Prediction

Machine Learning with R

Basel R Bootcamp

May 2019

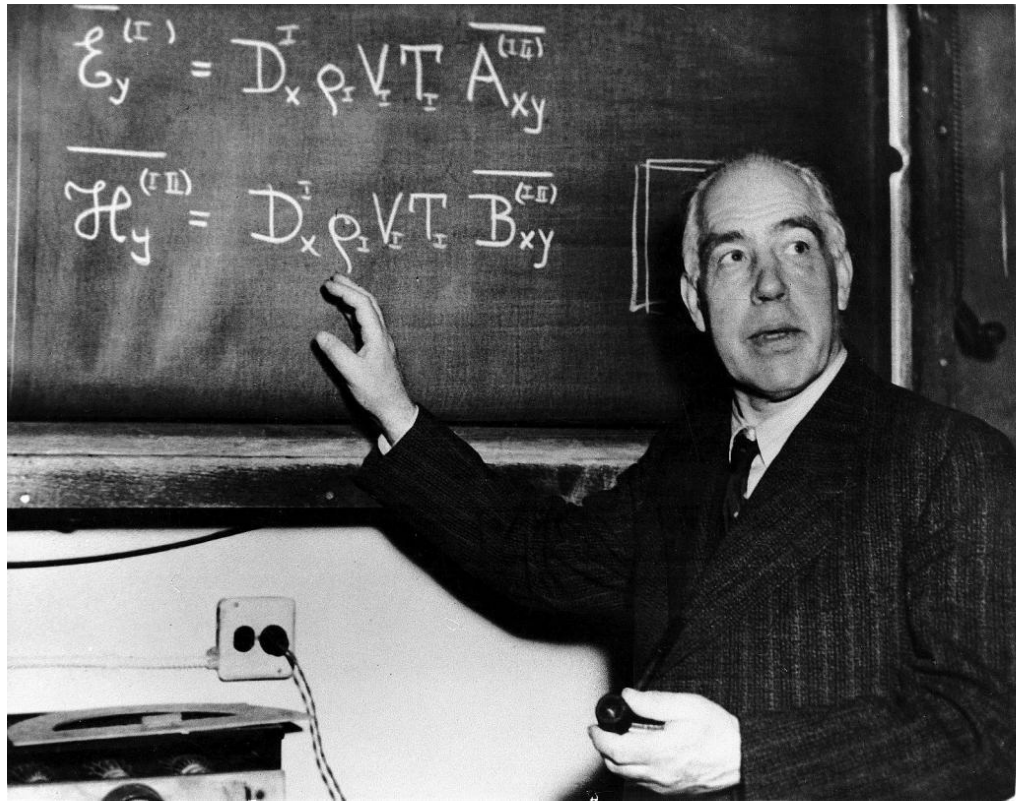
Prediction is...

Prediction is very difficult, especially if it's about the future.

Nils Bohr, Nobel Laureate in Physics

An economist is an expert who will know tomorrow why the things he predicted yesterday didn't happen today.

Evan Esar, Humorist



from futurism.com

Hold-out data

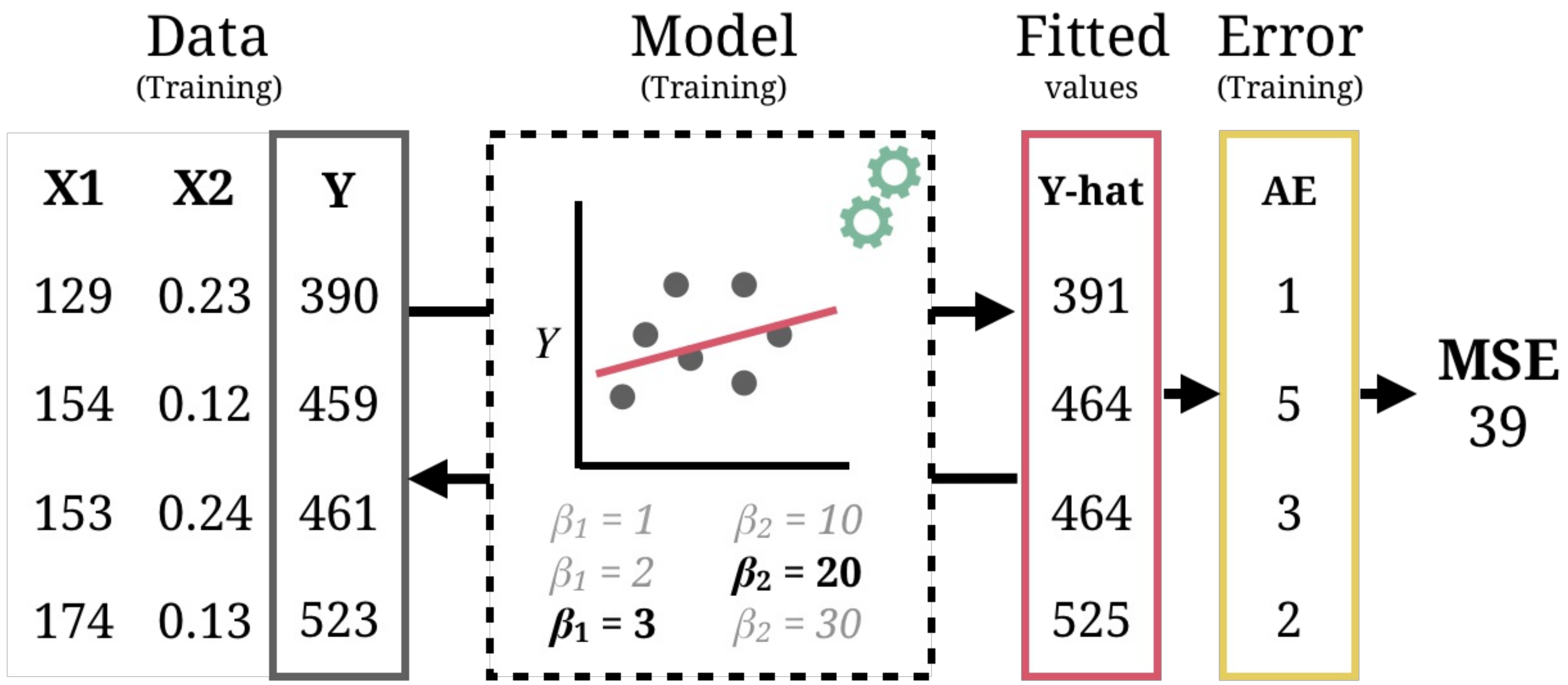
Model performance must be evaluated as true prediction on an **unseen data set**.

The unseen data set can be **naturally** occurring, e.g., using 2019 stock prizes to evaluate a model fit using 2018 stock prizes.

More commonly unseen data is created by splitting the available data into a training set and a test set.

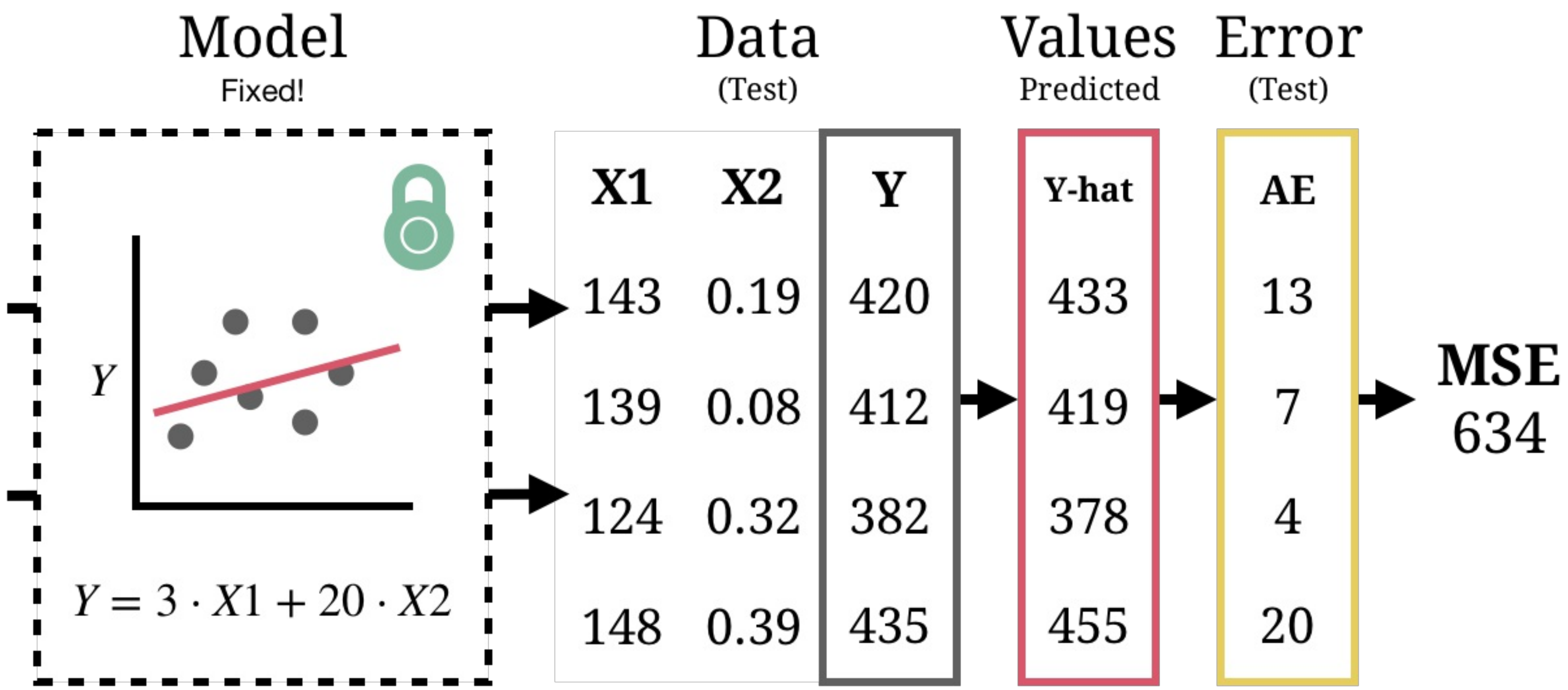
Training

Training a model means to **fit the model** to data by finding the parameter combination that **minizes some error function**, e.g., mean squared error (MSE).



Test

To test a model means to **evaluate the prediction error** for a fitted model, i.e., for a **fixed parameter combination**.



Why do we separate training from testing?

"Can you come up with a model that will perfectly fit the training criterion but is worthless in predicting test data?"

Training data

id	sex	age	fam_history	smoking	criterion
1	m	45	No	FALSE	0
2	m	43	Yes	FALSE	1
3	f	40	Yes	FALSE	1
4	m	51	Yes	FALSE	1
5	m	44	No	TRUE	0

Test data

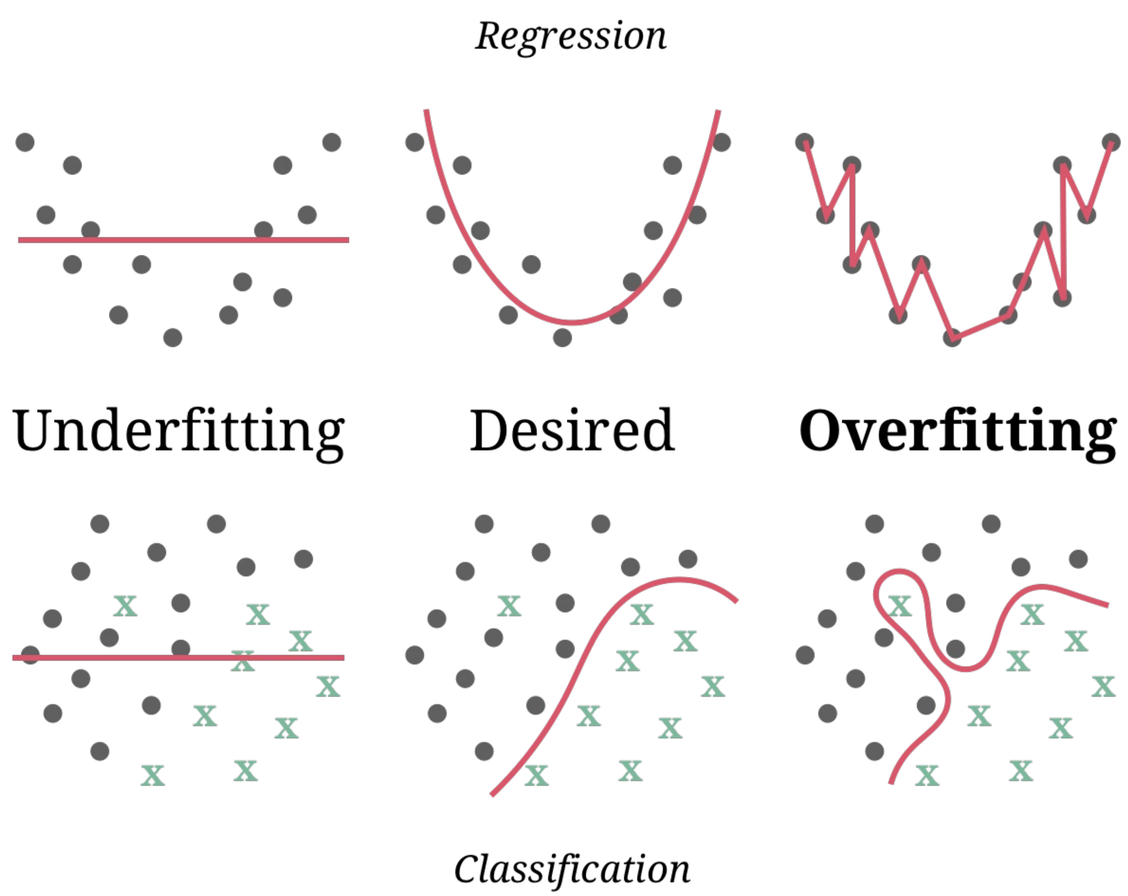
id	sex	age	fam_history	smoking	criterion
91	m	51	Yes	TRUE	?
92	f	47	No	TRUE	?
93	m	39	No	TRUE	?
94	f	51	Yes	TRUE	?
95	f	50	Yes	FALSE	?

Overfitting

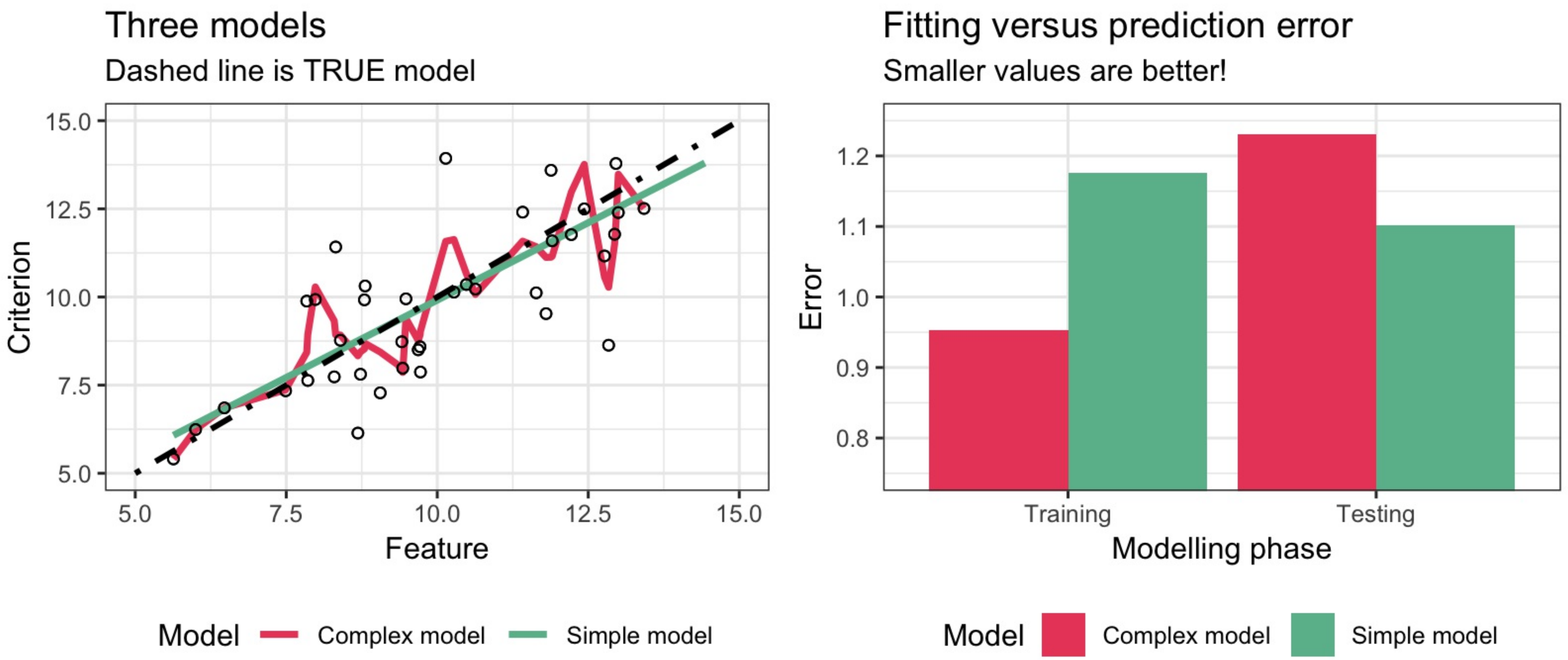
Occurs when a model **fits data too closely** and therefore **fails to reliably predict** future observations.

In other words, overfitting occurs when a model **'mistakes' random noise for a predictable signal**.

More **complex models** are more **prone to overfitting**.



Overfitting



Two new models enter the ring...

Regression

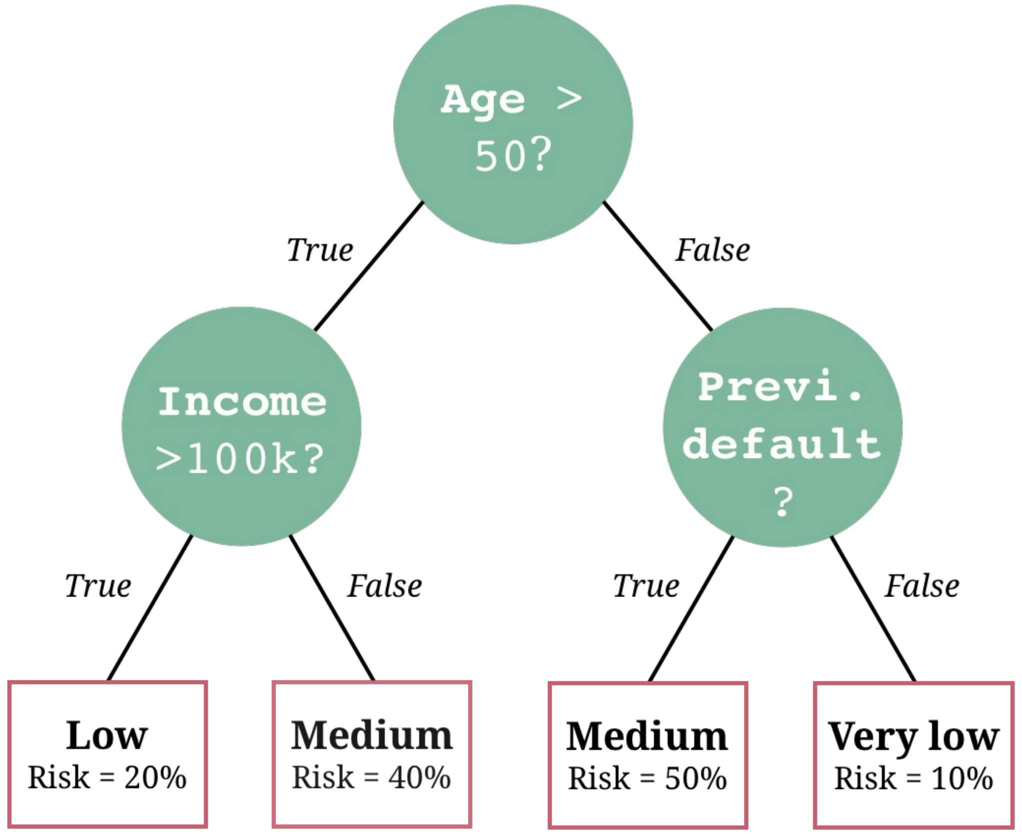
Decision Trees

Random Forests

CART

CART is short for **Classification and Regression Trees**,.which are often just called **Decision trees**.

In **decision trees**, the criterion is modeled as a **sequence of logical TRUE or FALSE questions**.



Classificiation trees

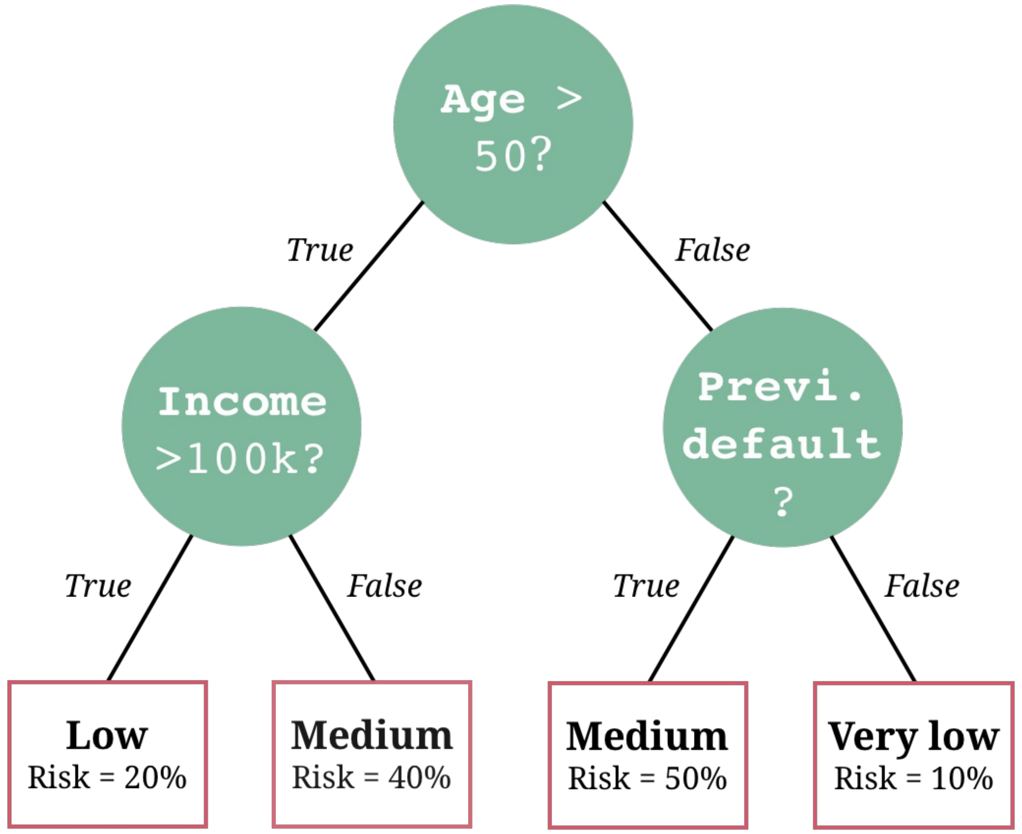
Classification trees (and regression trees) are created using a relatively simple **three-step algorithm**.

Algorithm

1 - **Split** nodes to maximize **purity gain** (e.g., Gini gain).

2 - **Repeat** until pre-defined threshold (e.g., `minsplit`) splits are no longer possible.

3 - **Prune** tree to reasonable size.



Node splitting

Classification trees attempt to **minize node impurity** using, e.g., the **Gini coefficient**.

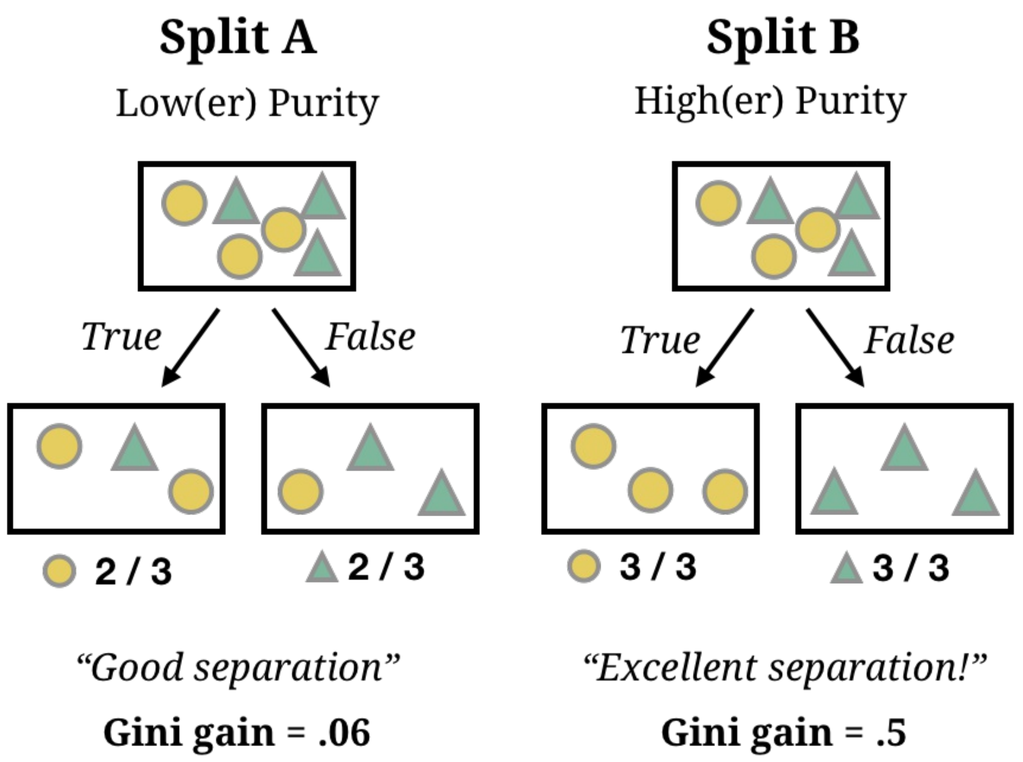
$$Gini(S) = 1 - \sum_j^k p_j^2$$

Nodes are **split** using the variable and split value that **maximizes Gini gain**.

$$Gini\ gain = Gini(S) - Gini(A, S)$$

with

$$Gini(A, S) = \sum \frac{n_i}{n} Gini(S_i)$$

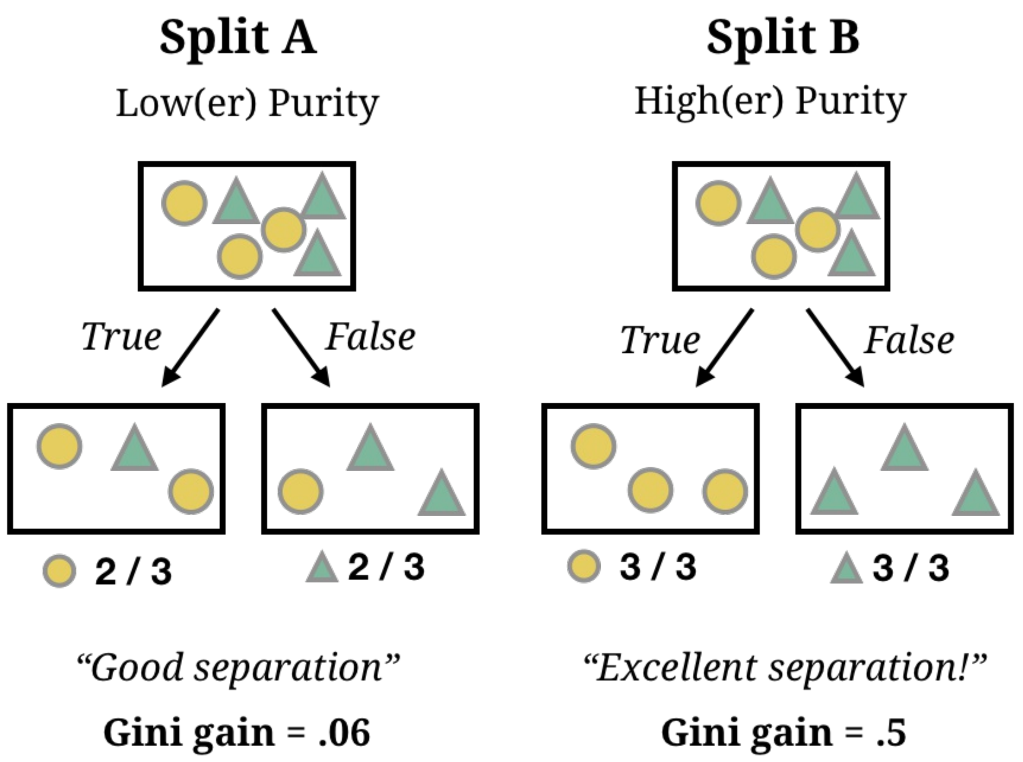


Pruning trees

Classification trees are **pruned** back such that every split has a purity gain of at least `cp`, with `cp` typically set to `.01`.

Minimize:

$$
\begin{aligned}
Loss = Impurity\ + \\
cp * (n\ terminal\ nodes)
\end{aligned}
$$



Regression trees

Trees can also be used to perform regression tasks. Instead of impurity, regression trees attempt to **minimize within-node variance** (or maximize node homogeneity):

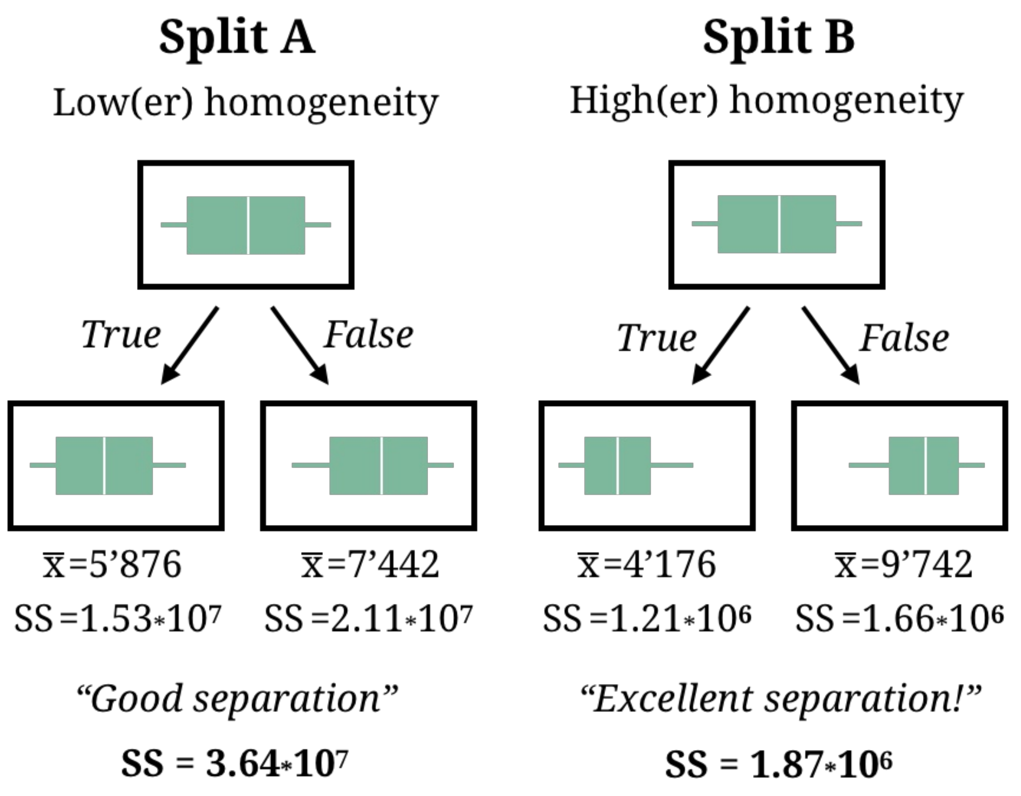
$$SSE = \sum_{i \in S_1} (y_i - \bar{y}_1)^2 + \sum_{i \in S_2} (y_i - \bar{y}_2)^2$$

<u>Algorithm</u>

1 - **Split** nodes to maximize **homogeneity gain**.

2 - **Repeat** until pre-defined threshold (e.g., `minsplit`) splits are no longe possible.

3 - **Prune** tree to reasonable size.



CART in `caret`

Fit **decision trees** in `caret` using `method = "rpart"`.

`caret` will **choose automatically** whether to use classification or regression trees depending on whether the criterion is a `factor` or not.

```
# Fit a decision tree predicting default

train(form = default ~ .,
      data = Loans,
      method = "rpart", # Decision Tree
      trControl = ctrl)

# Fit a decision tree predicting income

train(form = income ~ .,
      data = baselers,
      method = "rpart", # Decision Tree
      trControl = ctrl)
```

Regression

Decision Trees

Random Forests

Random Forest

In **Random Forest**, the criterion is modeled as the **aggregate prediction of a large number of decision trees** each based on different features.

Algorithm

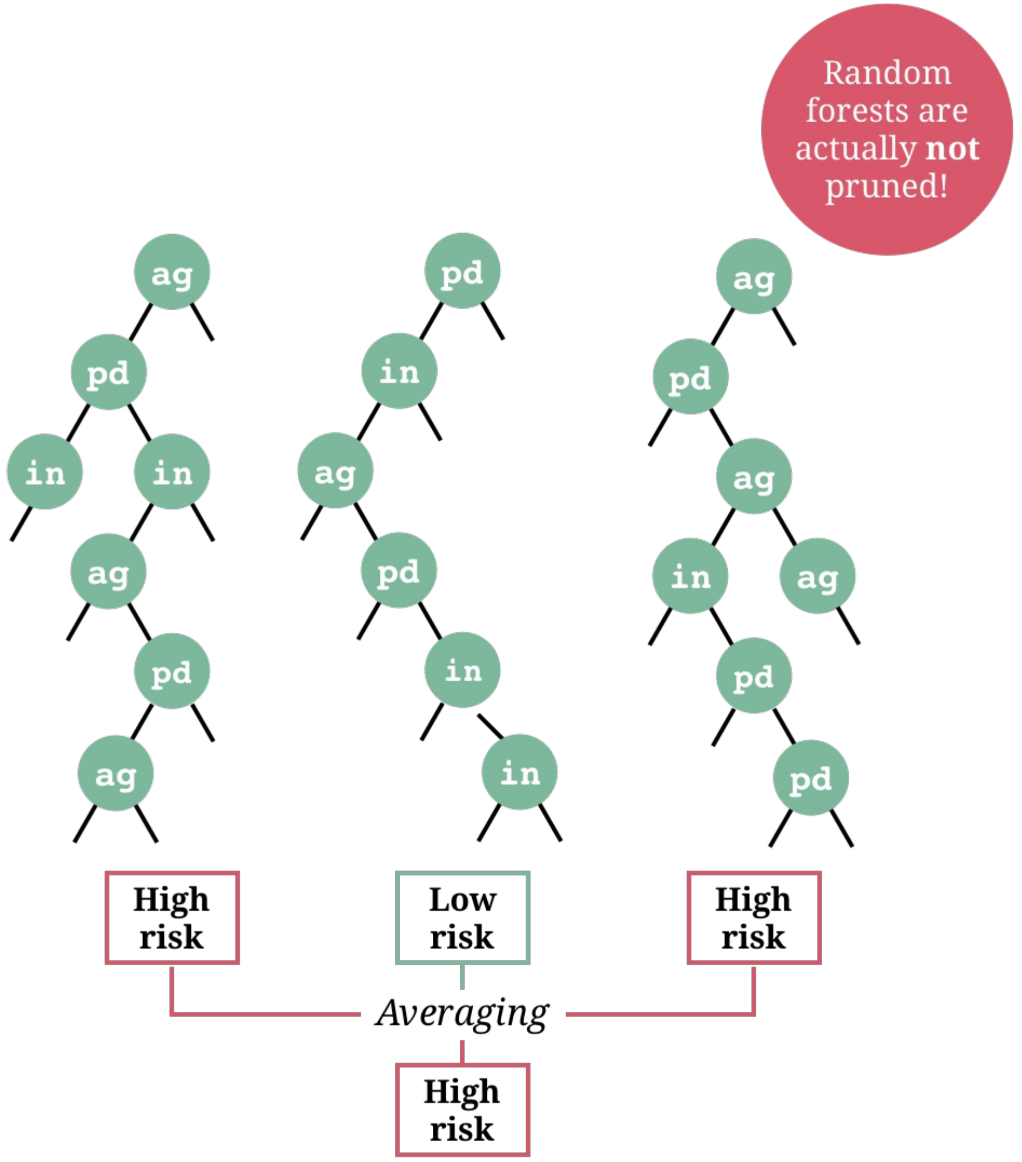
1 - **Repeat** n times

  1 - **Resample** data

  2 - **Grow** non-pruned decision tree

    Each split **consider only** m **features**

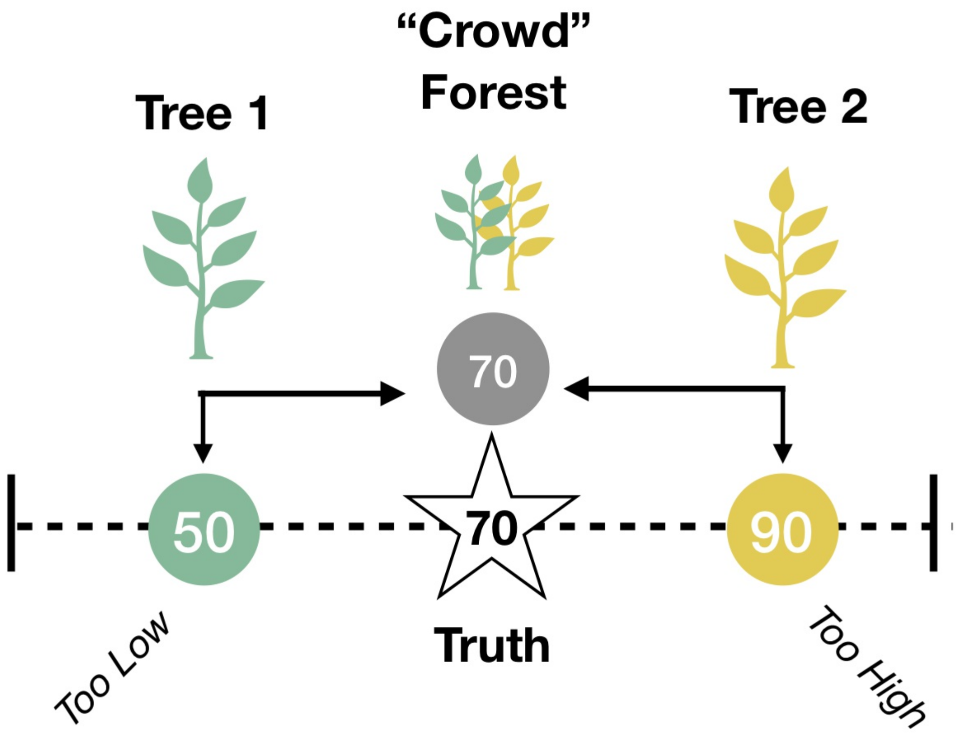
2 - **Average** fitted values



Random Forest

Random forests make use of important machine learning elements, **resampling** and **averaging** that together are also referred to as **bagging**.

Element	Description
Resampling	Creates new data sets that vary in their composition thereby **deemphasizing idiosyncracies** of the available data.
Averaging	Combining predictions typically **evens out idiosyncracies** of the models created from single data sets.



Random forests in `caret`

Fit **decision trees** in `caret` using `method = "rf"`.

Just like CART, random forests can be used for **classification or regression**.

`caret` will **choose automatically** whether to use classification or regression trees depending on whether the crition is a `factor` or not.

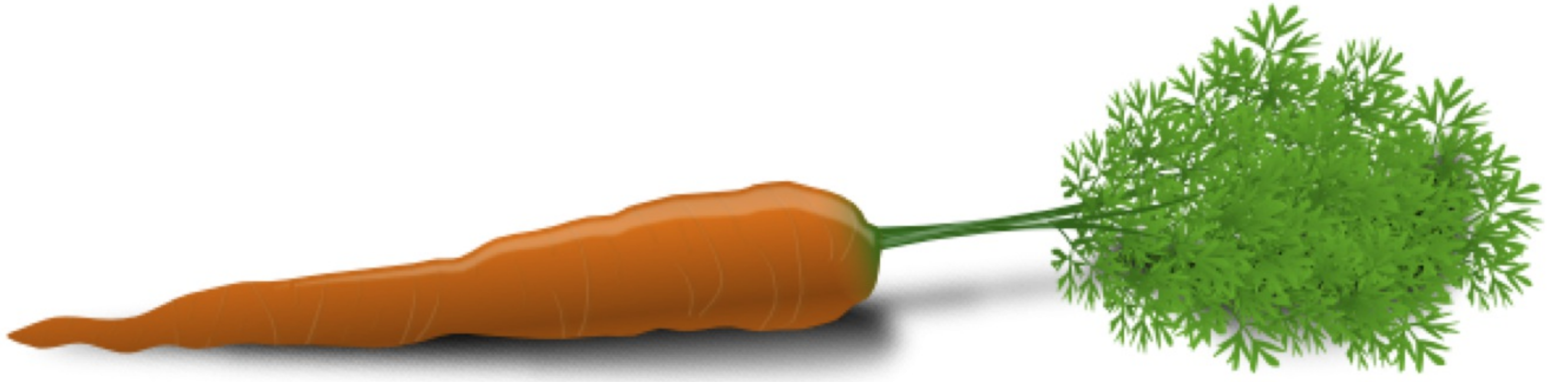
```
# Fit a decision tree predicting default

train(form = default ~ .,
      data = Loans,
      method = "rf", # Decision Tree
      trControl = ctrl)

# Fit a decision tree predicting income

train(form = income ~ .,
      data = baselers,
      method = "rf", # Decision Tree
      trControl = ctrl)
```

Evaluating model predictions with caret



createDataPartition()

Use `createDataPartition()` to **split a dataset** into separate training and test datasets.

Argument	Description
y	The criterion. Used to create a **balanced split**.
p	The **proportion of data** going into the training set. Often `.8` or `.5`.

```
# Set the randomisation seed to get the
#  same results each time
set.seed(100)

# Get indices for training
index <-
  createDataPartition(y = baselers$income,
                      p = .8,
                      list = FALSE)

# Create training data
baselers_train <- baselers %>%
  slice(index)

# Create test data
baselers_test <- baselers %>%
  slice(-index)
```

predict(, newdata)

To **test model predictions** with `caret`, all you need to do is get a vector of predictions from a new dataframe `newdata` using the `predict()` function:

Argument	Description
`object`	`caret` fit object.
`newdata`	Test data sest. Must contain same features as provided in `object`.

```r
# Fit model to training data
mod <- train(form = income ~ .,
             method = "glm",
             data = baselers_train)

# Get fitted values (for training data)
mod_fit <- predict(mod)

# Predictions for NEW data_test data!
mod_pred <- predict(mod,
                    newdata = baselers_test)

# Evaluate prediction results
postResample(pred = mod_pred,
             obs = baselers_test$income)
```

Practical