

Computational Data Analysis

Classification And Regression Trees

Lars Arvastson and Line Clemmensen

March 19, 2018

Today's Lecture

- ▶ Recap
- ▶ Regression Trees
- ▶ Classification Trees

Recap

Unsupervised clustering

- ▶ Dissimilarity measures
- ▶ K-means clustering
- ▶ K-medoids clustering
- ▶ Hierarchical clustering
 - ▶ cluster-cluster distance
- ▶ Gaussian mixture
- ▶ Validation and model selection
 - ▶ CV?
 - ▶ Gap-statistics
 - ▶ BIC

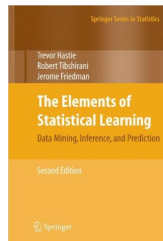


Classification And Regression Trees

- A decision tree is a model that partitions the feature space into regions, each of which is associated with a class label or a numerical value.
- The partitioning is done by a series of binary splits, each of which is based on a feature and a threshold value.
- The resulting regions are then used to predict the class label or the numerical value for a given input.
- Decision trees are a type of supervised learning algorithm that can be used for both classification and regression tasks.
- They are easy to interpret and can handle both numerical and categorical data.
- However, they can be prone to overfitting, especially if the tree is too deep.
- To mitigate this, various techniques have been developed, such as pruning and ensemble methods.
- Decision trees are a fundamental concept in machine learning and are widely used in many applications.

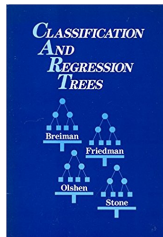
CART

Classification And Regression Trees



The Elements of Statistical Learning:

“Tree based methods partition the feature space into a set of rectangles, and then fit a simple model (like a constant) in each one.”



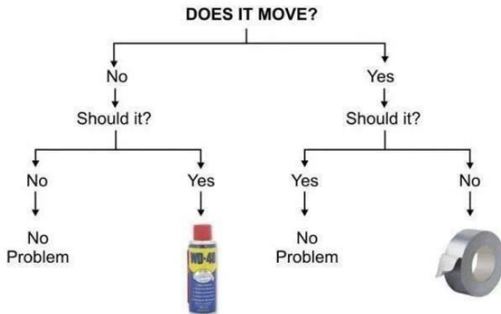
Classification And Regression Trees (1984) by Breiman, Friedman, Olshen and Stone introduced,

- ▶ CART
- ▶ The one-standard-error rule

Decision trees

Like a decision tree
but built on data
instead of expert
knowledge

Engineering Flowchart



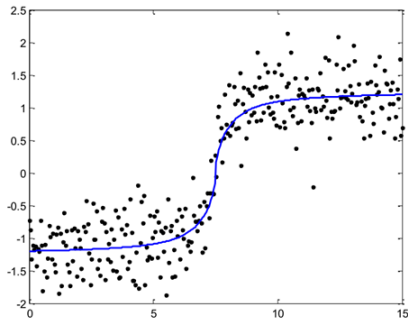
Regression trees

- Regression trees are a type of decision tree where the leaf nodes contain numerical values representing the predicted output for a given input.
- They are used for regression tasks, where the goal is to predict a continuous numerical value.
- Regression trees are built by recursively splitting the data into subsets based on a chosen feature and a threshold value.
- The splitting process is guided by a criterion that measures the quality of the split, such as the reduction in variance or the minimization of the mean squared error.
- The resulting regression tree structure can be visualized as a hierarchical diagram, where the root node branches into internal nodes, which eventually lead to leaf nodes containing the predicted values.
- Regression trees are robust to outliers and non-linear relationships, making them a popular choice for regression analysis.
- They are also computationally efficient and easy to interpret, allowing for a clear understanding of the model's decision-making process.
- Regression trees can be used for both single-output and multi-output regression problems.
- They are often used in combination with other machine learning techniques, such as ensemble methods, to improve predictive performance.
- Regression trees are a fundamental concept in machine learning and are widely used in various applications, including finance, healthcare, and engineering.

A simple example

Regression

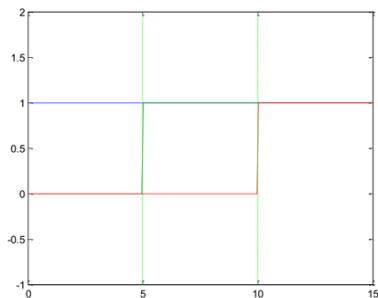
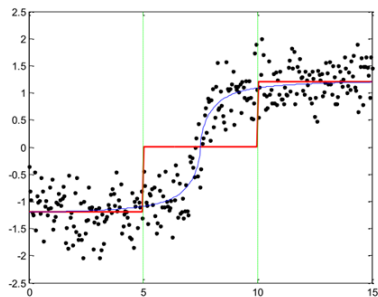
- ▶ True function: $y = f(x)$ (blue)
- ▶ Observation with noise: (x_i, y_i) (black)



A simple example

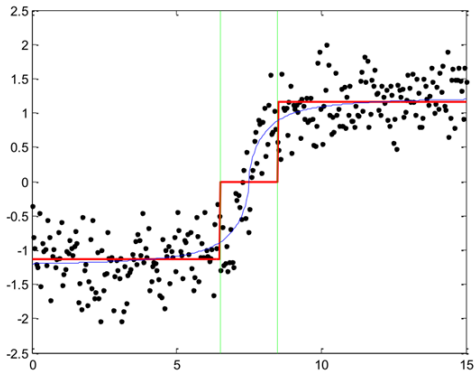
Fit a piecewise polynomial

- ▶ `X = [ones(n,1) double(x>5) double(x>10)] ;`
- ▶ A constant in each interval
- ▶ Evenly spaced knots



A simple example

Can we place the knots in a better way?



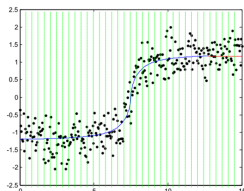
A simple example

Algorithm attempt:

- ▶ Choose a number of knots k .
- ▶ try all possible positions for each knot k
 - ▶ Infinite number of combinations

Try e.g. 100 positions on the x-axis for each knot

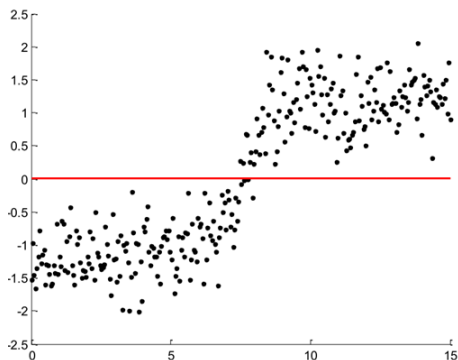
- ▶ 100^k positions to try
- ▶ E.g. $100^5 = 10\,000\,000\,000$ combinations
- ▶ With more than one input variable it gets even worse



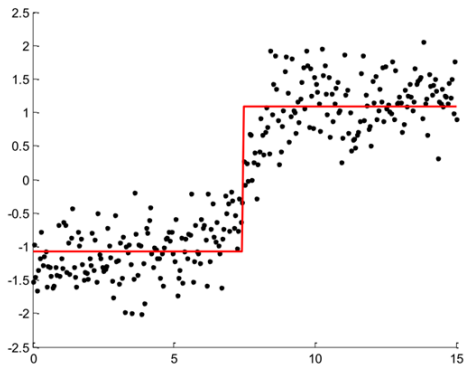
A simple example

New idea:

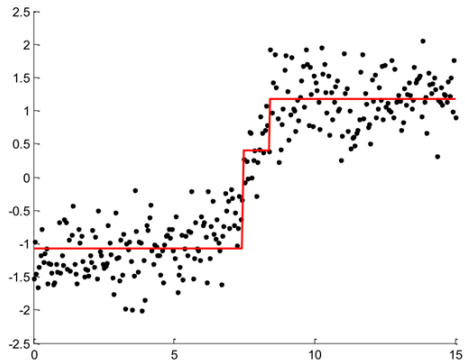
- ▶ Place the knots one after another
- ▶ Place each knot such that the fit is as good as possible



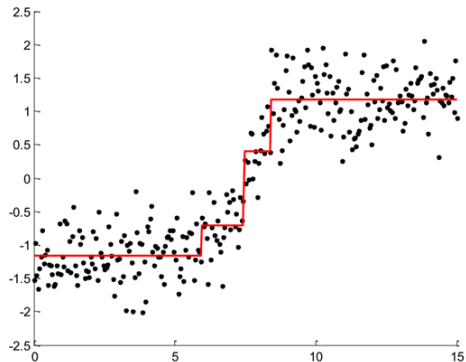
A simple example



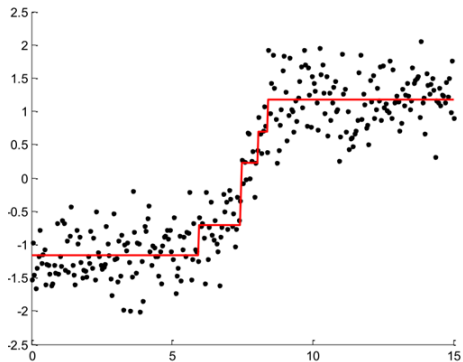
A simple example



A simple example

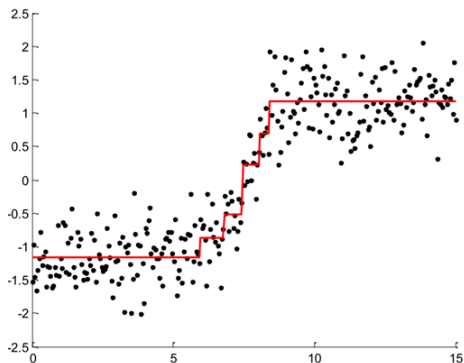


A simple example



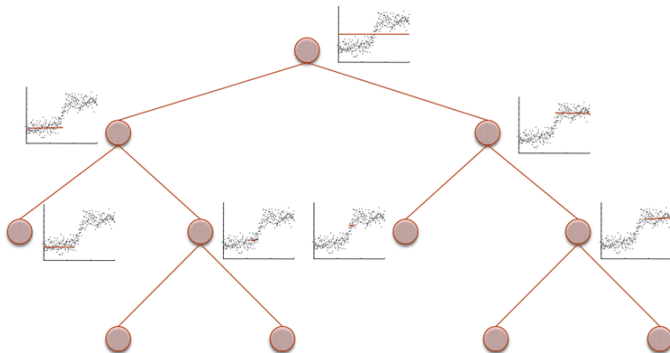
A simple example

At this point, the regression function is pretty good!



Tree representation

Each split can be represented by a parent node split into two child nodes



More than one input variable

Handled in the simplest way possible

For each input variable

 For each split

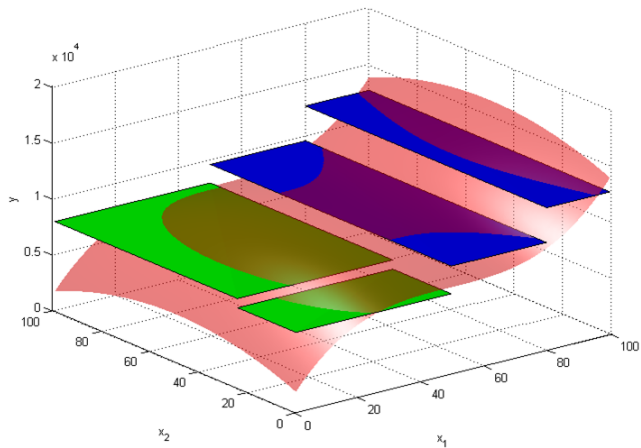
 Evaluate split point

 End

End

Choose the best variable and split

More than one input variable



Questions

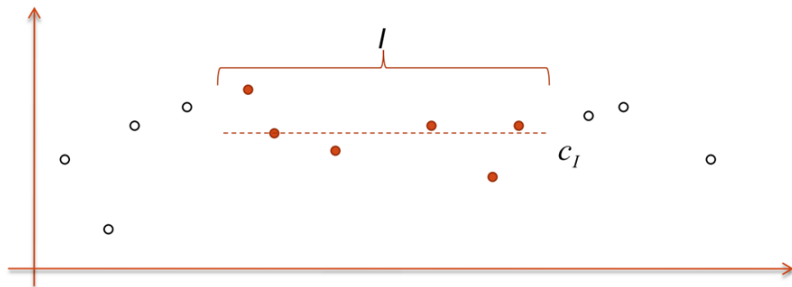
- ▶ What is a good split?
 - ▶ We need to know this to decide where to split
- ▶ How many splits should we try?
 - ▶ We used 100 before - is there a better choice?
- ▶ When do we stop splitting?

What is a good split?

- ▶ The terminal nodes each represent an interval of the input variable(s).
- ▶ We choose to represent the outcome in this interval by a **constant** function.
- ▶ As for most regression problems, we say that a constant function is good if it has low **residual sum of squares** when compared to training outcome data

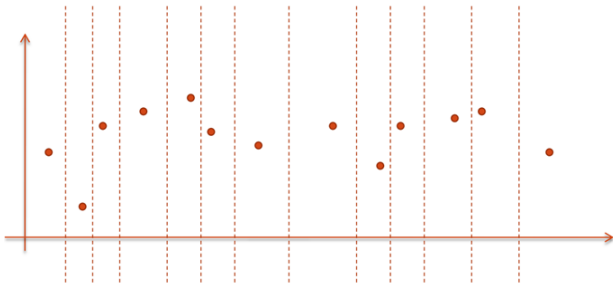
What is a good split?

- ▶ Prediction \hat{y} is a constant in each interval.
- ▶ Residual sum of squares (RSS) in interval I :
$$RSS_I = \sum_{i \in I} (y_i - \hat{y})^2$$
- ▶ Minimal when $\hat{y} = \sum_{i \in I} y_i / n$



How many splits?

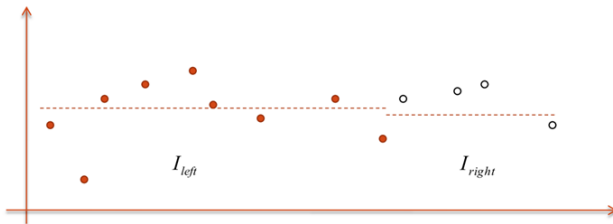
- ▶ Optimal constant function is the average of the outcome in the interval.
- ▶ Only important if observations are in the interval or not.
- ▶ Split somewhere in each gap between observations.



How many splits?

In an interval with n_I observations we have n_I possible splits

- ▶ Nothing fancy here, just try them all
- ▶ Chose the one with the lowest total RSS on the interval



$$RSS_I = RSS_{I_{left}} + RSS_{I_{right}}$$

Splitting categorical predictors

- ▶ Consider a two-category input (e.g. male-female)
 - ▶ Only one way to split, men versus women
 - ▶ Empty groups are not allowed
 - ▶ {male},{female} and {female},{male} is the same split
- ▶ Consider a three-category input {apple, orange, banana}
 - ▶ Three possible splits
 - {apple} , {orange, banana}
 - {apple, orange} , {banana}
 - {apple, banana} , {orange}
- ▶ How many splits for a variable with k categories?
 - ▶ In today's exercises!

How large do we grow the tree?

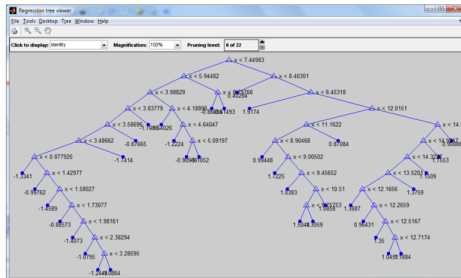
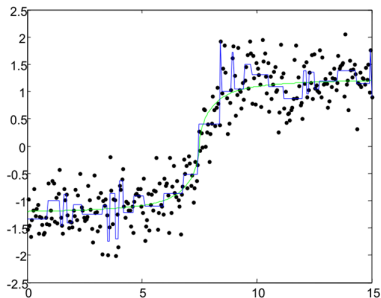
- ▶ Stop splitting when a node contains too few observations
- ▶ For instance, do not split nodes with 10 or less observations
- ▶ This variable is called `MinParent` in Matlab's `fitrtree` function

Tree-growing procedure

- ▶ First interval (node) is the entire range of X .
- ▶ For each variable
 - ▶ For each splitting position, calculate $RSS = RSS_{left} + RSS_{right}$
 - ▶ Remember position with the lowest RSS
- ▶ Split the variable with the lowest RSS at the corresponding position
- ▶ This produces a left and right sub-interval (child-nodes)
 - ▶ Split each of these into child nodes as above
 - ▶ Keep splitting nodes until a node contains too few observations
- ▶ Assign a constant function to terminal nodes, the average of the observations

Resulting tree

MinParent = 10

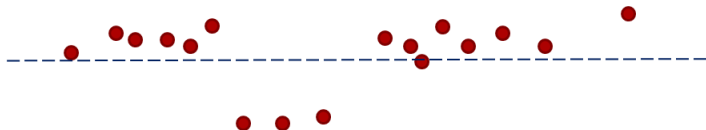


Finding the right sized tree

- ▶ The tree was split too far - overfitting
- ▶ How do we know when to stop splitting?

Answer: we don't

- ▶ A seemingly worthless split may lead to excellent splits below

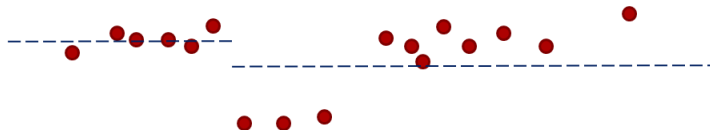


Finding the right sized tree

- ▶ The tree was split too far - overfitting
- ▶ How do we know when to stop splitting?

Answer: we don't

- ▶ A seemingly worthless split may lead to excellent splits below

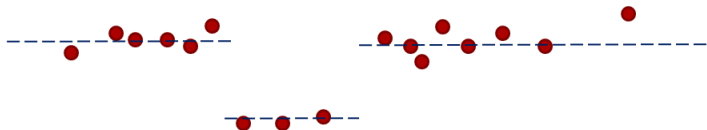


Finding the right sized tree

- ▶ The tree was split too far - overfitting
- ▶ How do we know when to stop splitting?

Answer: we don't

- ▶ A seemingly worthless split may lead to excellent splits below



Finding the right sized tree

- ▶ We don't want to miss out on good splits
- ▶ Strategy: grow the tree really large (small `MinParent`) and then decide which splits were unnecessary and remove these.
- ▶ This is called **pruning** the tree
 - ▶ Pruning a node amounts to removing its sub-tree, thereby making a terminal node



Pruning rule

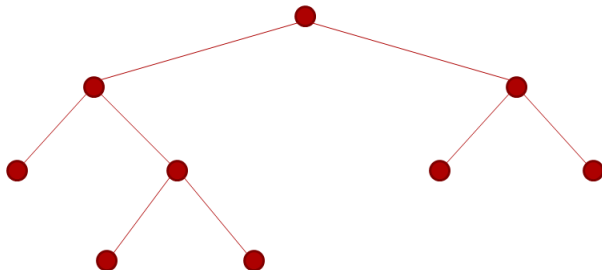
Prune the non-terminal node whose sub-tree gives the smallest **per node** reduction in RSS.

- ▶ Divide the reduction by the number of terminal nodes minus one

Pruning the tree

Weakest-link pruning: prune branches that contribute the least to lowering RSS

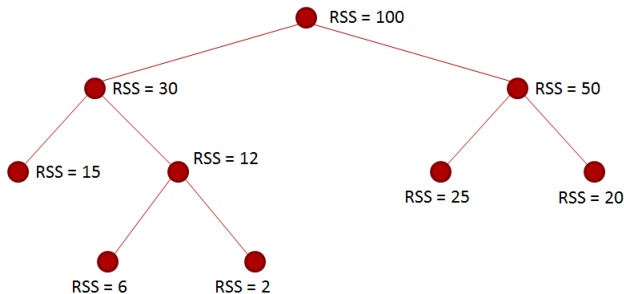
Example:



Pruning the tree

Weakest-link pruning: prune branches that contribute the least to lowering RSS

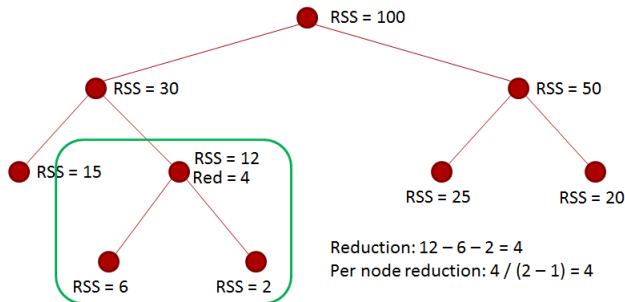
Example:



Pruning the tree

Weakest-link pruning: prune branches that contribute the least to lowering RSS

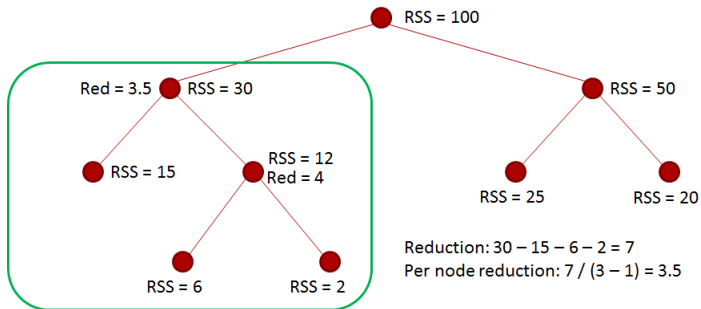
Example:



Pruning the tree

Weakest-link pruning: prune branches that contribute the least to lowering RSS

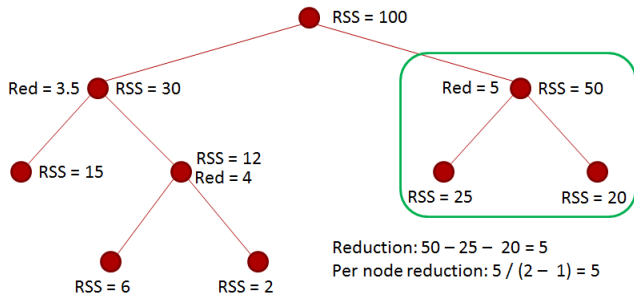
Example:



Pruning the tree

Weakest-link pruning: prune branches that contribute the least to lowering RSS

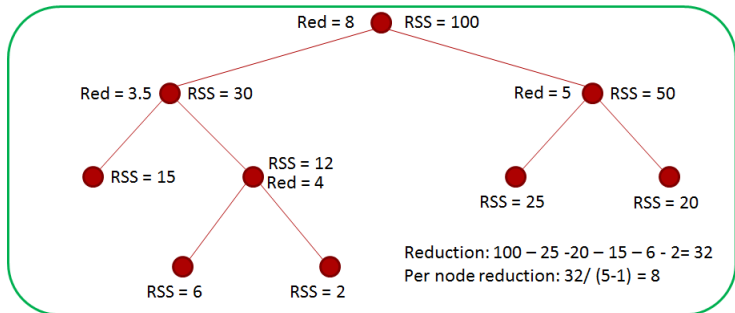
Example:



Pruning the tree

Weakest-link pruning: prune branches that contribute the least to lowering RSS

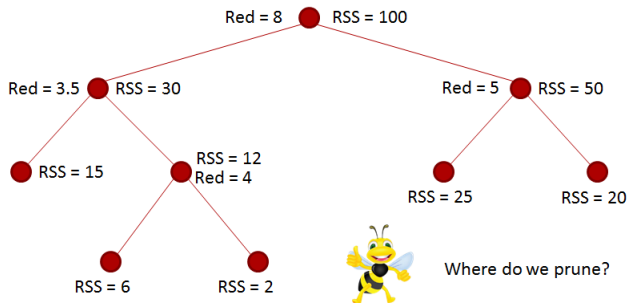
Example:



Pruning the tree

Weakest-link pruning: prune branches that contribute the least to lowering RSS

Example:

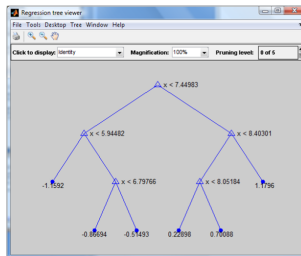
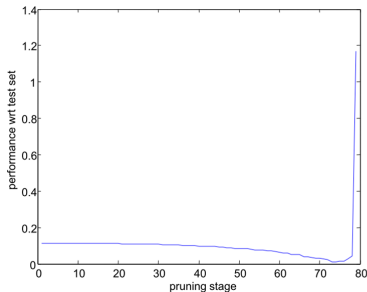


When to stop pruning?

- ▶ If we keep pruning, we will end up with just the root node
- ▶ Which of all these sub-trees do we choose?
- ▶ Two approaches
 - ▶ Independent test set
 - ▶ Cross-validation

Using an independent test set

- ▶ As we prune our way towards the root node, evaluate the performance of each sub-tree with respect to the test set.
- ▶ Continue all the way to the root node
 - ▶ Choose sub-tree with best performance



Using cross-validation

Three alternatives...

- ▶ Cross validate **all possible trees**, does not work in practice - too many trees.
- ▶ Use a **tuning parameter**. Choose the tree that minimizes,

$$RSS(T) + \alpha |T|$$

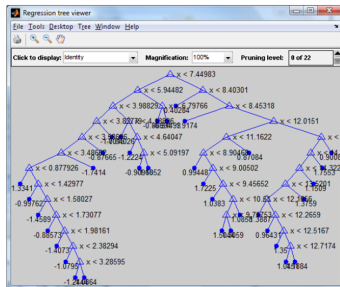
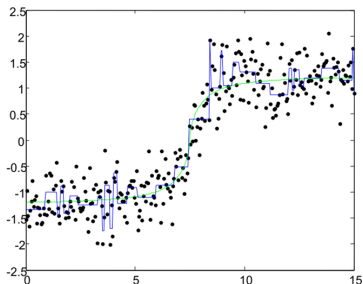
$|T| = \# \text{ end-nodes}$

Find the tuning parameter α using cross validation

- ▶ Use the cross validation to determine the optimal value of the minimum number of **observations in a terminal node**. Matlab parameter 'MinLeaf'. Then use the full grown tree without pruning.

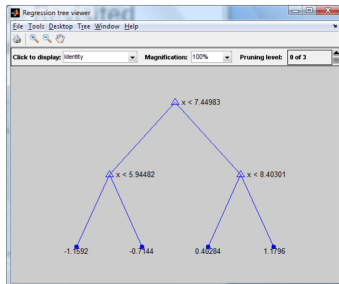
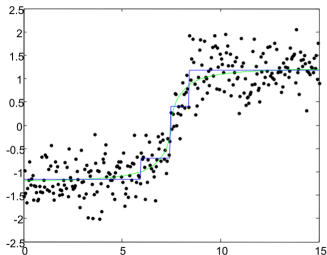
Regression example revisited

MinParent = 20 full tree

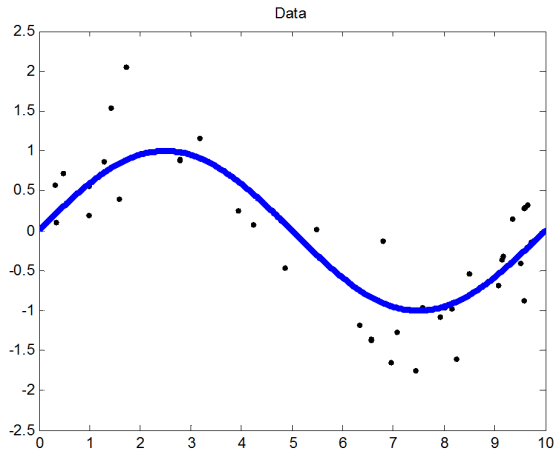


Regression example revisited

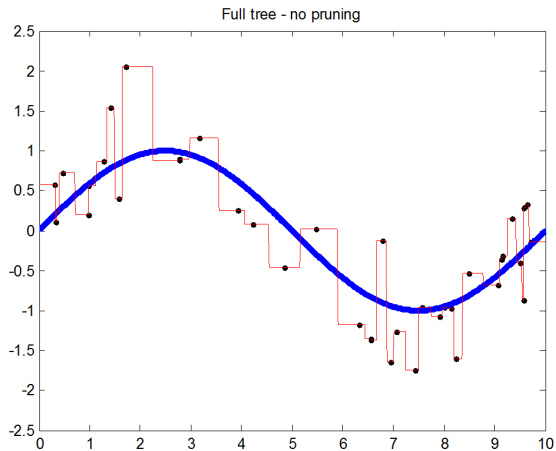
Best sub-tree chosen by 10-fold cross-validation.



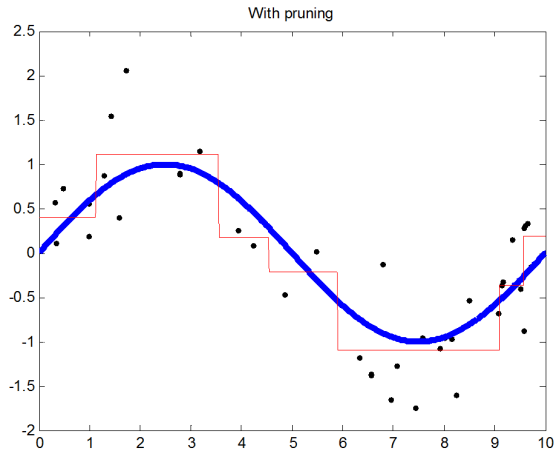
Bias and variance trade-off



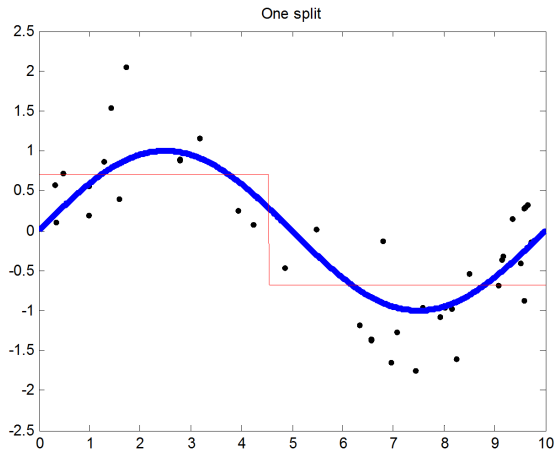
Bias and variance trade-off



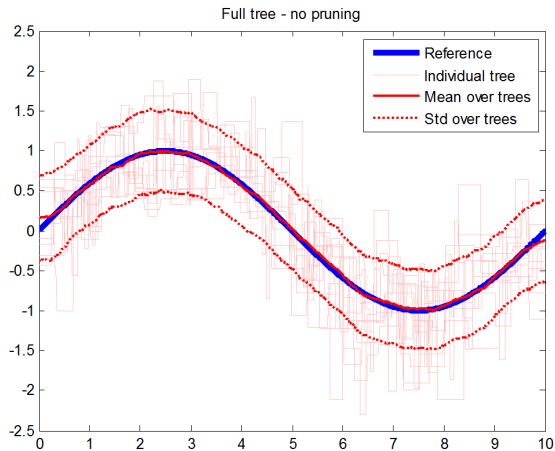
Bias and variance trade-off



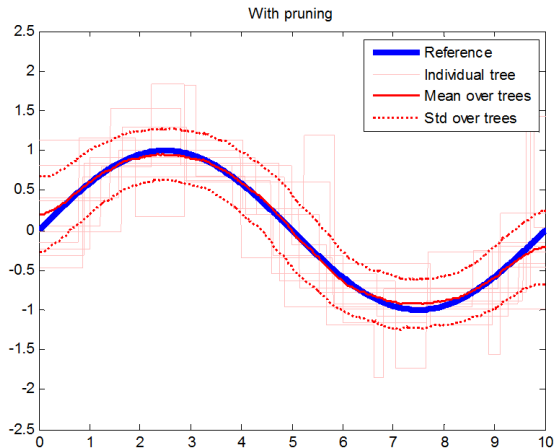
Bias and variance trade-off



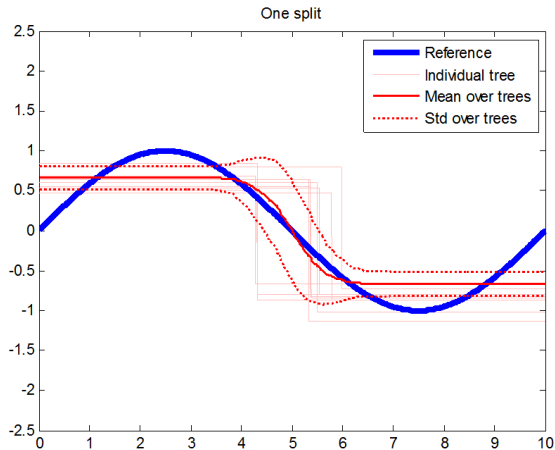
Bias and variance trade-off



Bias and variance trade-off



Bias and variance trade-off



Bias and variance trade-off

What do we conclude about regression trees in terms of

- ▶ Bias
- ▶ Variance



Classification trees

• A classification tree is a model that predicts the class of an object based on a set of features.

• The tree is built by recursively splitting the data into subsets based on the values of the features.

• The splits are chosen to maximize the homogeneity of the resulting subsets.

• The tree is then used to classify new objects by following the path of the splits.

• Classification trees are a popular method for classification tasks.

• They are easy to interpret and can handle both numerical and categorical features.

• They are also robust to outliers and missing data.

• Classification trees are a good choice for many classification tasks.

• They are a good choice for tasks where interpretability is important.

• They are a good choice for tasks where the data is not linearly separable.

• Classification trees are a good choice for tasks where the data is noisy.

• They are a good choice for tasks where the data is imbalanced.

• Classification trees are a good choice for tasks where the data is high-dimensional.

• They are a good choice for tasks where the data is sparse.

• Classification trees are a good choice for tasks where the data is complex.

• They are a good choice for tasks where the data is noisy.

• They are a good choice for tasks where the data is imbalanced.

• Classification trees are a good choice for tasks where the data is high-dimensional.

• They are a good choice for tasks where the data is sparse.

• Classification trees are a good choice for tasks where the data is complex.

• They are a good choice for tasks where the data is noisy.

• They are a good choice for tasks where the data is imbalanced.

• Classification trees are a good choice for tasks where the data is high-dimensional.

• They are a good choice for tasks where the data is sparse.

• Classification trees are a good choice for tasks where the data is complex.

• They are a good choice for tasks where the data is noisy.

• They are a good choice for tasks where the data is imbalanced.

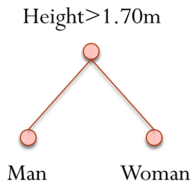
• Classification trees are a good choice for tasks where the data is high-dimensional.

• They are a good choice for tasks where the data is sparse.

• Classification trees are a good choice for tasks where the data is complex.

Classification trees

Terminal node assigns a class instead of a constant



Only difference: criteria for splitting nodes and pruning the tree

Node values

With **regression trees** we transform a new observation into a constant.

- ▶ The constant was derived from the training data.
- ▶ It was the **mean** of the output variable of the training observations in the node.

For **classification trees** we assign new observations to a certain class

- ▶ The class is derived from training data
- ▶ It is the **majority class** of the training observations in the node

Model error

Regression trees

- ▶ For regression trees we used RSS as a measure of node impurity

Classification trees

- ▶ For classification trees we have a few options
 - ▶ Missclassification rate
 - ▶ Gini index
 - ▶ Cross-entropy
- ▶ They all favor the split that increases purity the most.
 - ▶ Typically the predictive performance is not that different
 - ▶ The shape of the trees might, however, be very different

Node impurity for classification trees

In a specific node, representing a region R with N observations, let

$$\hat{p}_k = \frac{1}{N} \sum_{x_i \in R} \mathbb{1}(y_i = k)$$

Classify observations in the node to class,

$$K = \arg \max_k \hat{p}_k$$

Measures of impurity within a node,

Misclassification error: $Q = \frac{1}{N} \sum_{i \in R} \mathbb{1}(y_i \neq K) = 1 - \hat{p}_K$

Gini index: $Q = \sum_{k \neq k'} \hat{p}_k \hat{p}_{k'} = \sum_k \hat{p}_k (1 - \hat{p}_k)$

Cross-entropy (deviance): $Q = - \sum_k \hat{p} \log \hat{p}_k$

From node impurity to split criterion

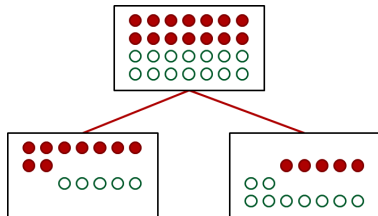
- ▶ The node impurity is weighted with the number of observations in each node.
- ▶ The split decision is based on the split that **minimizes**,

$$N_{left} Q_{left} + N_{right} Q_{right}$$

N , the number of observations in left and right node

Q , node impurity for left and right node

Comparing splits



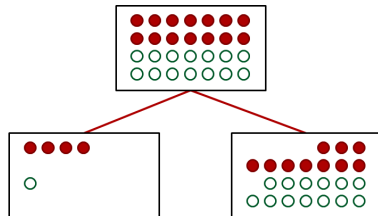
Missclassification error

Left node: 5/14

Right node: 5/14

Split criterion

$$14(5/14) + 14(5/14) = 10$$



Missclassification error

Left node: 1/5

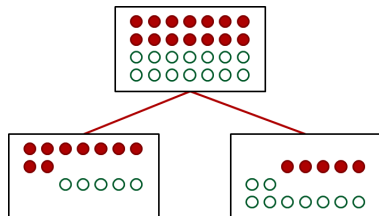
Right node: 10/23

Split criterion

$$5(1/5) + 23(10/23) = 11$$

Lower is better!

Comparing splits



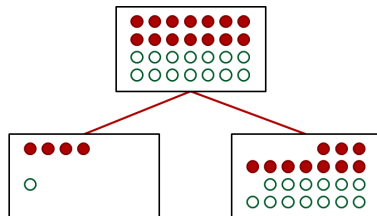
Gini index

Left node: ...

Right node: ...

Split criterion

...



Gini index

Left node: ...

Right node: ...

Split criterion

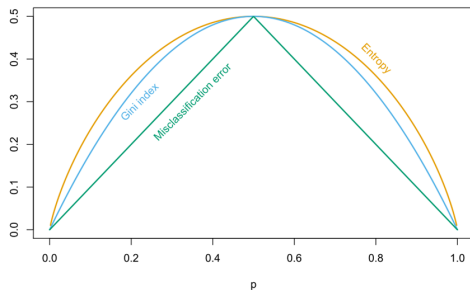
...



Calculate split criterion based on the Gini index!

Node impurity measures for classification trees

- ▶ Node impurity measures for a two-class problem.
- ▶ X-axis: proportion of samples belonging to class 2.
- ▶ Entropy and Gini index are better measures for growing tree because they are more sensitive to node probabilities.



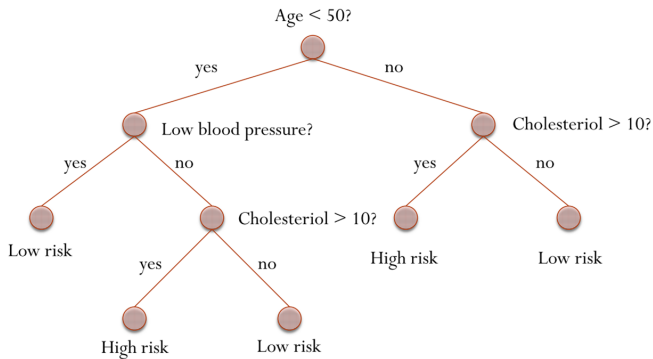
Standard practice

- ▶ Use **Gini index** as split criterion when **building** the tree.
- ▶ Use **missclassification rate** as criterion when deciding which node to **prune**.

Benefits of a tree structure

Interpretability!

Consider the following (classification) tree on heart diseases



Interpretability

- ▶ CART is popular in medical sciences because it may represent the way doctors reason.
- ▶ A single tree describes the entire partitioning on the input space.
- ▶ With $p > 3$ input variables, the partition (cf. knot positions) are difficult to visualize.
 - ▶ But a tree representation is always possible.
- ▶ A large tree might be difficult to interpret anyway...

Missing data

Incomplete data are common in many applications.

We can always

- ▶ Delete the observation
- ▶ Replace with mean or median

For trees we can

- ▶ Introduce and **extra category** “missing” - if it is a categorical variable.
- ▶ Use a **surrogate variable**. In each branch, have a list of alternative variables and split points - as a backup.
 - ▶ Matlab does this.

Questions?

