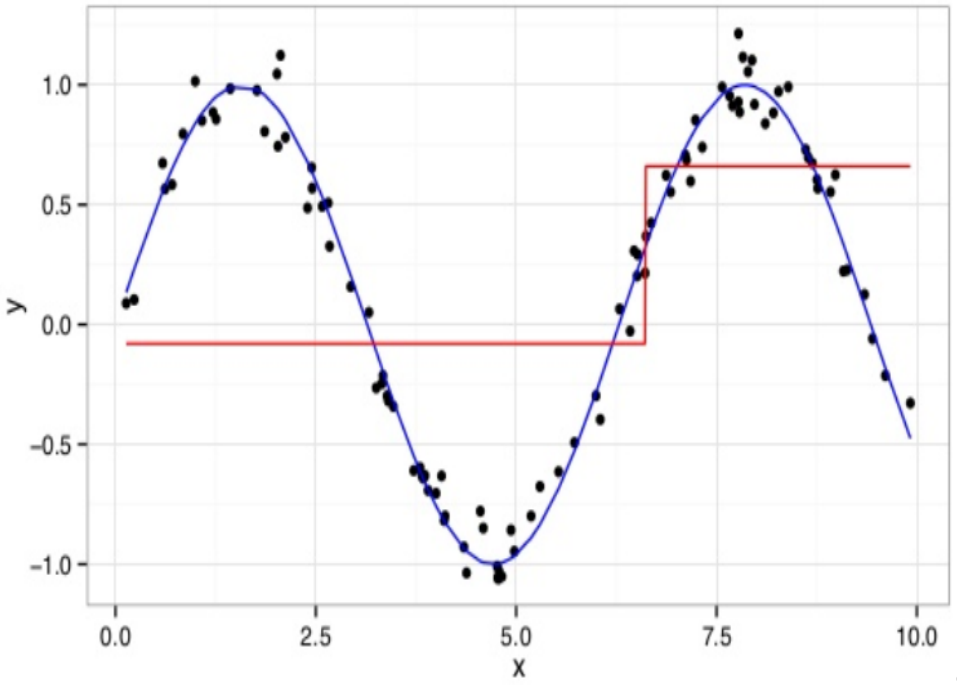
Regression tree growing algorithm

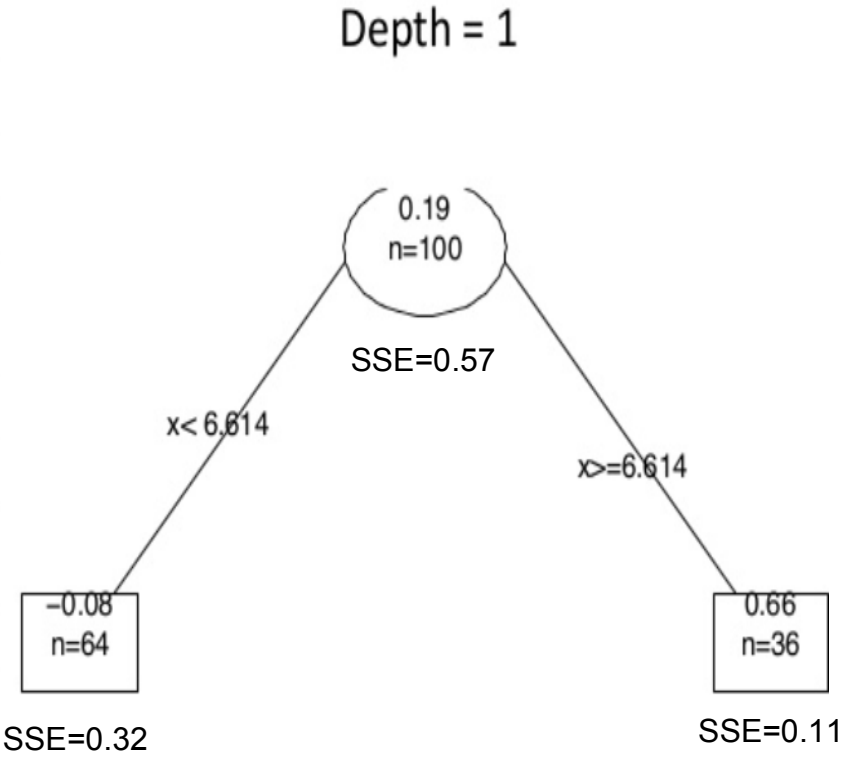
1. Start with one node (root) containing the entire training data
2. Compute mean of y for the training examples of the node (constant model). Compute it's error if it would be the prediction of y for those training examples
   - Sum of Squared Error, SSE
3. Try to split the data in the middle between each two adjacent values for each feature x. Compute how much would SSE reduce because of such partitioning (let's call it *SSE Reduction, SSER*) if each of the two new data subsets would have its own constant model. Split the data at the position with the biggest reduction (maximize SSER).

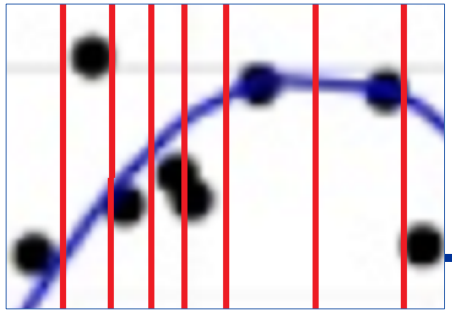
$$SSER = SSE_{root} - (SSE_{leaf1} + SSE_{leaf2})$$

4. With each new leaf recursively go to step 1

Example with one feature





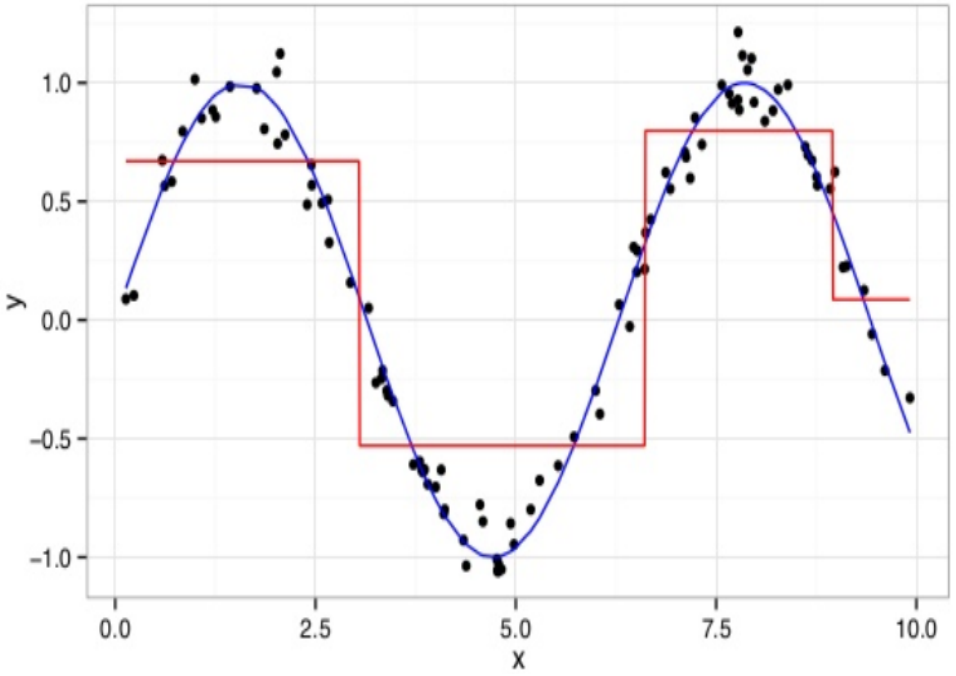


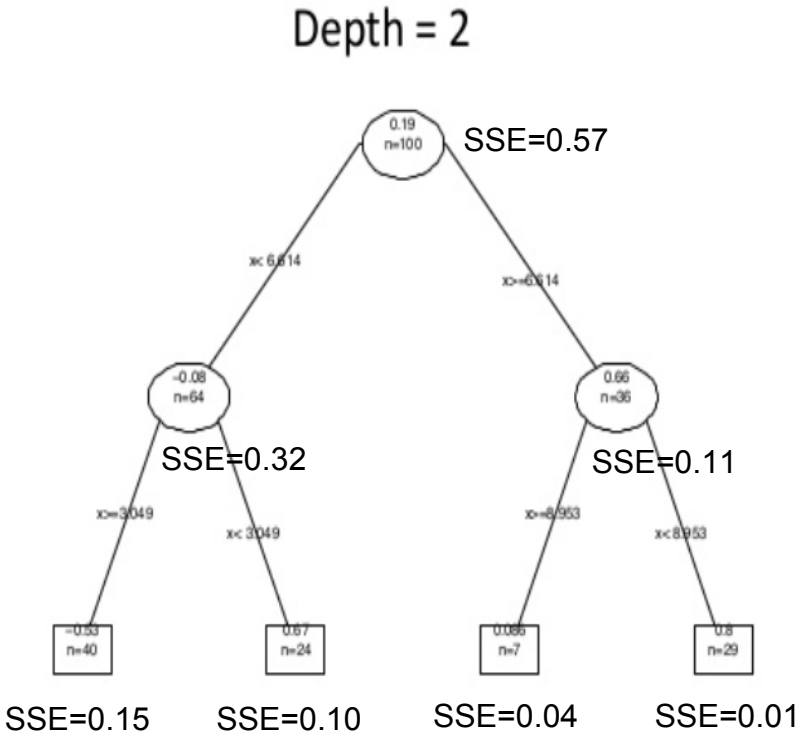
SSER = 0.57 - (0.32 + 0.11) = 0.57 – 0.43 = **0.14**

We have improvement because 0.14 > 0.

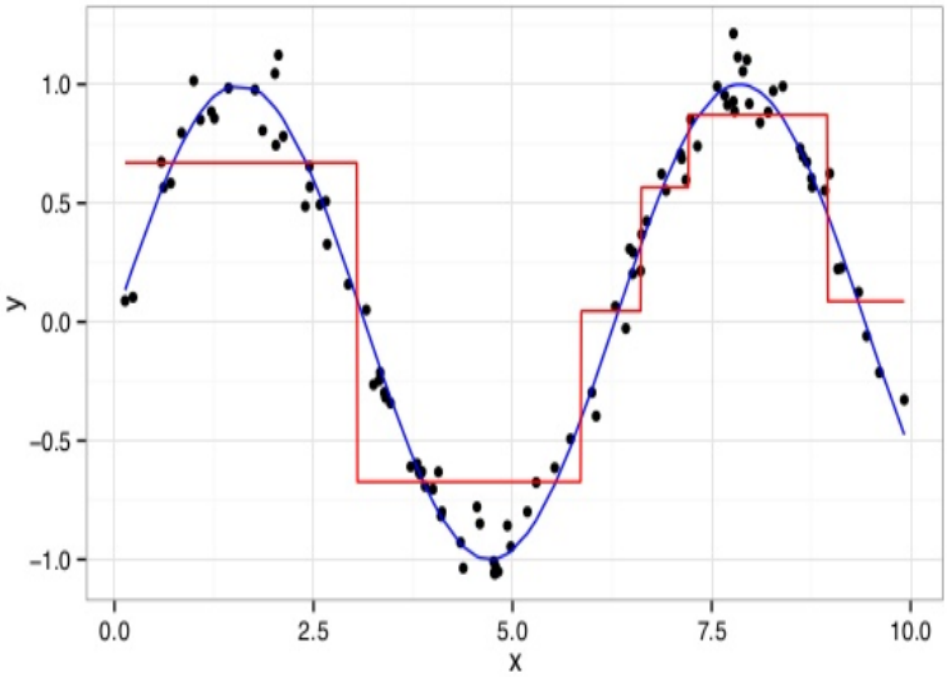
Let's say we tried all possible cutting positions and found out that 0.14 is the largest of all SSER so we use this position.

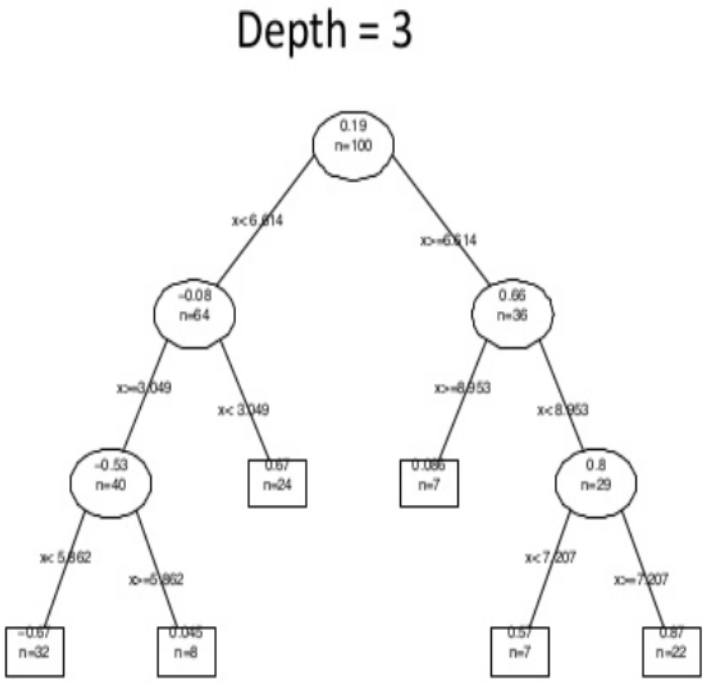
Example with one feature



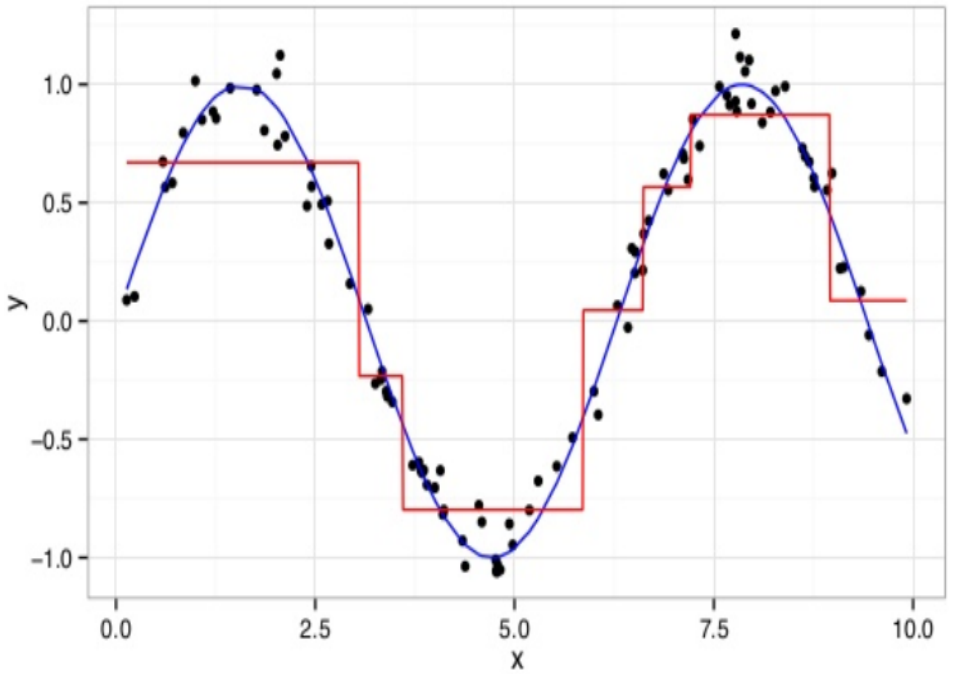


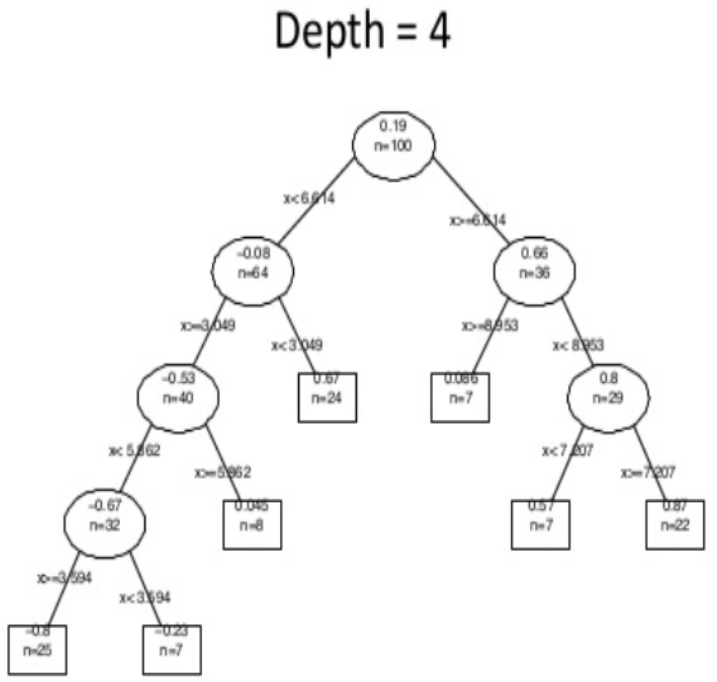
Example with one feature





Example with one feature





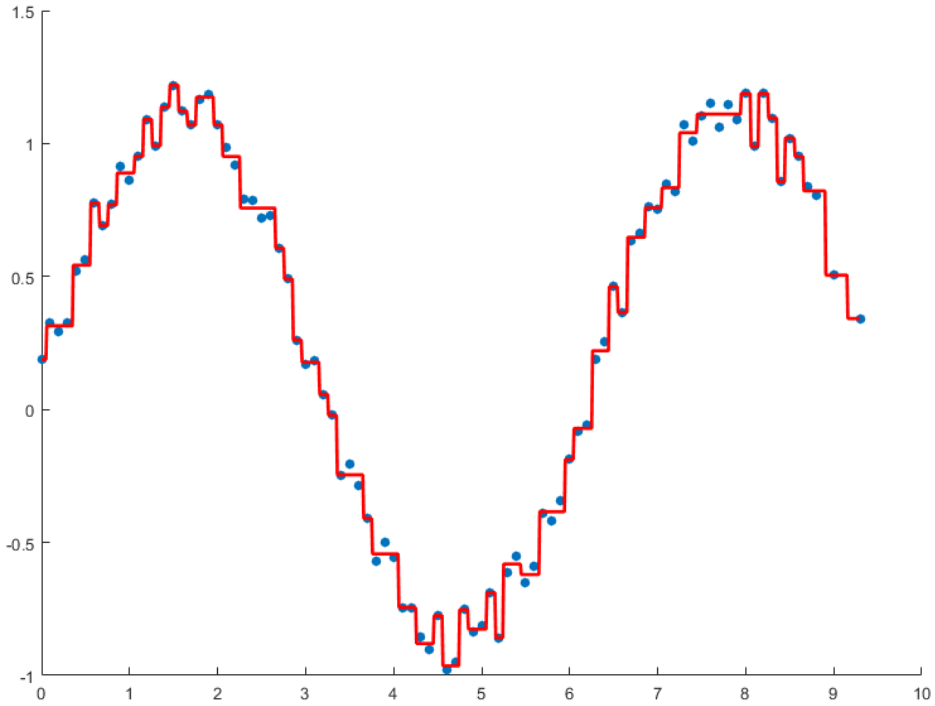
Termination conditions for the recursive partitioning (the last three introduce parameters):

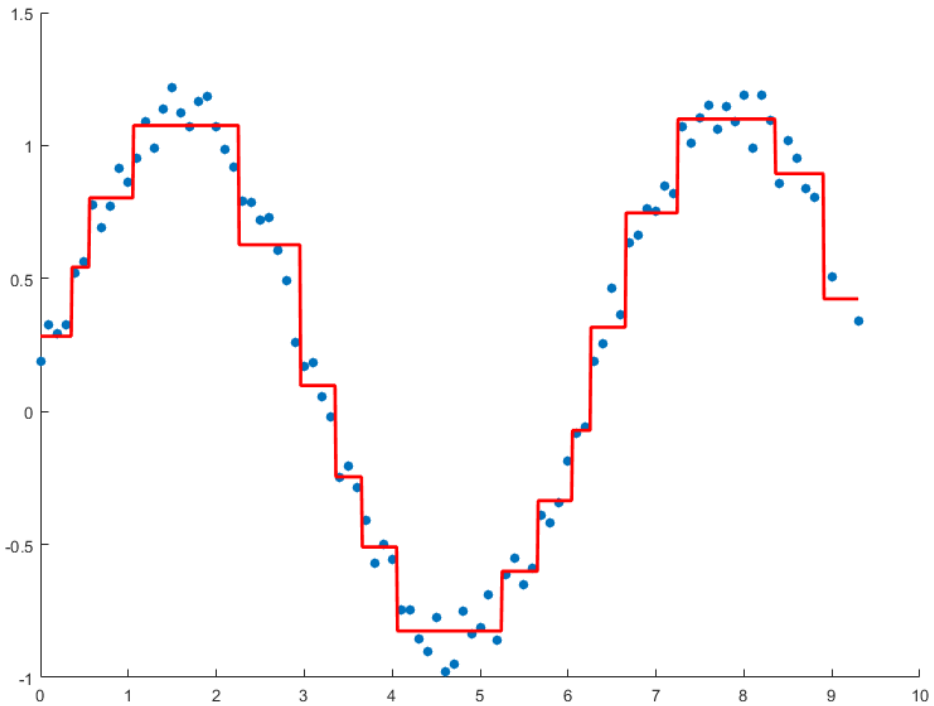
- If SSER is lower than or equal to 0
- If SSE is reached that is lower than, for example, 5% of the root node's SSE
- If a leaf would have a data subset of too small size
  (for example, 10 data points or 5% of data points from the whole data set)
- If a predefined maximum tree depth is reached (for example, 10 levels)

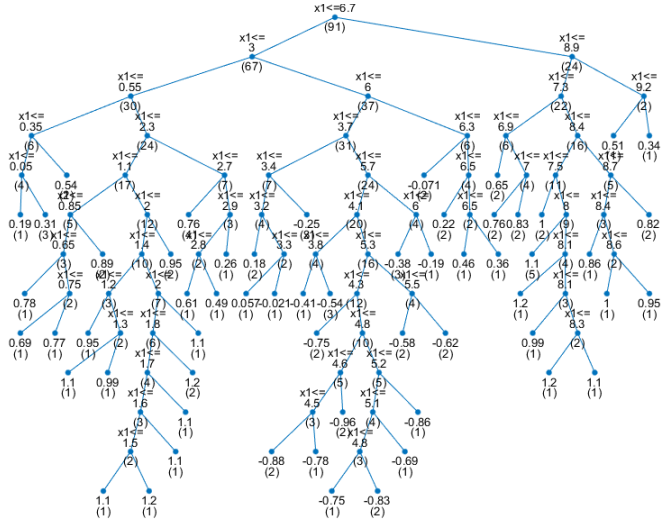
Tree pruning

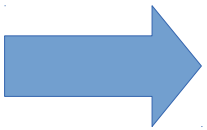
- The built tree is often overfitted, i.e., it is too complex for the data. It must be simplified, increasing it's predictive performance.
- Tree pruning
  - For pruning we no longer use simple training error
  - As a pruning criterion that estimates prediction error we can use either some complexity penalization criterion or error estimation in a separate validation data set (Hold-Out)
  - The process: For each pair of leaf nodes and their predecessor node compute the criterion and see whether the criterion would improve if the subtree would be pruned away and their predecessor would be made into a leaf. If it would improve, do the pruning.

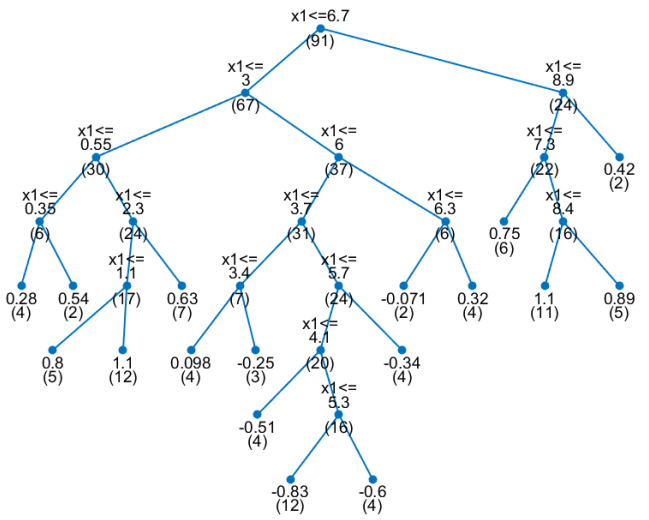
Tree pruning







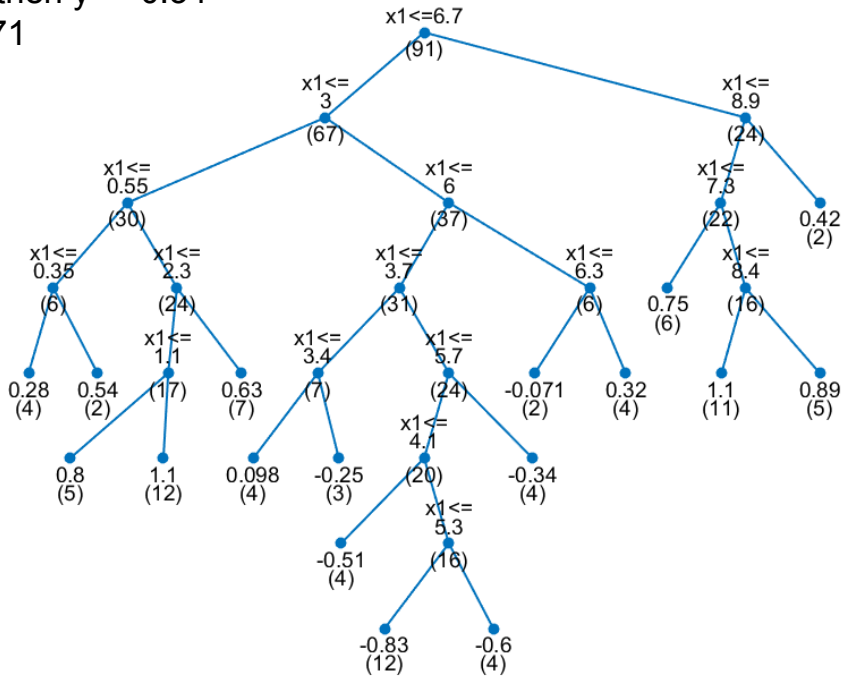




Rule sets

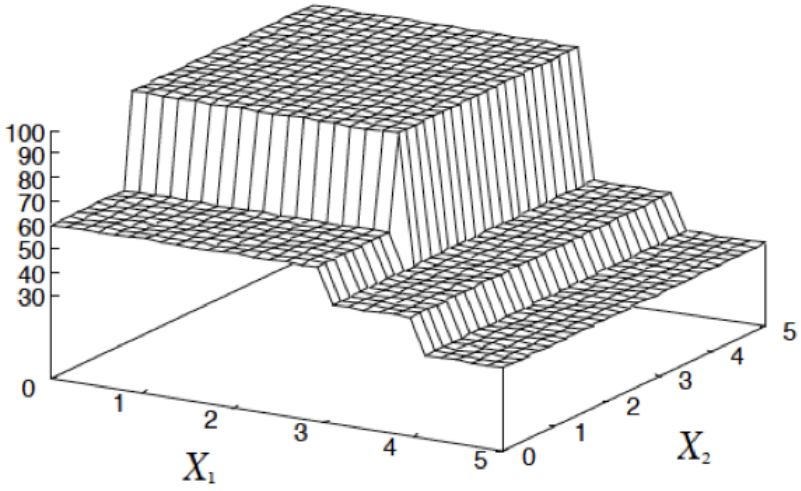
- A regression tree can be transformed into a rule set

if x1 <= 6.7 and x1 <= 3.0 and x1 <= 0.55 and x1 <= 0.35 then y = 0.28
if x1 <= 6.7 and x1 <= 3.0 and x1 <= 0.55 and x1 > 0.35 then y = 0.54
if x1 <= 6.7 and x1 <= 3.0 and x1 > 0.55 and x1 <= 2.3 and x1 <= 1.1 then y = 0.8
if x1 <= 6.7 and x1 <= 3.0 and x1 > 0.55 and x1 <= 2.3 and x1 > 1.1 then y = 1.1
if x1 <= 6.7 and x1 <= 3.0 and x1 > 0.55 and x1 > 2.3 then y = 0.63
if x1 <= 6.7 and x1 > 3.0 and x1 <= 6.0 and x1 <= 3.7 and x1 <= 3.4 then y = 0.098
if x1 <= 6.7 and x1 > 3.0 and x1 <= 6.0 and x1 <= 3.7 and x1 > 3.4 then y = -0.25
if x1 <= 6.7 and x1 > 3.0 and x1 <= 6.0 and x1 > 3.7 and x1 <= 5.7 and x1 <= 4.1 then y = -0.51
if x1 <= 6.7 and x1 > 3.0 and x1 <= 6.0 and x1 > 3.7 and x1 <= 5.7 and x1 > 4.1 and x1 <= 5.3 then y = -0.83
if x1 <= 6.7 and x1 > 3.0 and x1 <= 6.0 and x1 > 3.7 and x1 <= 5.7 and x1 > 4.1 and x1 > 5.3 then y = -0.6
if x1 <= 6.7 and x1 > 3.0 and x1 <= 6.0 and x1 > 3.7 and x1 > 5.7 then y = -0.34
if x1 <= 6.7 and x1 > 3.0 and x1 > 6.0 and x1 <= 6.3 then y = -0.071
if x1 <= 6.7 and x1 > 3.0 and x1 > 6.0 and x1 > 6.3 then y = 0.32
if x1 > 6.7 and x1 <= 8.9 and x1 <= 7.3 then y = 0.75
if x1 > 6.7 and x1 <= 8.9 and x1 > 7.3 and x1 <= 8.4 then y = 1.1
if x1 > 6.7 and x1 <= 8.9 and x1 > 7.3 and x1 > 8.4 then y = 0.89
if x1 > 6.7 and x1 > 8.9 then y = 0.42

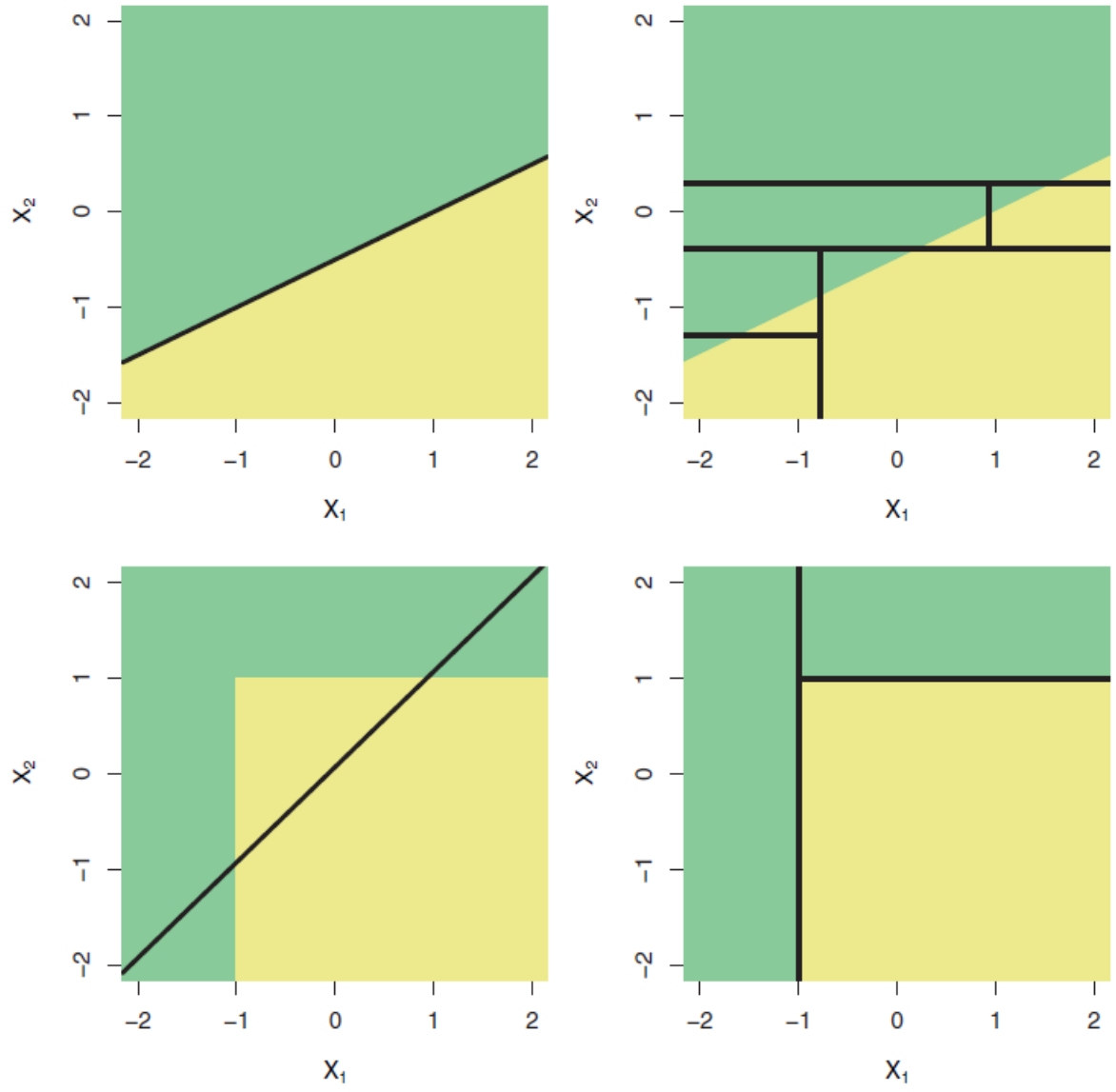


Notes

- There is no guarantee in the optimality of the tree. It is built using the so-called "greedy" principle. It is gradually made more complex by choosing best splits.
- Disadvantages of trees:
  - The tree building process is not stable. Even little changes in data can result in vastly different trees (but it also can be used as an advantage)
  - In multidimensional data, trees partition the data in rectangular regions parallel to axis because they always split data using just one x



Notes



Algorithms

- ***For classification*** *(decision trees)*
  - ID3
  - C4.5 / C5.0 / J48
  - CART
  - RPART
  - CHAID
  - CTREE
  - GUIDE
  - QUEST
  - ...
- ***For regression*** *(regression trees)*
  - M5 / M5'
  - CART
  - RPART
  - CTREE
  - GUIDE
  - ...
- ***Related method***
  - Multivariate Adaptive Regression Splines (MARS)
- ***Ensembles***
  - Bagging
  - Boosting
  - Random Forest
  - Extremely Randomized Trees
  - ...