Nearest Neighbour Algorithms, Trees and Forests

Maschinelles Lernen 1 - Grundverfahren WS20/21

Prof. Gerhard Neumann

KIT, Institut für Anthropomatik und Robotik

Learning Outcomes

What will we learn today?

Nearest Neighbors:

- What an non-parametric/instance based learning algorithm is
- … start with the most simple non-parametric algorithm: k-Nearest Neighbor
- What is the curse of dimensionality
- How to compute the nearest neighbors efficiently

Trees:

- How can we use trees for classification and regression
- Why should we use ensembles of trees (forests)?
- Why should these be random to some extend?

Today‘s Agenda!

Nearest Neighbour Algorithms:

- k-Nearest Neigbhour Classifiers
- Curse of dimensionality
- Indexing with KD-trees

Tree-based methods

- **For regression:** Regression Tree
- **For classification:** Decision Tree
- Almost the same algorithms!

Random forest

- Bagging predictors
- Randomization

Slides adapted from David Sontag, New York University

K-Nearest Neighbor Algorithms

Non-parametric Methods

Non-parametric methods store all the training data and use the training data for doing predictions. They do not adapt parameters or a parametric model. They are also often referred to as instance-based methods.

- ✓ Complexity adapts to training data
- ✓ Very fast at training
- × Slow for prediction
- × Hard to use for high-dimensional input

Algorithms:

- k-Nearest Neighbor Algorithm (today)
- Locally Weighted Regression (not covered)
- Kernel Methods and Gaussian Processes (later)

K-Nearest Neighbour Classifier

To classify a new input vector x, examine the k-closest training data points to x and assign the object to the most frequently occurring class

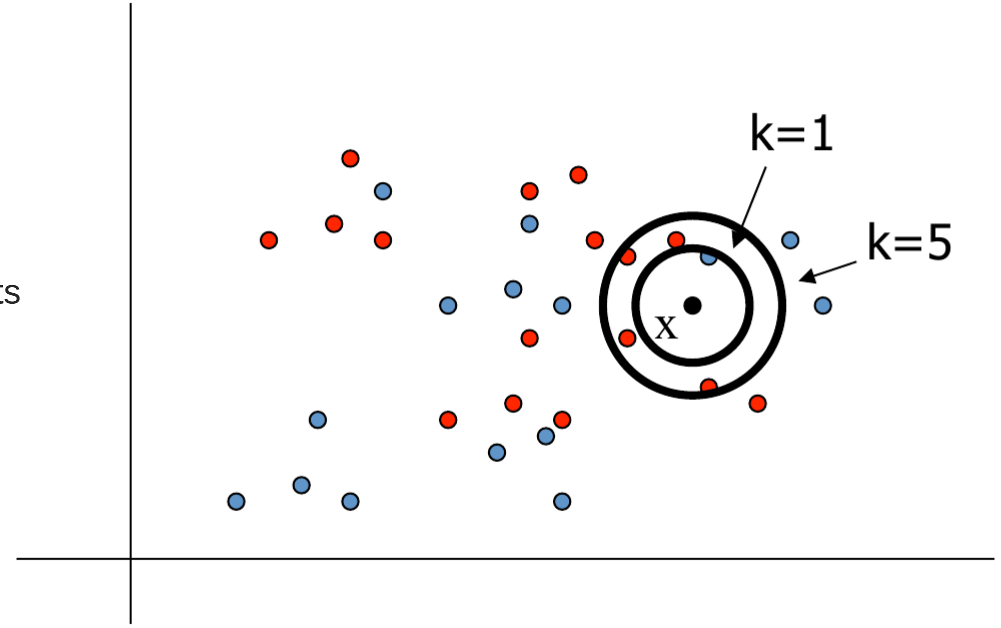
- common values for k: k = 3, k = 5

When to consider:

- Can measure distances between data-points
- Less than 20 attributes per instance
- Lots of training data

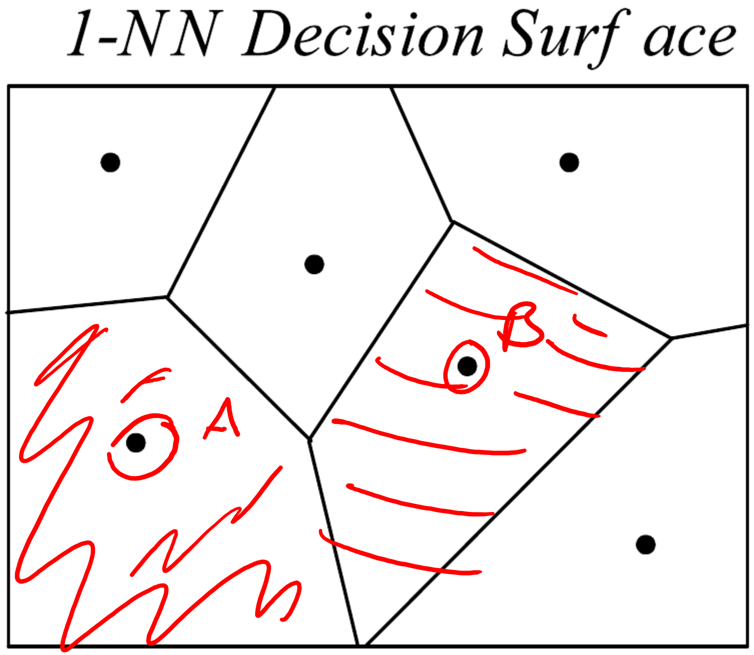
Advantages:

- Training is very fast
- Learn complex target functions
- Similar algorithm can be used for regression



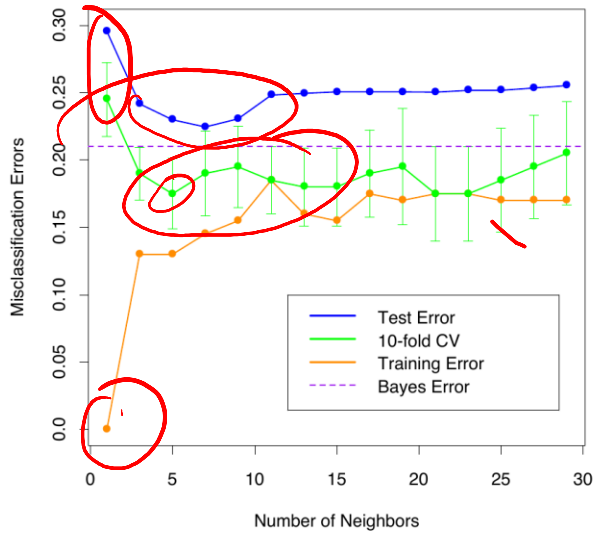
Decision Boundaries

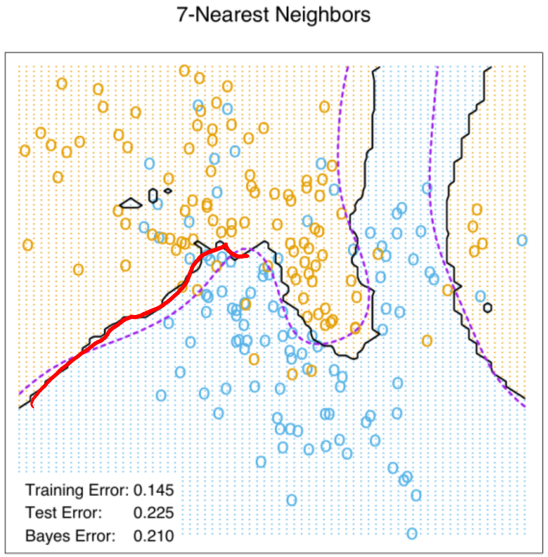
- The nearest neighbour algorithm does not explicitly compute decision boundaries.
- However, the decision boundaries form a subset of the Voronoi diagram for the training data.
- The more data points we have, the more complex the decision boundary can become



Example Result

- Bayes error: error of perfect decision boundary
- Increasing k reduces variance, increases bias
  - K < 7: overfitting
  - K > 7: underfitting
- Has to be selected by cross-validation





Distance Metrics

Most common distance metric is **Euclidean distance (ED)**:

$$d(\boldsymbol{x}, \boldsymbol{y}) = ||\boldsymbol{x} - \boldsymbol{y}|| = \sqrt{\left(\sum_{k=1}^{d} (\boldsymbol{x}_k - \boldsymbol{y}_k)^2\right)}$$

- ED makes sense when different features are commensurate; each is variable measured in the same units.
- If the units are different, say length and weight, data needs to be normalized:

$$\tilde{\boldsymbol{x}} = (\boldsymbol{x} - \boldsymbol{\mu}) \oslash \boldsymbol{\sigma}$$

  - Mean $\boldsymbol{\mu}$, standard deviation $\boldsymbol{\sigma}$, element-wise division $\oslash$
  - I.e. resulting input dimensions are zero mean, unit variance

Distance Metrics

- **Cosine Distance:** Good for documents, images, etc.

$$d(\boldsymbol{x}, \boldsymbol{y}) = 1 - \frac{\boldsymbol{x}^T \boldsymbol{y}}{\|\boldsymbol{x}\| \|\boldsymbol{y}\|}$$

- **Hamming Distance:** For string data / categorical features

$$d(\boldsymbol{x}, \boldsymbol{y}) = \sum_{k=1}^{d} (\boldsymbol{x}_k \neq \boldsymbol{y}_k)$$

- **Manhattan Distance:** Coordinate-wise distance

$$d(\boldsymbol{x}, \boldsymbol{y}) = \sum_{k=1}^{d} |\boldsymbol{x}_k - \boldsymbol{y}_k|$$

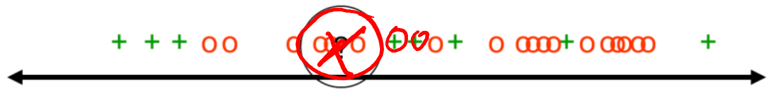
Distance Metrics

- **Mahalanobis Distance:** Normalized by the sample covariance matrix – unaffected by coordinate transformations.

$$d(\boldsymbol{x}, \boldsymbol{y}) = ||\boldsymbol{x} - \boldsymbol{y}||_{\boldsymbol{\Sigma}^{-1}} = \sqrt{(\boldsymbol{x} - \boldsymbol{y})^T \boxed{\boldsymbol{\Sigma}^{-1}} (\boldsymbol{x} - \boldsymbol{y})}$$

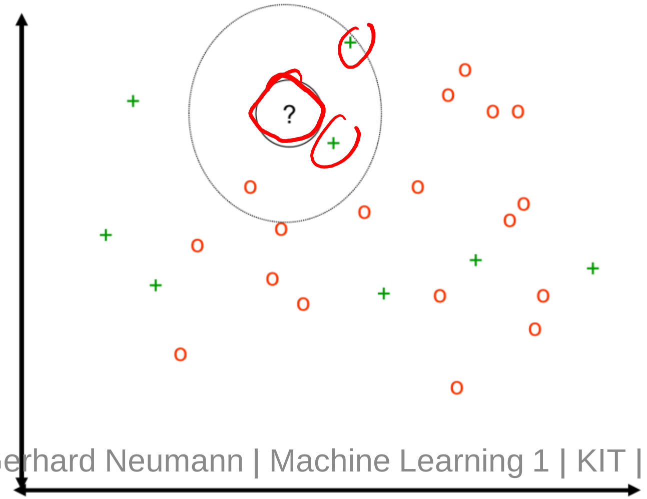
k-NN and irrelevant features

- No irrelevant input:



- Class can be clearly determined

- Added irrelevant dimension:

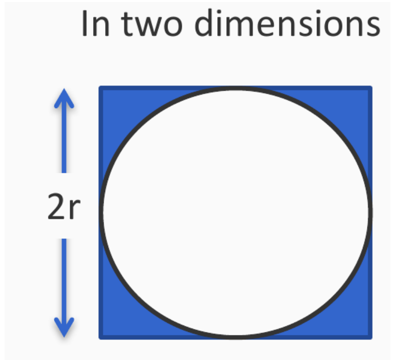


- Neighborhood needs to be increased
- Heavily affected by noise
- Needs much more training data

The performance of k-NN degrades with more (irrelevant) dimensions

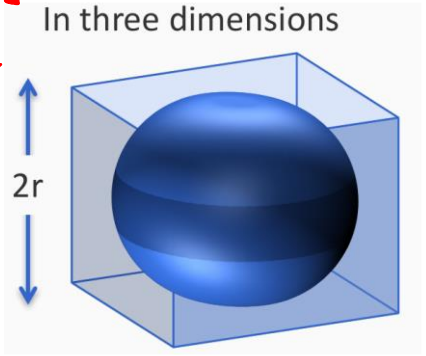
Curse of dimensionality

Example 1: What fraction of the points in a cube lie outside the sphere inscribed in it?



$$\frac{4r^2 - \pi r^2}{4r^2}$$

$$1 - \frac{\pi r^2}{\boxed{4r^2}}$$
$$= 1 - \frac{\pi}{4}$$



$$1 - \frac{4/3\pi r^3}{8r^3}$$
$$= 1 - \frac{\pi}{6}$$

- For $d \to \infty$ this fraction approaches 1!

Curse of dimensionality

Most of the points in high dimensional spaces are far away from the origin!

- In 2 or 3 dimensions, most points are near the center
- Need more data to “fill up the space”

Bad news for nearest neighbor classification in high dimensional spaces

- Even if most/all features are relevant, in high dimensional spaces, most points are equally far from each other!
- “Neighborhood” becomes very large

Remedies (to some extend):

- Most “real-world” data is not uniformly distributed in the high dimensional space
- E.g.: Dimensionality reduction techniques, manifold learning
- Feature selection (pick a good set based on a validation set)

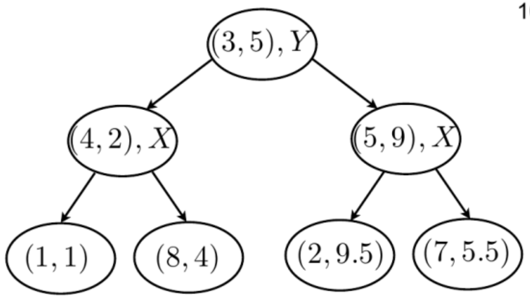
Finding the neighbours: KD-Trees

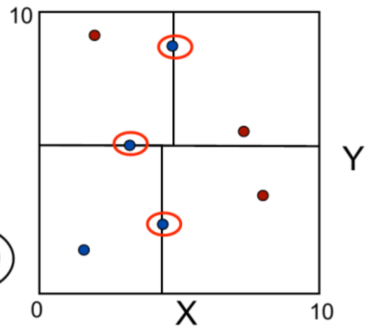
Problem: given a sample set $S = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$, find the k-NNs of test point $\boldsymbol{x}^*$.

Building the tree: for each non-leaf node

- Choose dimension (e.g., longest hyperrectangle).
- Choose median as pivot
- Split node according to (pivot, dimension).

Balanced tree, binary space partitioning.





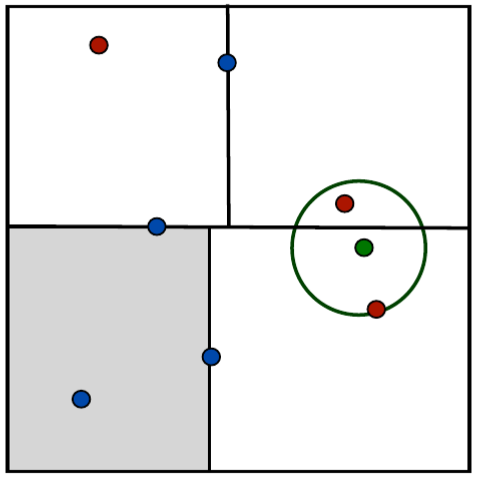
KD-Trees

Finding the neighbours (k = 1):

- Find region containing **x** (starting from root node, move to child node based on node test).
- Save region point **x*** = $\mathbf{x}_0$ as current best.
- Move up tree and recursively search regions intersecting hypersphere $S(\boldsymbol{x}, \|\boldsymbol{x} - \boldsymbol{x}^*\|)$
- Update **x*** if new nearest neighbour has been found

For k > 1:

- Same algorithm, but save **x*** as k-nearest neighbour.
- **Complexity:** O(k log N)



k-NN Summary

Probably the oldest and simplest learning algorithm

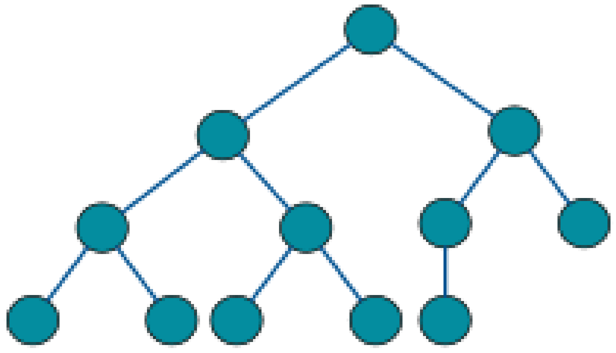
- Prediction is expensive.
- Efficient data structures help. k-D trees: the most popular, works well in low dimensions
- Good baseline: If you do not beat k-NN, you are doing something wrong

Requires a distance measure between instances

- Partitions the space into a Voronoi Diagram
- Beware the curse of dimensionality

Trees and Forests

Regression and Classification Trees



Grow a binary tree

- At each node, “split” the data into two “daughter” nodes.
- Splits are chosen using a splitting criterion.
- Bottom nodes are “terminal” nodes.

For regression:

- the predicted value at a node is the **average response** variable for all observations in the node.

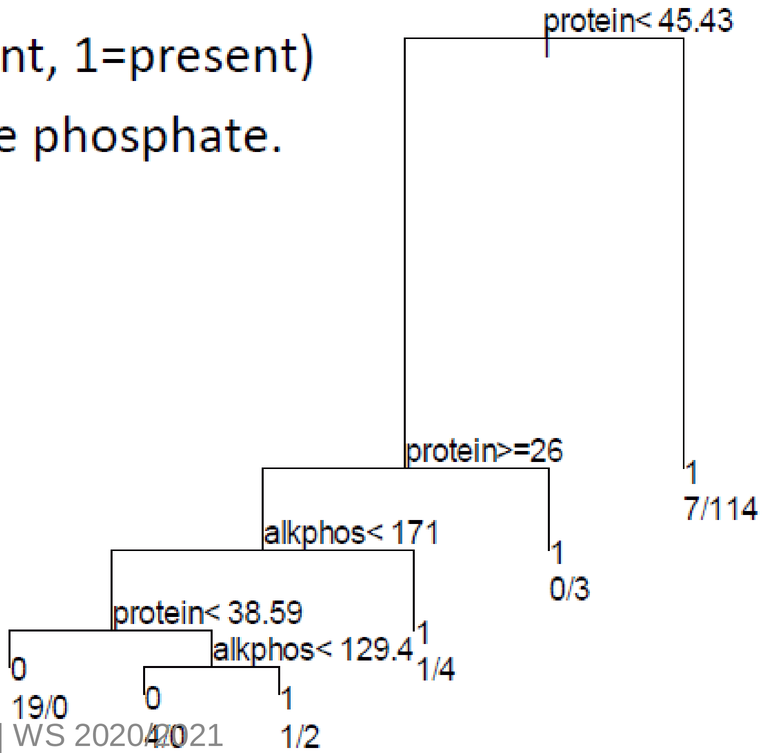
For classification:

- the predicted class is the **most common class** in the node (majority vote).
- Can also get estimated probability of membership in each of the classes

A classification tree

Predict hepatitis (0=absent, 1=present) using protein and alkaline phosphate.

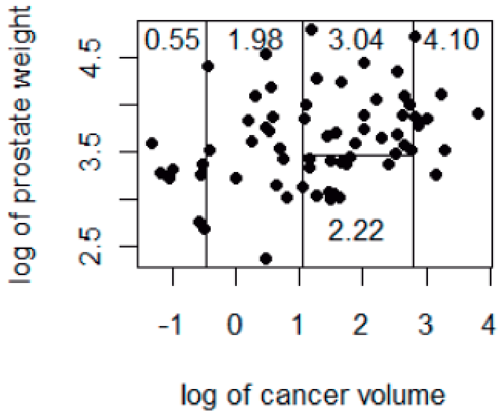
“Yes” goes left.

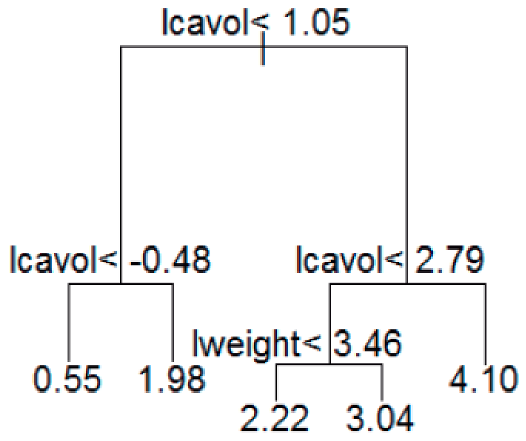


A regression tree

Predict (log) prostate specific antigen from

- Log cancer volumne
- Log prostate weight





Splitting criterion

- **Regression:** Minimum residual sum of squares

$$\mathrm{RSS} = \sum_{\text{left}} (y_i - \bar{y}_L)^2 + \sum_{\text{right}} (y_i - \bar{y}_R)^2$$

  - where $\bar{y}_L$ and $\bar{y}_R$ are the average label values in the left and right subtree
  - Split such that variance in subtrees is minimized

Splitting criterion (thats the second mathy slide…)

- **Classification:** Minimum entropy in subtrees

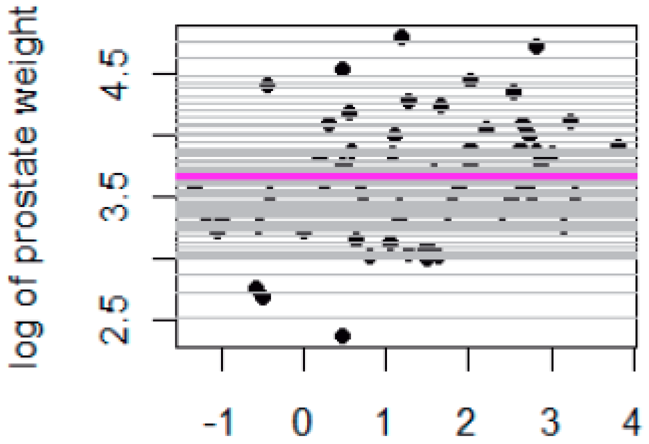
$$\mathrm{score} = N_L H(p_\mathrm{L}) + N_R H(p_\mathrm{R})$$

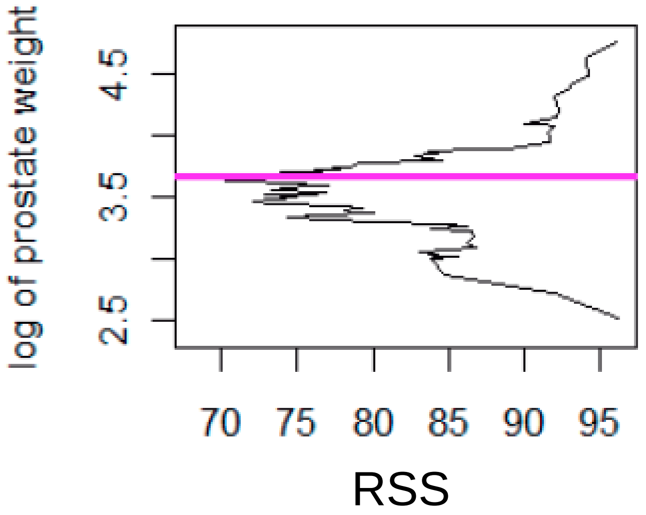
  - where $H(p_L) = -\sum_k p_L(k) \log p_L(k)$ is the entropy in the left sub-tree
  - and $p_L(k)$ is the proportion of class k in left tree

- **Entropy is a measure of uncertainty**

  - Split such that class-labels in sub-trees are „pure“

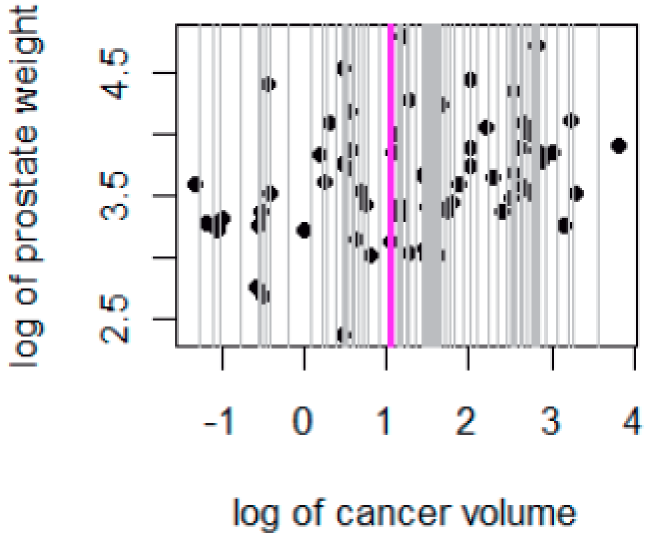
Finding the best horizontal split

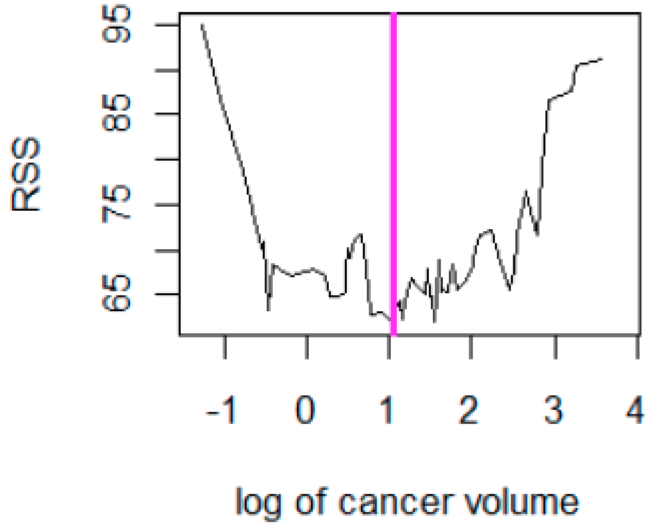




Best horizontal split is at 3.67 with RSS = 68.09.

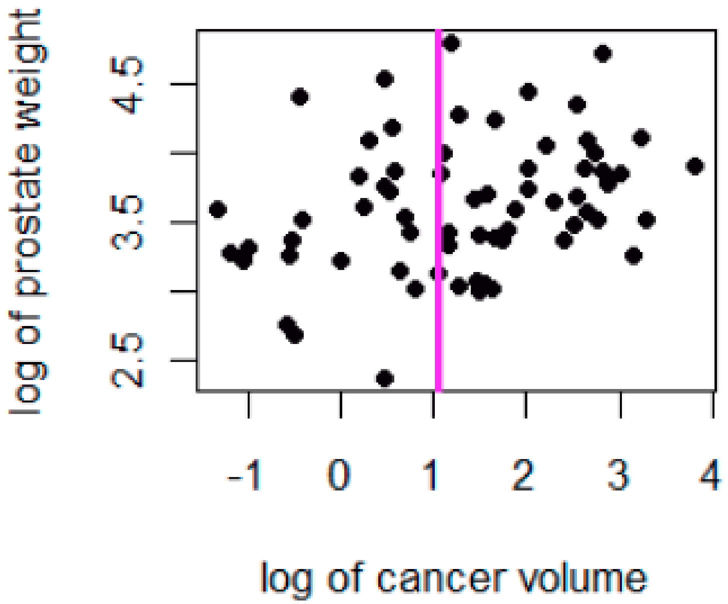
Finsing the best vertical split

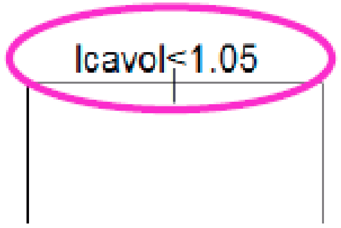




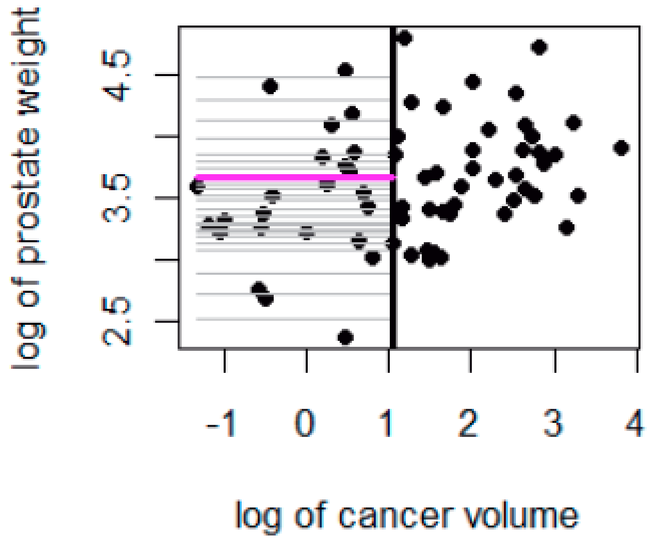
Best vertical split is at 1.05 with RSS = 61.76.

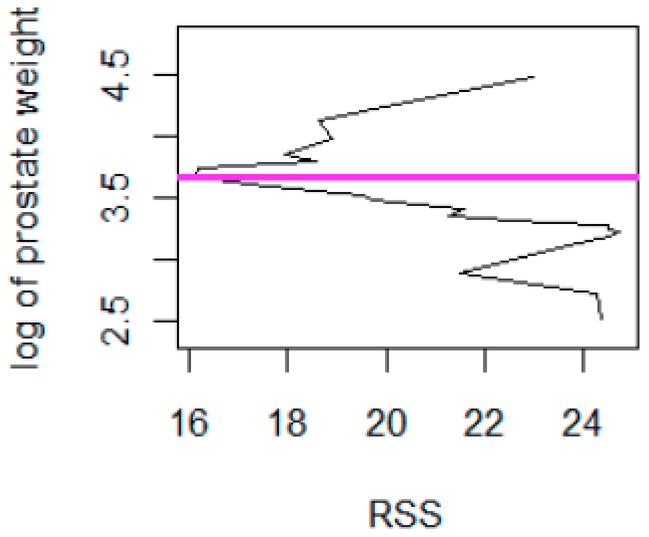
Creating the root node





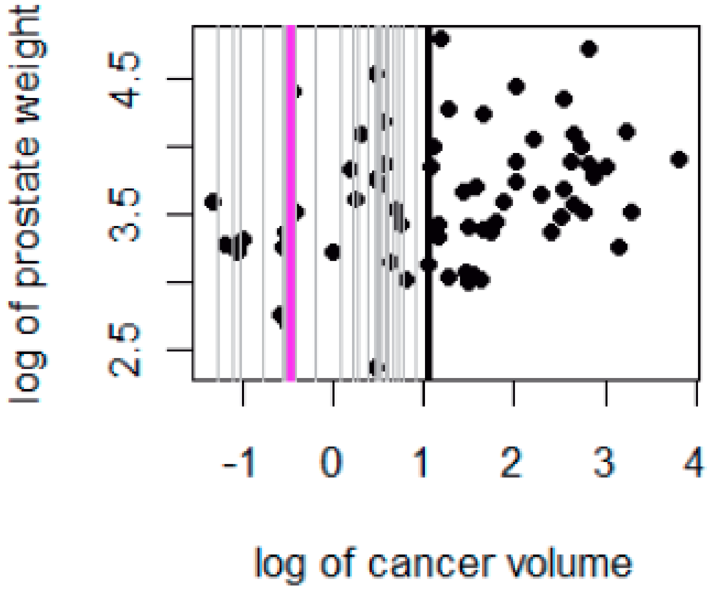
Finding the best split in the left node

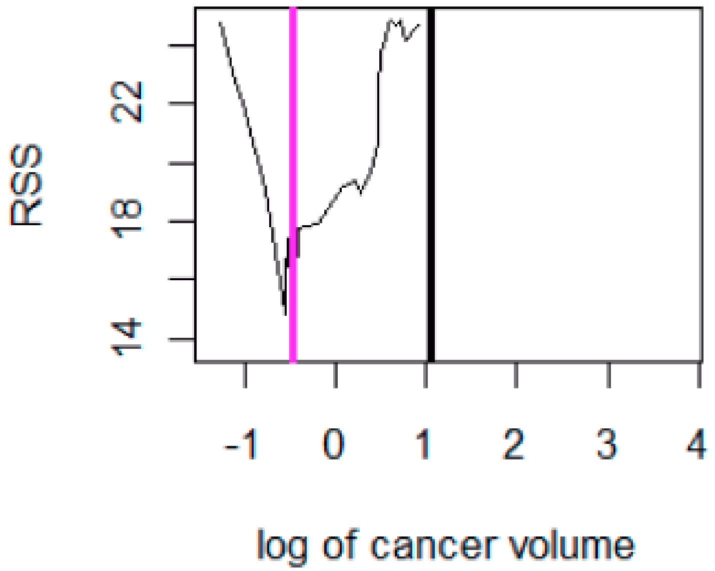




Best horizontal split is at 3.66 with RSS = 16.11.

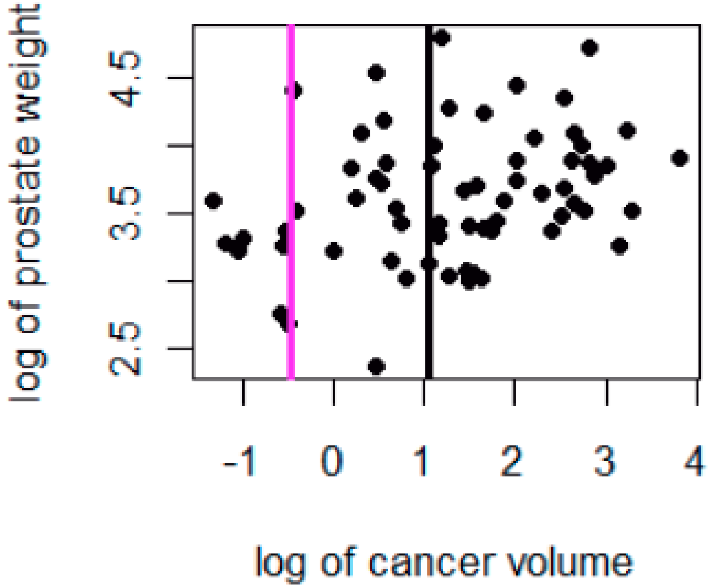
Finding the best split in the left node

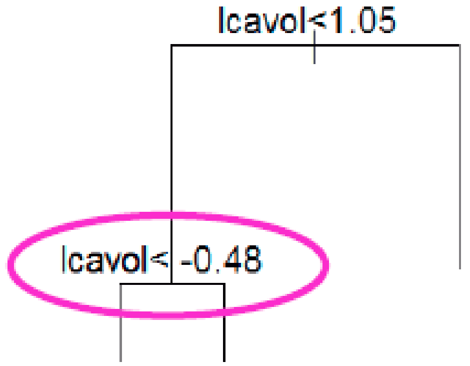




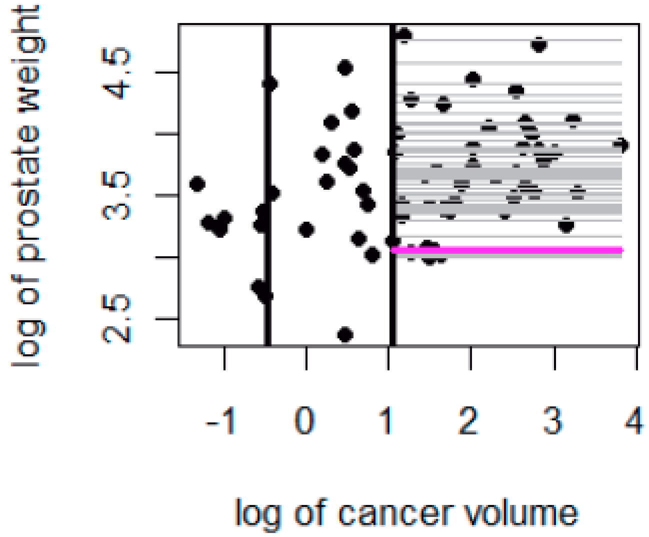
Best vertical split is at -.48 with RSS = 13.61.

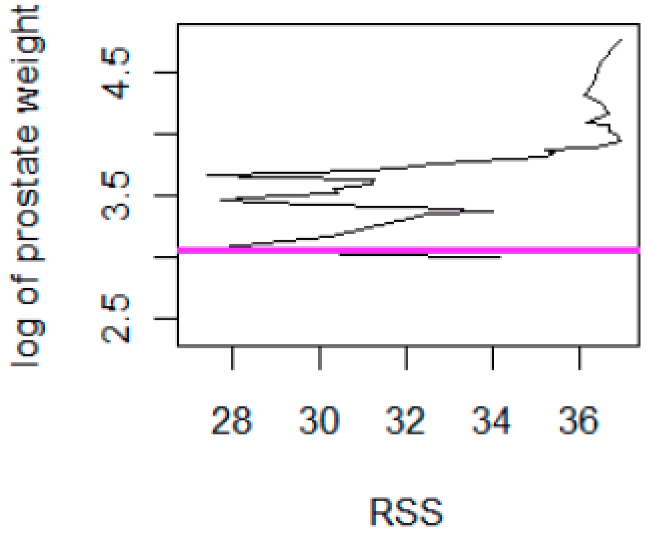
Building the regression tree…





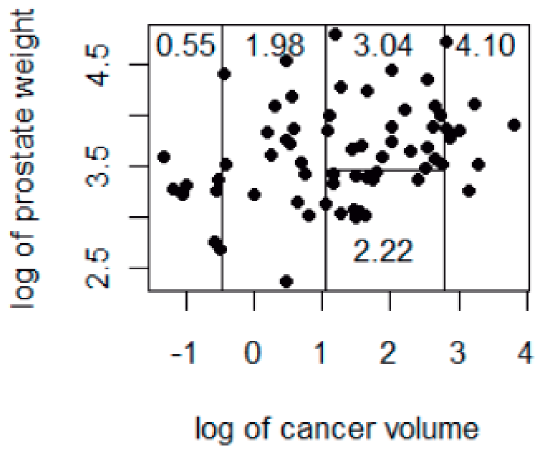
Finding the best split in the right node…

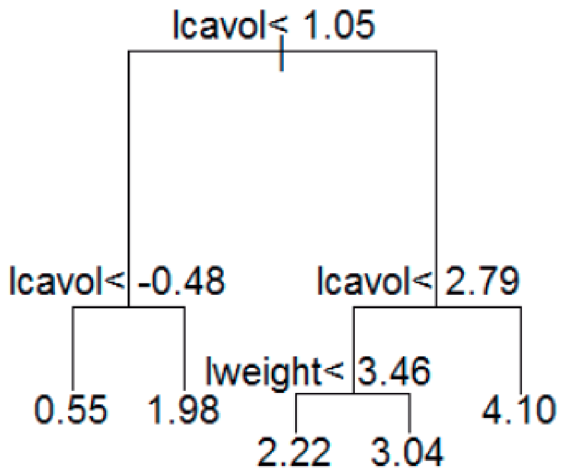


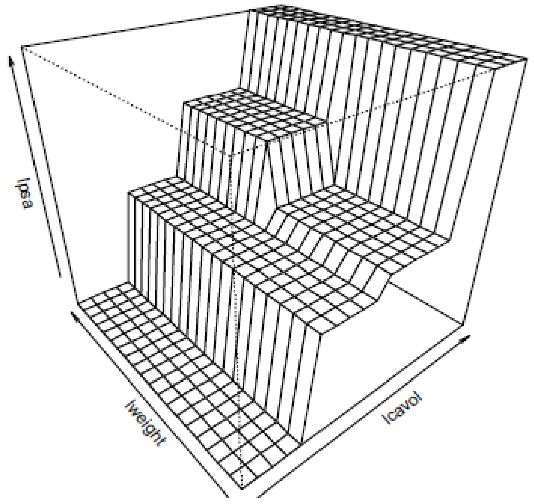


Best horizontal split is at 3.07 with RSS = 27.15.

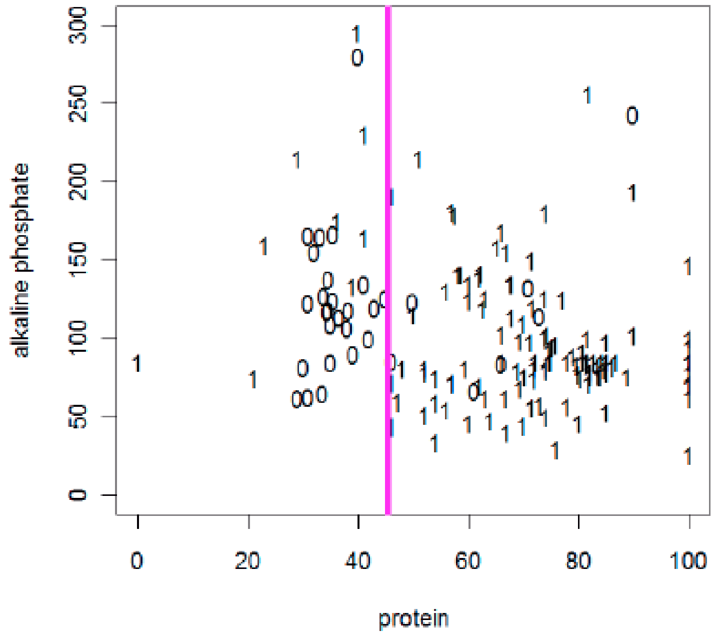
Skipping some steps… final result

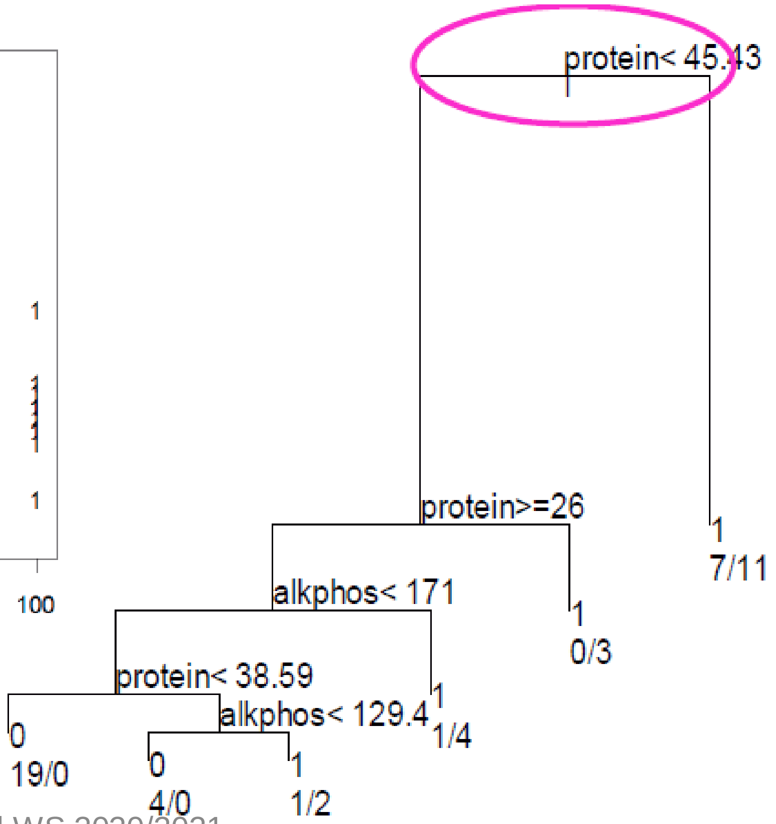




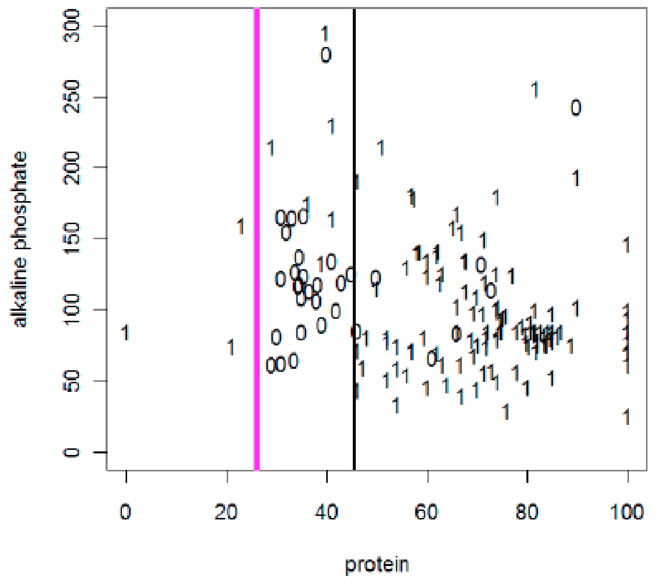


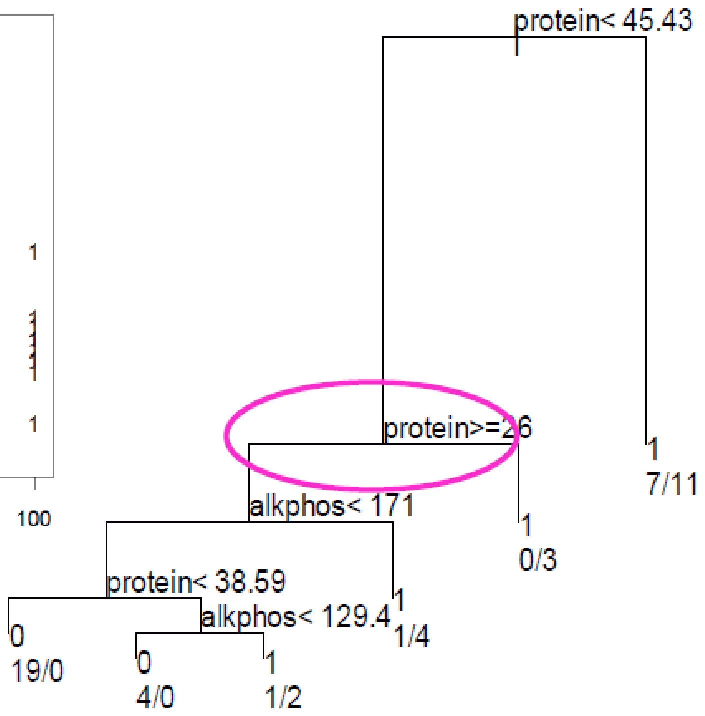
Classification Tree



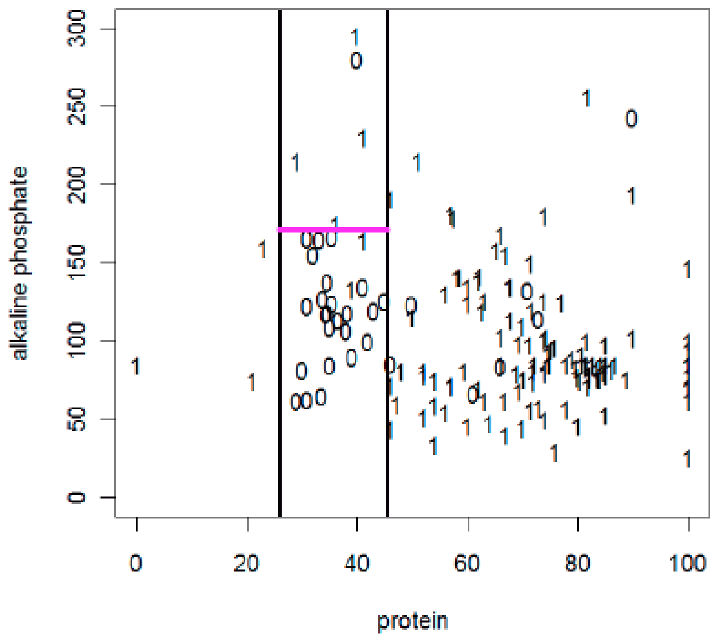


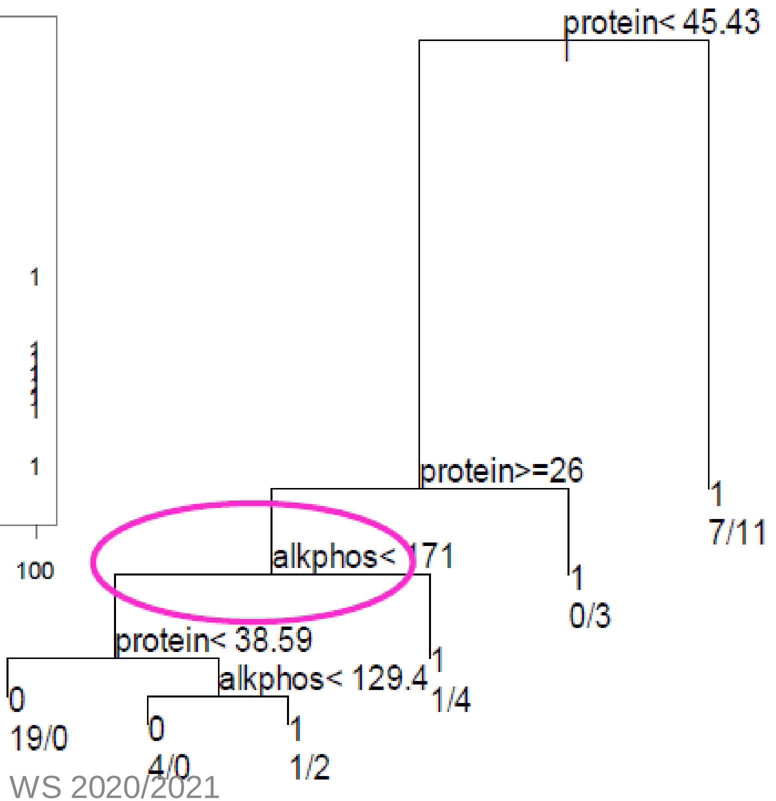
Classification Tree



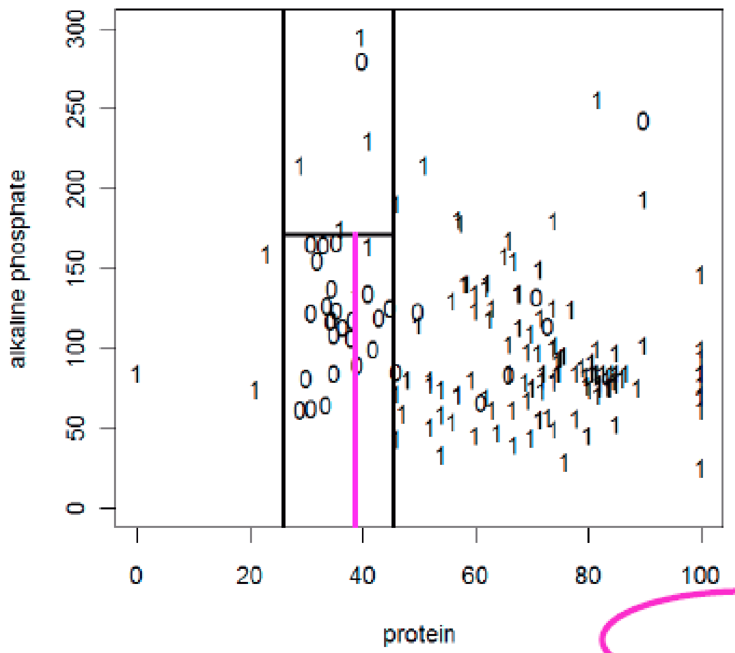


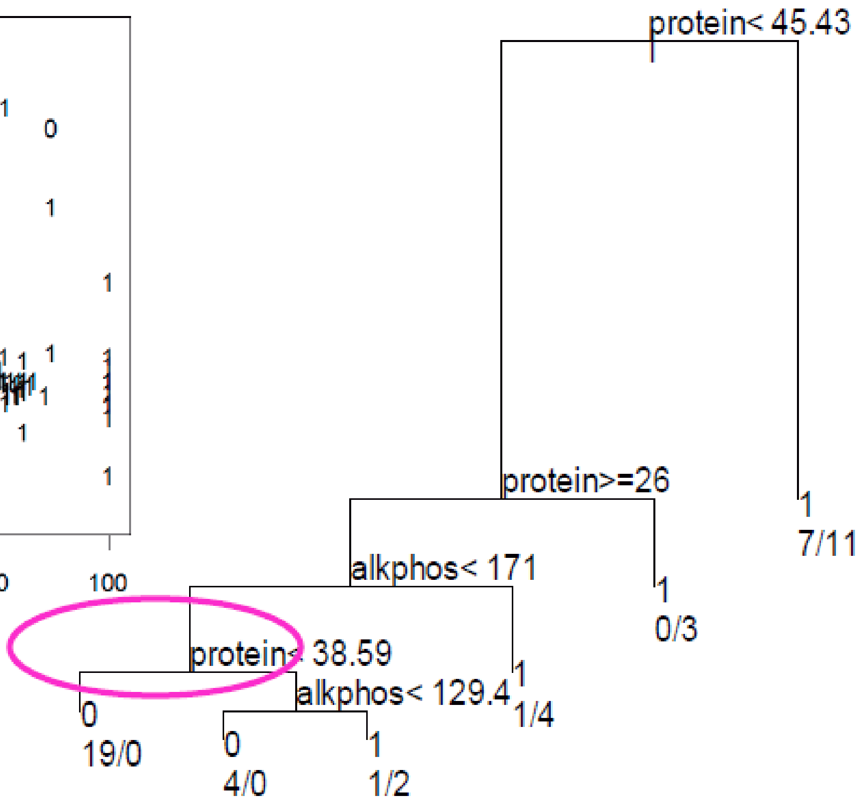
Classification Tree



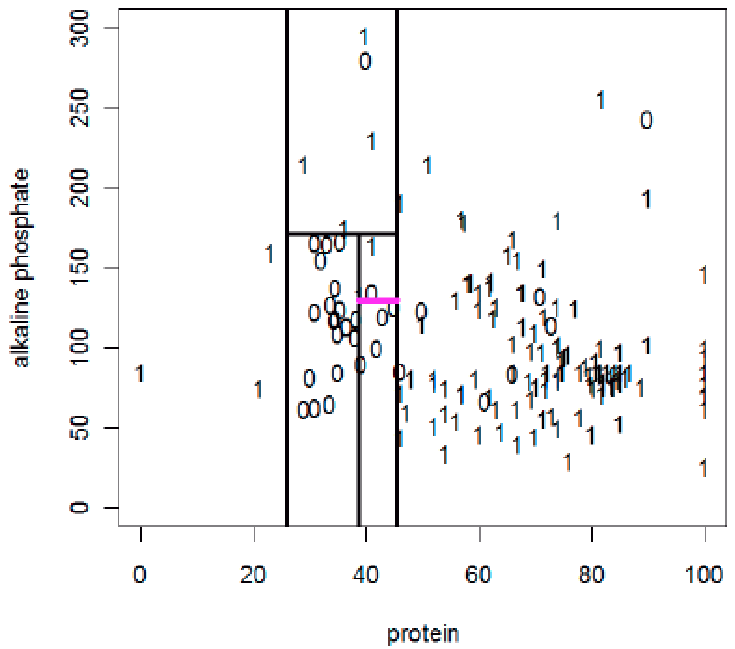


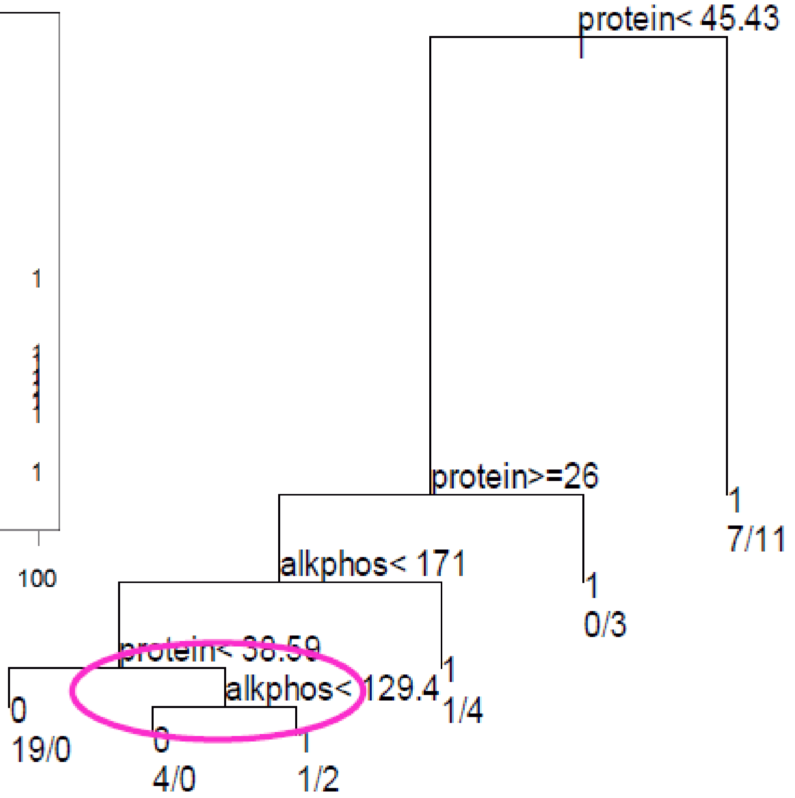
Classification Tree





Classification Tree





When do we stop?

There are many stopping criterias, the 2 main ones are:

Stop if:

- Minimum number of samples per node
- Maximum depth

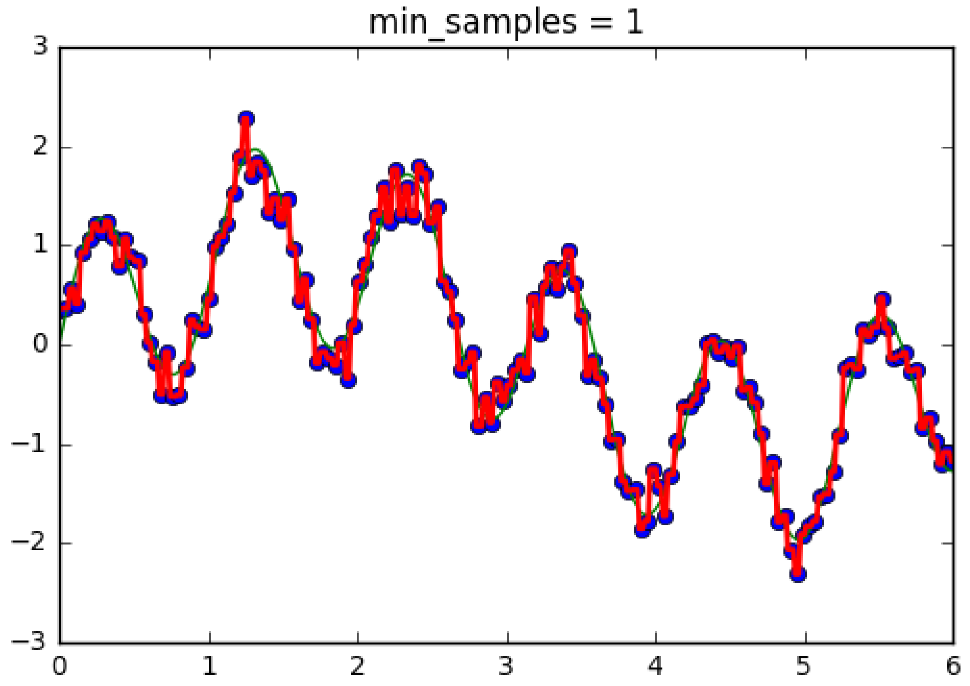
… has been reached

Both criterias again influence the **complexity** of the tree !

Controlling the tree complexity

Small number of samples per leaf:

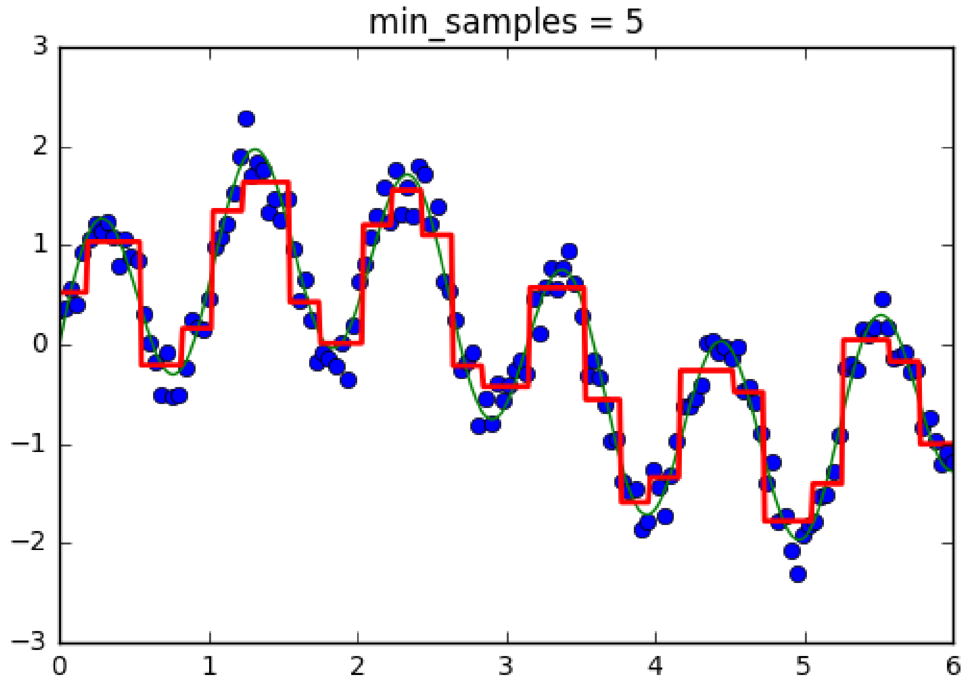
- Tree is very sensitive to noise



Controlling the tree complexity

Small number of samples per leaf:

- Tree is very sensitive to noise



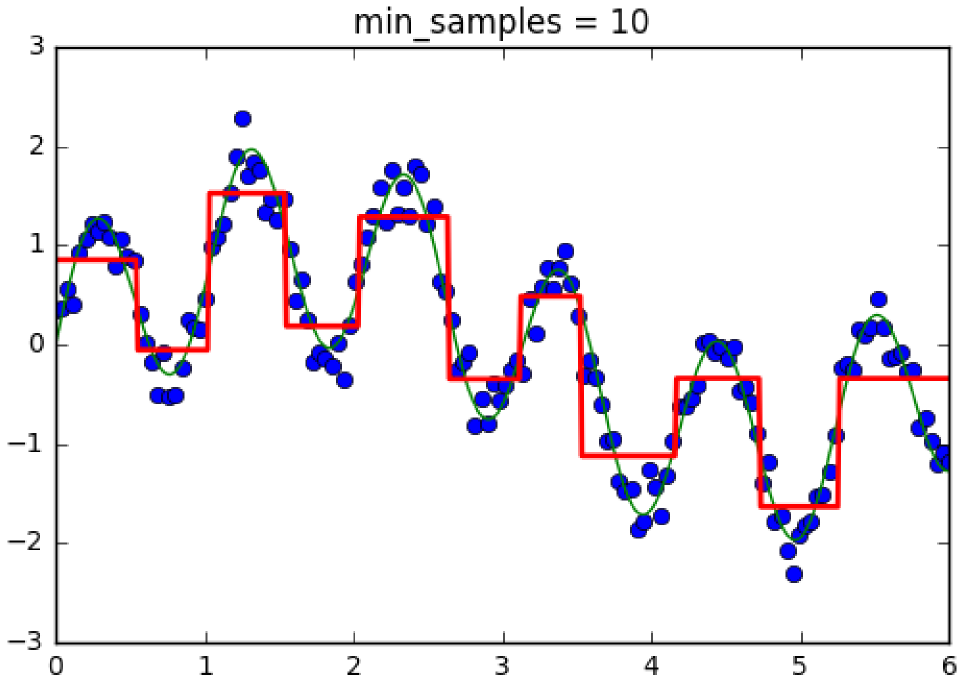
Controlling the tree complexity

Small number of samples per leaf:

- Tree is very sensitive to noise

Large number of samples per leaf:

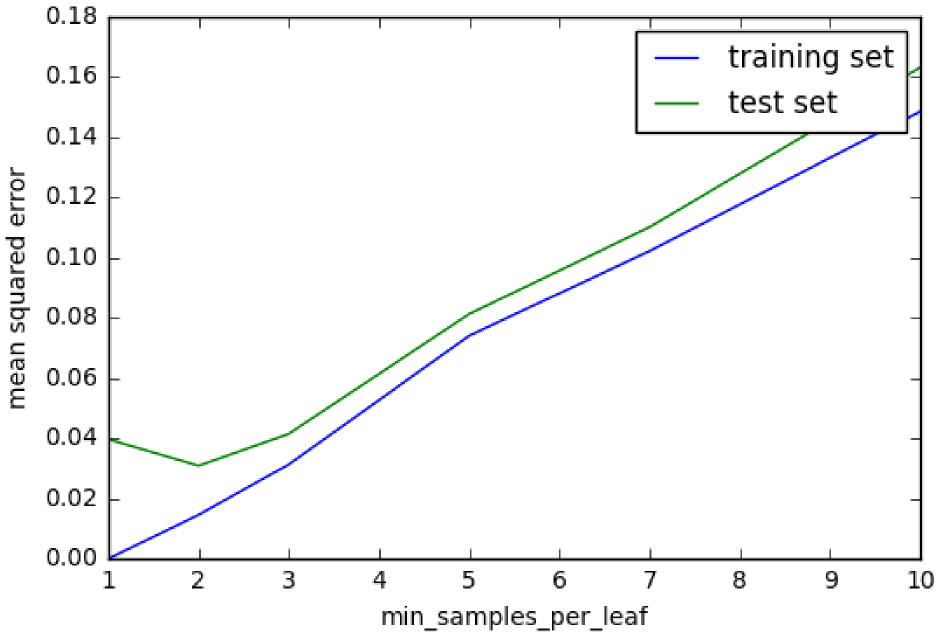
- Tree not expressive enough



Model-Selection for Regression Trees

Evaluate error on validation-set

- Overfitting for min_samples = 1
- Underfitting for min_sampes > 2
- Larger min_samples -> lower complexity



Classification and Regression Trees

Advantages

- Applicable to both regression and classification problems.
- Handle categorical predictors naturally.
- Computationally simple and quick to fit, even for large problems.
- No formal distributional assumptions
- Can handle highly non-linear interactions and classification boundaries.
- Automatic variable selection.
- Very easy to interpret if the tree is small.

Classification and Regression Trees (CART)

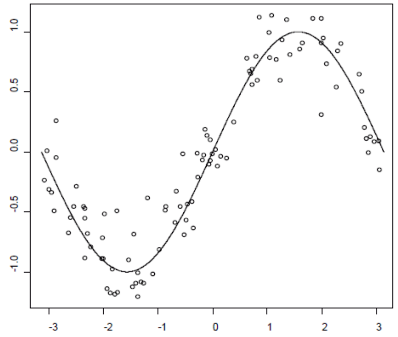
Disadvantages

- *Accuracy* - current methods, such as NNs, support vector machines and ensemble classifiers often have 30% lower error rates than CART.
- *Instability* – if we change the data a little, the tree picture can change a lot. So the interpretation is not as straightforward as it appears.

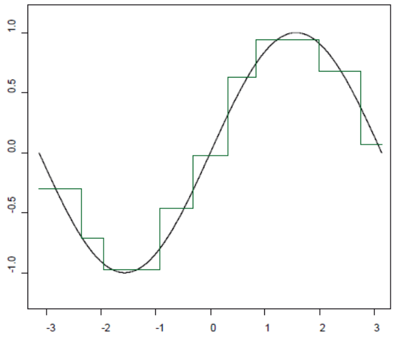
Nowadays, we can do better! **Random Forests!**

Key Idea: Use multiple trees to improve accuracy

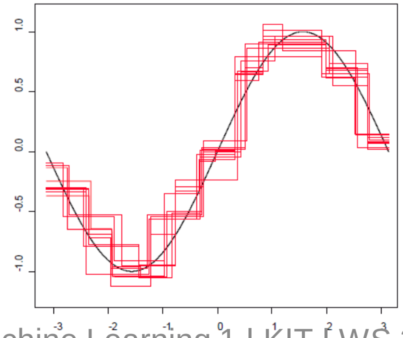
Data and Underlying Function



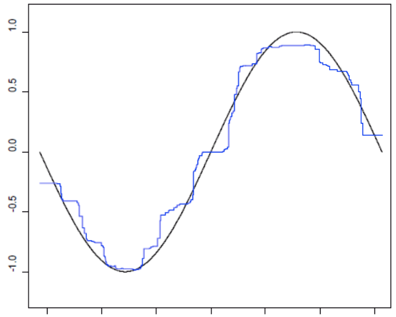
Single Regression Tree



10 Regression Trees

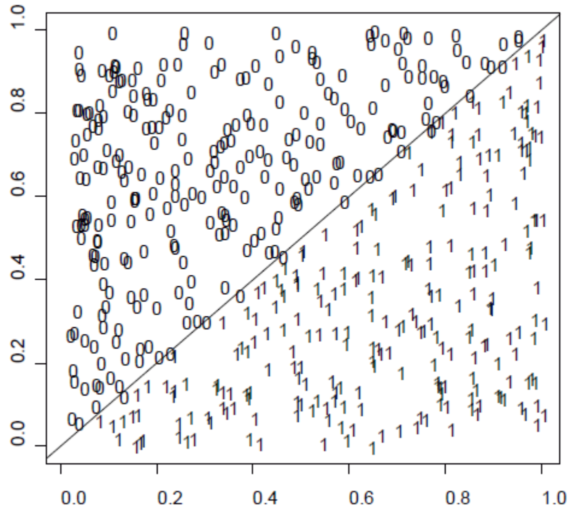


Average of 100 Regression Trees

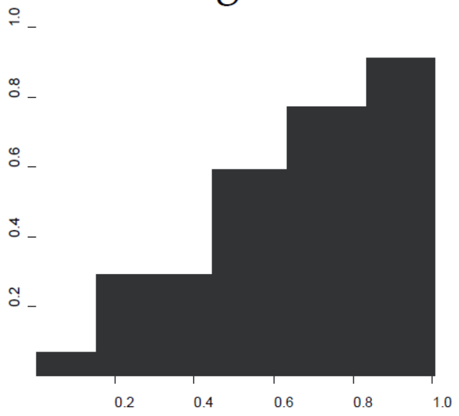


Key Idea: Use multiple trees to improve accuracy

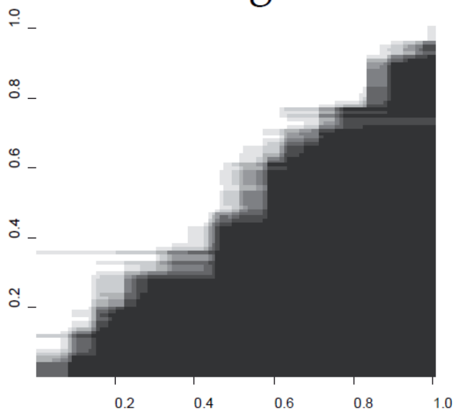
Hard problem for a single tree:



Single tree:



25 Averaged Trees:

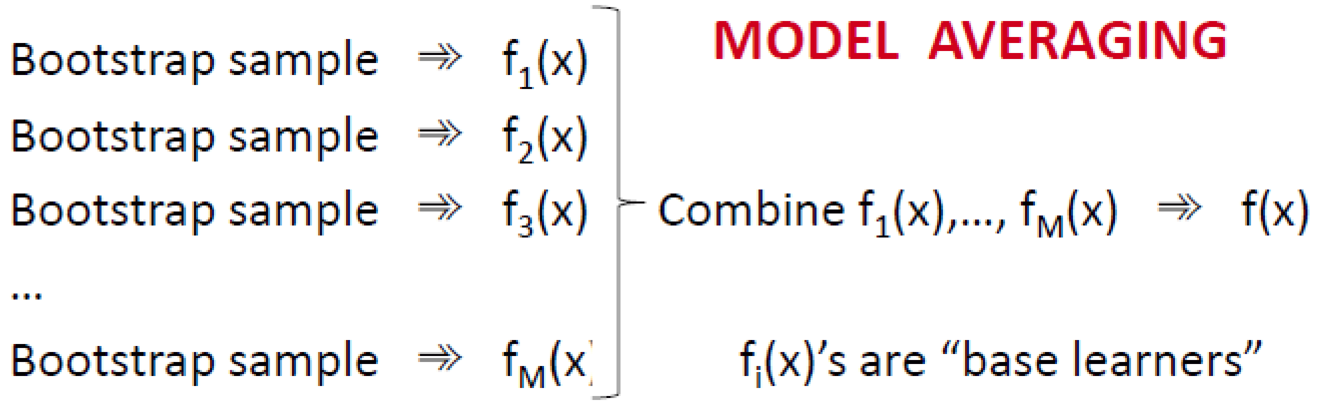


How do we get **variability** in the **trees**?

Bagging (Bootstrap Aggregating)

Breiman, “Bagging Predictors”, *Machine Learning*, 1996.

Fit classification or regression models to **bootstrap samples** from the data and combine by **voting** (classification) or **averaging** (regression).



MODEL AVERAGING

Bootstrap sample $\Rightarrow$ $f_1(x)$

Bootstrap sample $\Rightarrow$ $f_2(x)$

Bootstrap sample $\Rightarrow$ $f_3(x)$

...

Bootstrap sample $\Rightarrow$ $f_M(x$

Combine $f_1(x),\ldots, f_M(x)$ $\Rightarrow$ $f(x)$

$f_i(x)$’s are “base learners”

Bagging (Bootstrap Aggregating)

A **bootstrap sample** is chosen at **random *with* replacement** from the data. Some observations end up in the bootstrap sample more than once, while others are not included (“out of bag”).

Variance reduction

In general: $\mathrm{Var}\left[\frac{1}{M}\sum_{i=1}^{M} X_i\right] = \frac{1}{M^2}\mathrm{Var}\left[\sum_{i=1}^{M} X_i\right] = \frac{1}{M}\mathrm{Var}\left[X\right], \quad \text{if } X \text{ i.i.d.}$

- i.e., ideally, the variance would reduce linearly with the number of trees

In practice: $\mathrm{Var}\left[\frac{1}{M}\sum_{i=1}^{M} \mathrm{Tree}_i\right] > \frac{1}{M}\mathrm{Var}\left[\mathrm{Tree}\right], \quad \text{as trees are still correlated}$

- But variance reduction is still significant
- Bagging reduces the **variance** of the base learner but has limited effect on the **bias**
  - I.e. no overfitting: The more trees the better
- It’s most effective if we use *strong* base learners that have very little bias but high variance (unstable). E.g. trees.

Bagging CART

Dataset	# cases	# vars	# classes	CART	Bagged CART	Decrease %
Waveform	300	21	3	29.1	19.3	34
Heart	1395	16	2	4.9	2.8	43
Breast Cancer	699	9	2	5.9	3.7	37
Ionosphere	351	34	2	11.2	7.9	29
Diabetes	768	8	2	25.3	23.9	6
Glass	214	9	6	30.4	23.6	22
Soybean	683	35	19	8.6	6.8	21

Leo Breiman (1996) "Bagging Predictors", Machine Learning, 24, 123-140.

Randomization

Grow a **forest** of many trees. (R default is 500)

Grow each tree on an independent **bootstrap sample** from the training data.

- Sample N cases at random with replacement.

At each node:

1. Select ***m*** variables **at random** out of all ***M*** possible variables (independently for each node).
2. Find the best split on the selected ***m*** variables.

Grow the trees to maximum depth (classification).

Vote/average the trees to get predictions for new data.

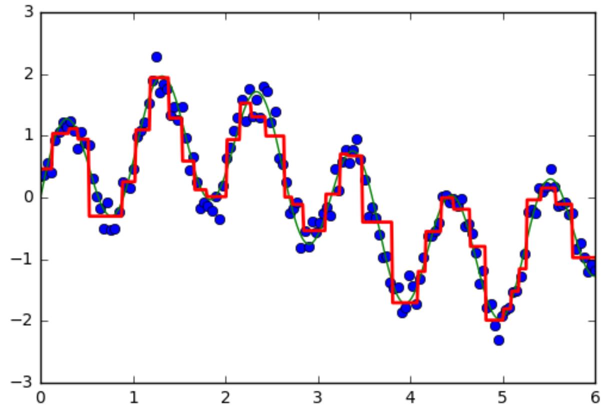
Why does that work?

Intuition: Why randomization?

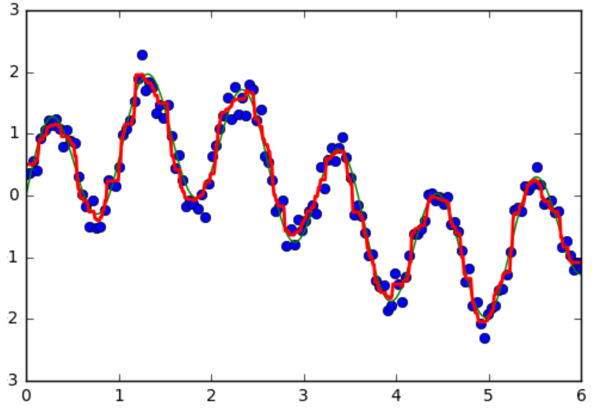
- Increase variability of the single trees
- A single tree is less likely to over-specialize
- The trees are less likely to overfit

Random Regression Forests

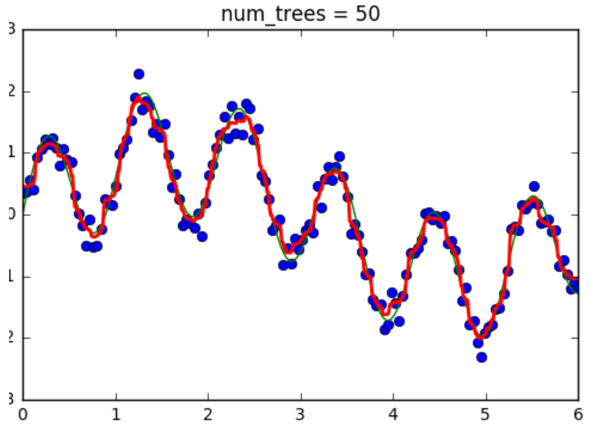
Num Trees = 1



Num Trees = 10



Num Trees = 50



We can represent almost continuous functions!

Random Forests

Dataset	# cases	# vars	# classes	CART	Bagged CART	Random Forests
Waveform	300	21	3	29.1	19.3	17.2
Breast Cancer	699	9	2	5.9	3.7	2.9
Ionosphere	351	34	2	11.2	7.9	7.1
Diabetes	768	8	2	25.3	23.9	24.2
Glass	214	9	6	30.4	23.6	20.6

Leo Breiman (2001) “Random Forests”, Machine Learning, 45, 5-32.

Random Forests

Advantages

- Applicable to both regression and classification problems. Yes
- Handle categorical predictors naturally. Yes
- Computationally simple and quick to fit, even for large problems. Yes
- No formal distributional assumptions (non-parametric). Yes
- Can handle highly non-linear interactions and classification boundaries. Yes
- Automatic variable selection. Yes
- Very easy to interpret if the tree is small. No

Random Forests

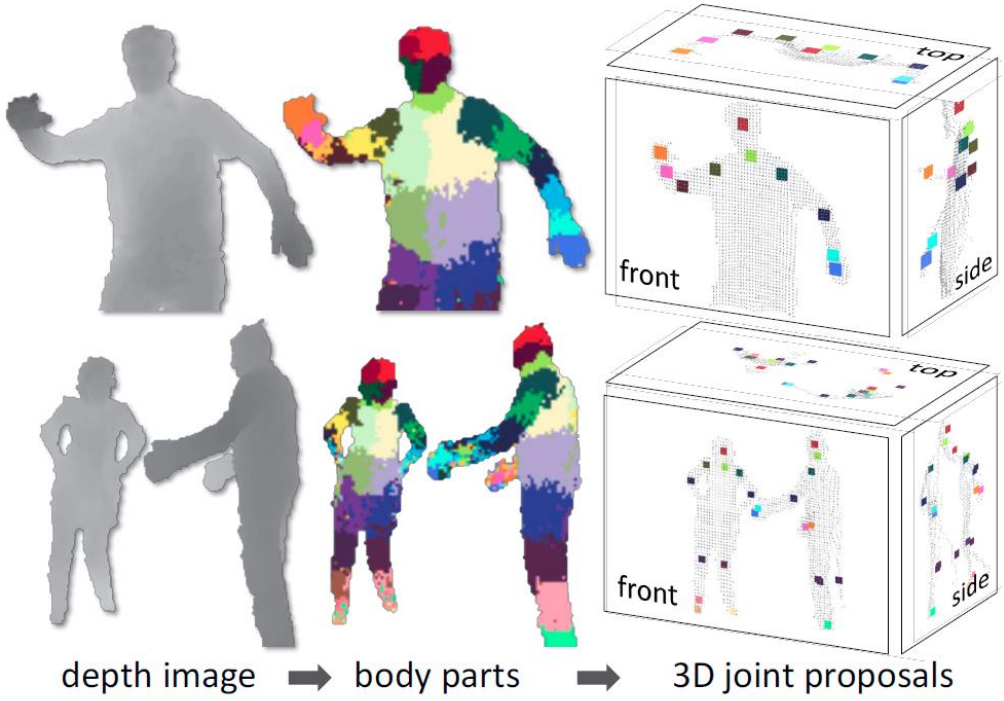
Improve on CART with respect to:

- *Accuracy* – Random Forests is competitive with the best known machine learning methods

- *Instability* – if we change the data a little, the individual trees may change but the forest is relatively stable because it is a combination of many trees.

Random Forests and the Kinect

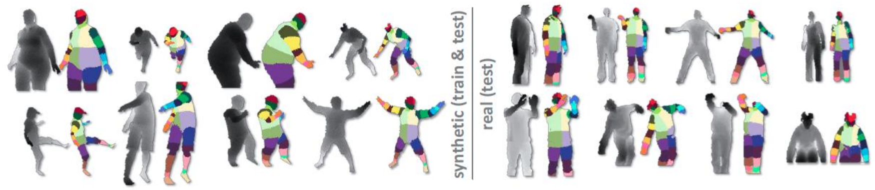
Random Forests and the Kinect



[Jamie Shotton et al., 2011]

Random Forests and the Kinect

Use computer graphics to generate plenty of data



Shotton, et. al., Real-Time Human Pose Recognition in Parts from a Single Depth Image, CVPR 2011

Take-home messages

- **CART:** Binary decision trees can be used for **classification and regression**
- Complexity can be set by minimum samples per leaf
- **Variability in the trees:**
  - Bootstrap
  - Randomized splits
- **Averaging over multiple trees** reduces variance while bias is unaffected!

Self-test questions

You should know now:

- What we mean with non-parametric / instance-based machine learning algorithms ?
- How k-NN works ?
- How to choose the k?
- Why is it hard to use for high-D data ?
- How do search for nearest neighbours efficiently ?
- What a *binary* regression / decision tree is
- What are useful splitting criterions
- How can we influence the model complexity of the tree?
- Why is it useful to use multiple trees and randomization?