# Unit 07: Pomona Sprout and Random Forests

## Applied AI with R

Ferdinand Ferber and Wolfgang Trutschnig

Paris Lodron Universität Salzburg

5/1/24

# Table of contents I

# Pomona Sprout and Random Forests

AI generated image for the prompt “Pomona Sprout with a wizzard hat watering a tree growing gears in the yard of Hogwards.”

Section 1

# Decision Trees

# Decision Trees

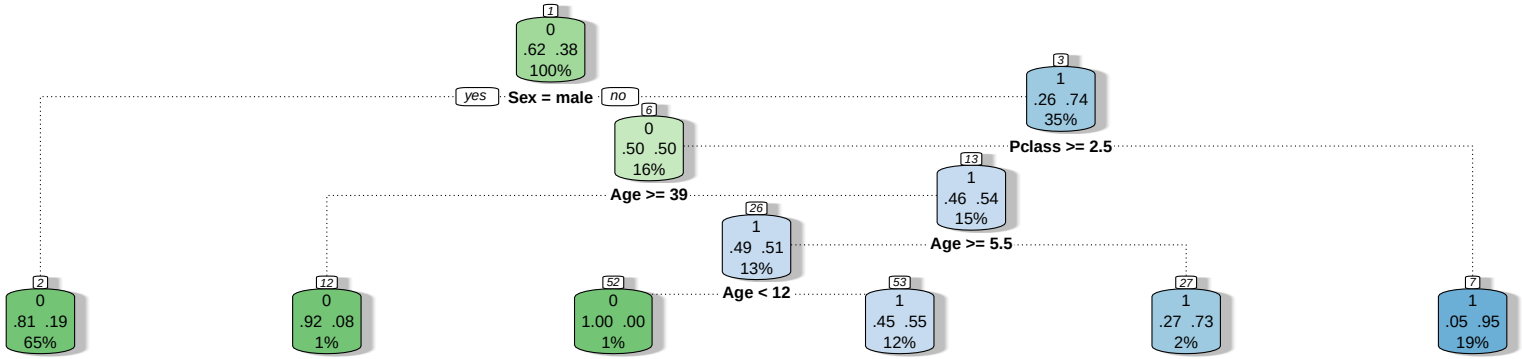
## Decision Tree

A *decision tree* is a tree (an undirected, acyclic, connected, rooted graph) with the following properties:

- every inner node is labeled by some predictor variable (feature)
- every outgoing edge is labeled by some condition on the corresponding variable ("split condition")
- every leaf is either labeled by a single class or a probability distribution over the classes.
- For every inner node the predicates of its outgoing edges cover ALL possible cases without contradiction or overlap.

# Decision Trees

- Example of a decision tree trained on the `titanic` dataset
- The numbers inside each node mean: Predicted class (top), class probabilities (middle) and size of the current split (bottom)




- What would this model predict for a 50 years old, female first-class passenger? Class 1 means `survived`.

# Why Decision Trees?

- Perform surprisingly well on most tabular data tasks
- Easy to interpret
- Quite robust
  - Can handle categorical data without preprocessing
  - Can handle missing data directly
  - No need for data normalization
  - Insensitive to outliers
- Scale well with large datasets

# Inference on Decision Trees

- Start at the root node
- For every inner node:
  - Test the predicates of its outgoing edges
  - Follow the edge that evaluated to `true`
  - Proceed recursively
- If a leaf node (terminal node) is reached, return the corresponding class (or the class with the highest confidence)

# Idea/principle underlying Decision Trees:

- Main approach: Greedy algorithm with recursive partitioning
  - Select the test attribute/feature that maximizes predictive power (homogeneity on the nodes)
  - Generate the predicates
  - Divide the training set according to the predicates
  - Proceed recursively...
  - Terminate the recursion if all cases belong to the same class or if some other termination criterion applies
- This greedy approach only finds a local optimum, finding the best decision tree is NP-hard, i.e., infeasable.

# What kind of predicates are allowed?

- Usually we only allow specific kinds of predicates in the decision tree depending on their type:
- @factorial attributes:
  - Tests of equality for each category
  - E.g.: if predictor variable $X$ has possible values $\text{im}(X) = \{A, B, C\}$, then the considered predicates are `X == A`, `X == B` and `X == C`
- @continuous attributes:
  - Tests of inequality
  - E.g.: if predictor variable $X$ is real-valued and we have $a_1 < a_2 < \cdots < a_k$, then the considered tests are $X < a_1, a_1 \leq X < a_2, a_2 \leq X < a_3, \ldots, a_k \leq X$

# Roadmap

- We start with the case where all predictor variables (in addition to the outcome variable) are discrete
- In the next section we will extend quickly to continuous predictor variables...
- ...and to continuous outcomes (called *regression trees*)

# How to select a test attribute/how to split?

- In the next couple of slides we will learn about the *entropy* as a measure quantifying uncertainty of a random variable $X$
- Entropy is one of the standard splitting options; another commonly used one is the *Gini impurity*
- We are interested in the entropy of the outcome/class variable, i.e. how uncertain is the class for a given subset of the data
- Among all possible splits of the data we can compute the difference between the current entropy of the class variable and the entropy of the class variable after the split, called *information gain*
- We want to choose the variable that maximizes the expected information gain for all its possible splits

# Entropy

- Let $X$ be a discrete random variable with finite range/image $\text{im}(X) = A$ (i.e., $\#A < \infty$)
- Consider the following task:
  - You need to guess the output of $X$
  - You know the outcome probabilities and you can ask an oracle if the output of $X$ is contained in some set $B \subseteq A$ or not
  - How many yes/no-questions do you need - on average - until you found the output?

# Entropy

- Consider r.v. $X$ having the following probabilities:
- $\mathbb{P}(X = a_1) = 0.4, \mathbb{P}(X = a_2) = 0.3, \mathbb{P}(X = a_3) = 0.2, \mathbb{P}(X = a_4) = 0.1.$
- What's the optimal asking strategy in this setting?
- Suppose we decide to ask according to decreasing probabilities (why?)
- The symbol $Q$ will denote the number of required questions
- Then we have $\mathbb{P}(Q = 1) = 0.4, \mathbb{P}(Q = 2) = 0.3, \mathbb{P}(Q = 3) = 0.2, \mathbb{P}(Q = 4) = 0.1$
- Hence $\mathbb{E}(Q) = \sum_{i=1}^{4} \mathbb{P}(Q = i) \cdot i = ... = 2$
- I.e., on averge we need $2$ questions.

# Exercise

- Consider r.v. $X$ having the following probabilities:
- $\mathbb{P}(X = a_1) = \mathbb{P}(X = a_2) = \mathbb{P}(X = a_3) = \mathbb{P}(X = a_4) = 0.25$.
- Check that our strategy from before yields $\mathbb{E}(Q) = 2.5$
- Is it possible to find an alternative strategy with the same expected number of questions, which, however, at most needs 3 questions?

# Entropy

- Naive way (but not entirely stupid)
  - Sort the values of $A$ according to their probabilities
  - $A = \{a_1, \dots, a_n\}$, where $\mathbb{P}[a_i] \geq \mathbb{P}[a_{i+1}]$
  - Ask the singletons $\{a_1\}, \{a_2\}, \dots, \{a_n\}$ to the oracle
- Alternative way (Shannon-Fano Coding)
  - Recursively arrange the possible values into two sets $B_1$ and $B_2$ with $\mathbb{P}[B_1] \approx \mathbb{P}[B_2]$
  - Ask $B_1$ to the oracle
  - If the oracle answers that the outcome is in $B_1$, then apply this scheme recursively to $B_1$. Otherwise, apply it to $B_2$
  - The recursion terminates if $\#B_1 = \#B_2 = 1$
- Optimal way: Huffman Coding

# Entropy

- One can prove that a lower bound on the expected number of yes/no-questions (of the above type) to the oracle needed is given by the *entropy* $H(X)$ of $X$

**Entropy**

Let $X$ be a discrete random variable, then its *entropy* $H(X)$ is defined as

$$H(X) := -\sum_{x \in \mathsf{im}(X)} \mathbb{P}[X = x] \log_2 \left(\mathbb{P}[X = x]\right)$$

with the convention $0 \cdot \log(0) := 0$

- Entropy measures the extent of uncertainty of a random outcome ($log_2$ comes from coding and bits)

# Entropy

- Sketch of the proof: Fix any questioning scheme; let $\lambda_i$ be the number of yes/no questions required to identify element $a_i$.
- Then the average number of yes/no questions to identify the outcome of $X$ is $\overline{\lambda} := \sum_{i=1}^{n} \lambda_i \mathbb{P}[X = a_i]$. Consider

$$\begin{aligned} H(X) - \overline{\lambda} &= \sum_{i=1}^{n} \left( \log_2 \left( \frac{1}{\mathbb{P}[X = a_i]} \right) - \lambda_i \right) \mathbb{P}[X = a_i] \\ &= \sum_{i=1}^{n} \log_2 \left( \frac{2^{-\lambda_i}}{\mathbb{P}[X = a_i]} \right) \mathbb{P}[X = a_i] \end{aligned}$$

# Entropy

- Applying Jensen's inequality to the previous equation yields

$$
\begin{aligned}
H(X) - \overline{\lambda} &\overset{\text{Jensen's ineq.}}{\leq} \log_2 \left( \sum_{i=1}^{n} \frac{2^{-\lambda_i}}{\mathbb{P}[X = a_i]} \mathbb{P}[X = a_i] \right) \\
&= \log_2 \left( \sum_{i=1}^{n} 2^{-\lambda_i} \right) \\
&\overset{\text{Kraft ineq.}}{\leq} \log_2(1) \\
&= 0
\end{aligned}
$$

- Hence, $H(X) - \overline{\lambda} \leq 0$, i.e. $H(X) \leq \overline{\lambda}$ for any questioning scheme.

# Joint Entropy

- We now look at the case where we have two discrete random variables $X$ and $Y$
- Up to this point we are able to measure the amount of uncerainty $H(X)$ in $X$ and $H(Y)$ in $Y$
- It is easy to extend this to the random variable $(X, Y)$
- Set $\mathcal{X} := \mathrm{im}(X)$ and $\mathcal{Y} := \mathrm{im}(Y)$
- Then the *joint entropy* $H(X, Y)$ is defined as:

$$H(X,Y) := -\sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} \mathbb{P}[X=x, Y=y] \log_2 \left(\mathbb{P}[X=x, Y=y]\right)$$

# Conditional Entropy

- Interesting question: Assume that we already know the outcome $Y = y$, what is the remaining entropy of $(X, Y)$?
- If $Y = y$ provides knowledge about $X$ then $H(X|Y = y)$ should be lower than $H(X)$
- We calculate the entropy for the conditioned random variable $X|Y = y$
- Plugging it into the definition yields:

$$H(X|Y = y) := -\sum_{x \in \mathcal{X}} \mathbb{P}[X = x|Y = y] \log_2 \left(\mathbb{P}[X = x|Y = y]\right)$$

# Information Gain

- As stated before, we are interested in the entropy of the outcome / class variable, i.e. the amount of uncertainty of the class
- The *information gain* is a metric that quantifies how much uncertainty of the class variable we lose (= how much info we gain) when knowing the value of some given predictor variable

# Information Gain

- For two discrete random variables $X$ and $Y$, the information gain $I(X, Y = y)$ should quantify how much uncertainty we lose in $X$ if we knew $Y = y$
- After learning about entropy and conditional entropy, it makes sense to define the information gain as

**Information Gain**

Let $X$ and $Y$ be two discrete random variables, and $y \in \mathcal{Y}$, then the *information gain* $I(X, Y = y)$ is defined as

$$I(X, Y = y) := H(X) - H(X|Y = y)$$

# Expected Information Gain

- If the decision tree were to be a binary tree, then we could just compute the information gain of the outcome / class variable for every possible equality test on any predictor variable and directly use the data split with the highest information gain
- But usually we are allowed to have multiple outgoing edges, i.e. we test all possible values of a predictor variable at the same time
- Hence, we calculate the *expected information gain* for every predictor variable to decide on the next node:

$$I(X, Y) := H(X) - \mathbb{E}\left[H(X|Y)\right]$$

# Expected Information Gain

The second term of $I(X, Y)$ is called *conditional entropy* and can be calculated as:

$$
\begin{aligned}
H(X|Y) :&= \sum_{y \in \mathcal{Y}} \mathbb{P}[Y = y] H(X|Y = y) \\
&= \sum_{y \in \mathcal{Y}} \mathbb{P}[Y = y] \sum_{x \in \mathcal{X}} \mathbb{P}[X = x|Y = y] \log_2 \left(\mathbb{P}[X = x|Y = y]\right) \\
&= \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} \mathbb{P}[Y = y] \mathbb{P}[X = x|Y = y] \log_2 \left(\mathbb{P}[X = x|Y = y]\right) \\
&= \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} \mathbb{P}[X = x, Y = y] \log_2 \left(\mathbb{P}[X = x|Y = y]\right)
\end{aligned}
$$

# Recap

- Let $Y$ be the outcome / class variable and $X_1, \ldots, X_n$ the discrete predictor variables (a.k.a. features)
- In every step of the decision tree induction algorithm we calculate the expected information gains $I(Y, X_1), \ldots, I(Y, X_n)$ based on the remaining data
- Take the $X_i$ with the highest expected information gain and insert a new inner node in the decision tree
- The outgoing edges are labeled with `X_i == a_1`, ..., `X_i == a_k` for $\{a_1, \ldots, a_k\} = \text{im}(X_i)$
- The current subset of the data is further divided into $k$ splits, based on the values at $X_i$
- The algorithm is applied recursively to every data split (till some stopping criterium is met)

Section 2

# Continuous Case

# Continuous predictor variables

- So far we only defined the entropy (and by extension also the information gain) for discrete predictor variables
- The general idea for handling a continuous predictor variable $X$ is to discretize $X$ by considering all possible splits of the form $a \leq X$, for $a$ in the sampled image of $X$
- Then the continuous predictor variable $X$ is replaced by all those binary predictor variables $X_{\leq a_1}, \ldots, X_{\leq a_r}$ and the expected information gain can be computed as before
- Think of our titanic example and the resulting decision tree

# Exercise

- Reconsider the titanic dataset in the Code Snippet 07_Sprout.R
- Change the method from 'information' to Gini impurity
- Are the obtained results similar?

# Regression trees

- If the outcome variable is continuous, then the decision tree is called a *regression tree*
- Usually, the prediction of a regression tree is the average outcome of all training data points that landed in the corresponding leaf node
- The standard idea is to minimize variability (variance) in the nodes, i.e., we split the data in such a way that the resulting variances in the individual nodes is minimal

Section 3

# Random Forests

# Random Forests

- Random forests consist (as the name suggests) of many decision trees which are combined for producing a prediction
- Decision trees naturally suffer from overfitting
- Random Forests overcome this problem, they exhibit less overfitting and better performance
- There are two main ingredients in random forests
  - Bagging
  - Random subspace method

# Bagging

- *Bagging* stands for *bootstrap aggregating*
- Let $D$ be the training set with $n := \#D$ data points
- Out of $D$ draw several random samples $D_1, \dots, D_r$ (with replacement) of size $\#D_i =: m < n$ and train a decision tree for each of those new datasets.
- This results in $r$ models $f_1, \dots, f_r$
- At inference time take the majority class of all predictions in the classification setting
- ...and the mean of the forecasts in the regression setting

# Random Subspace Method

- Sometimes called *feature bagging*
- At variable selection in the training process consider only a random subset of the predictor variables for possible splits
- Typically, if there are $p$ predictor variables, only consider $\lfloor \sqrt{p} \rfloor$ variables for splitting

# Exercise

- We run a Random Forest on the titanic dataset
- Use the R-Code in Snippet 07_Sprout.R starting at line 60
- Compare the obtained results with the ones obtained by the prior classification tree - are the obtained results similar?

Section 4

# Trees and Extrapolation

# Extrapolation

- Consider the following prediction tasks
  - Predicting future rainfall in flood or drought regions
  - Forecasting peak energy consumption
  - Estimating heart rate changes for vulnerable patients
- In all these cases, we are modeling extreme cases. Predicting the average rainfall or heart rate changes for healthy patients is uninteresting here.
- Often, the training dataset only contains very few data points for these extreme cases
- Therefore, models need to *extrapolate* well

# Extrapolation

- Let's test the extrapolation capabilities of linear models and decision trees:

```
train <- nutrition |>
  filter(calories < quantile(calories, 0.75, na.rm = T))

test <- nutrition |>
  filter(calories >= quantile(calories, 0.75, na.rm = T))
```

# Extrapolation

- Train both models on the train dataset:

```
fitted_lm <- linear_reg() |>
  fit(calories ~ proteins + fat + carbohydrate,
      data = train)

fitted_dt <- decision_tree() |>
  set_mode("regression") |>
  fit(calories ~ proteins + fat + carbohydrate,
      data = train)
```

# Extrapolation

- Test it on both, the train and the test dataset:

```
# A tibble: 2 x 3
  model         rmse_train rmse_test
  <chr>              <dbl>     <dbl>
1 lm                  30.5      147.
2 decision tree       34.1      203.
```

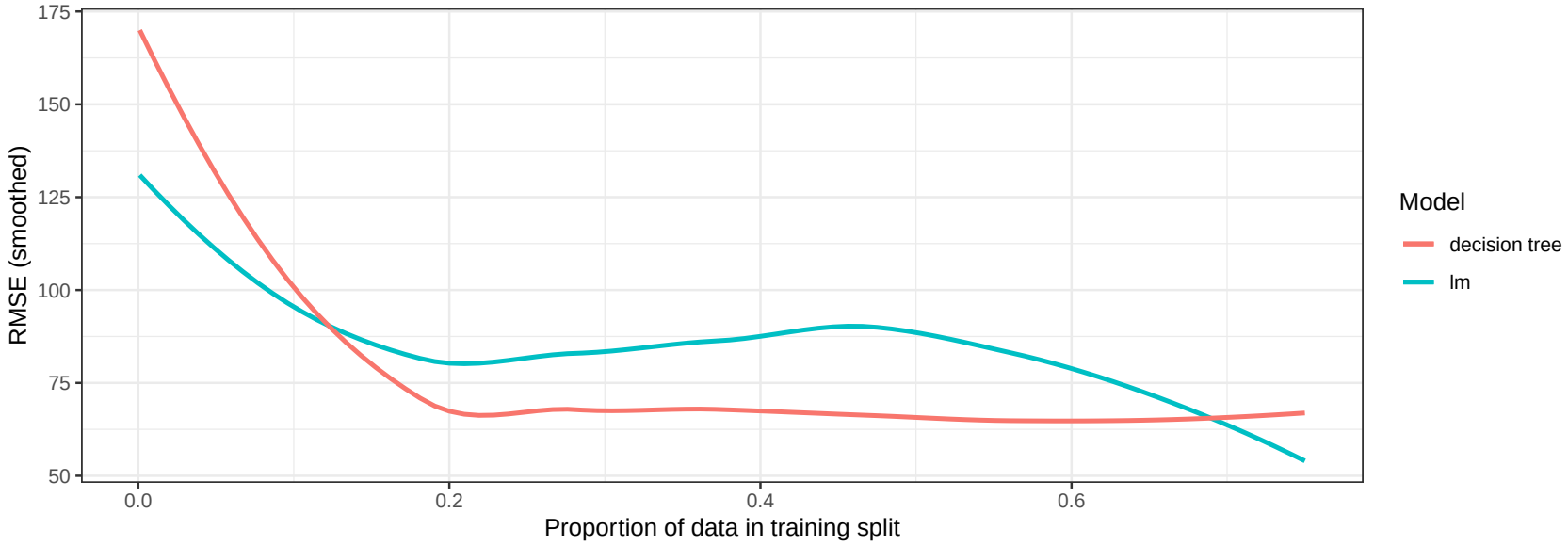
- As we can see, the linear model performs considerable better in the extrapolation regime than the decision tree, even though on the training dataset they had similar losses.

# Data hungryness

- On real-world data, regression trees have been observed to require up to 10 times as many data points to achieve the same validation/test loss compared to linear models - in a linear world
- This is especially relevant for situations where data is scarce, e.g. clinical datasets

# Data hungryness

- There we use the `nutrition` dataset again and plot the validation loss as a function of training dataset size (in percent of all available data)
- We see that the linear model outperforms the regression tree for small training sets



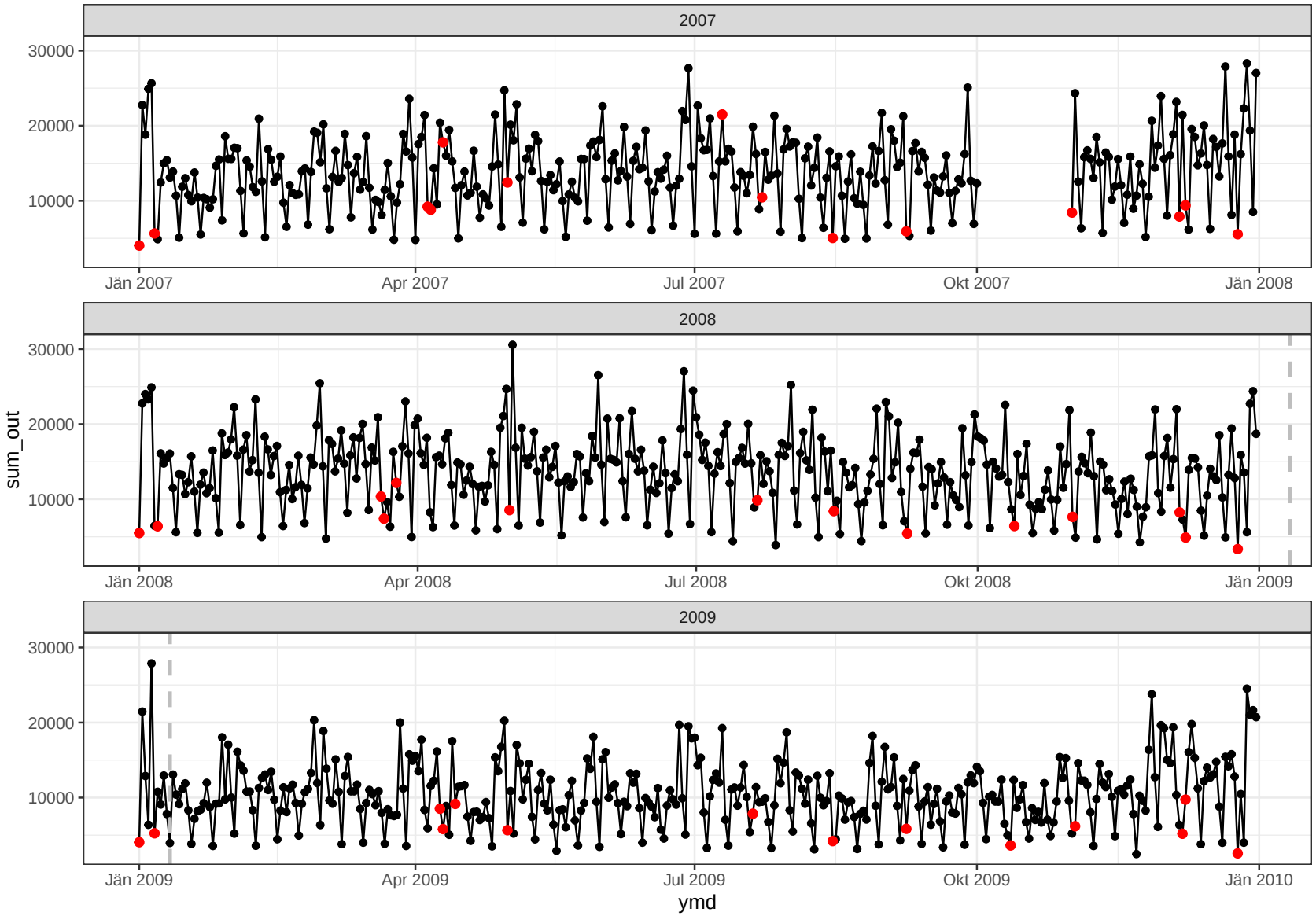
Section 5

# Forecasting with Trees

# Forecasting with Trees and Random Forests

- Sometimes trees and RFs are used as baseline for timeseries forecasting
- We return to the ATM dataset and try to forecast the daily withdrawn amounts
- Problem description ($d = 1$):
  - Use the data from 2008-01-01 till 2009-01-11 to train a model for predicing $d = 1$ day ahead.
  - Predict the withdrawn amount on 2009-01-12
  - Use the data from 2008-01-01 till 2009-01-12 to train a model or use the model from the first step
  - Predict the withdrawn amount on 2009-01-13
  - ...proceed analogously till 2009-12-12
  - Compare the forecasts with the true withdrawn amounts to check the performance of the model

# ATM Data


2007
2008
2009
sum_out
30000
20000
10000
Jän 2007
Apr 2007
Jul 2007
Okt 2007
Jän 2008
Jän 2008
Apr 2008
Jul 2008
Okt 2008
Jän 2009
Jän 2009
Apr 2009
Jul 2009
Okt 2009
Jän 2010
ymd

# Forecasting with Trees and Random Forests

- Problem description ($d = 7$):
  - Use the data from 2008-01-01 till 2009-01-11 to train a model for predicing $d = 7$ days ahead.
  - Predict the withdrawn amount on 2009-01-18
  - Use the data from 2008-01-01 till 2009-01-12 to train a model or use the model from the first step
  - Predict the withdrawn amount on 2009-01-19
  - ...proceed analogously till 2009-12-12
  - Compare the forecasts with the true withdrawn amounts to check the performance of the model
- Is the $d = 7$ much worse than the $d = 1$ model?
- Before you start hand-on, think through what features can/should be used for the forecast!