HANDS-ON AI I

Supervised Machine Learning Basics

Andreas Schörgenhumer
Institute for Machine Learning

JOHANNES KEPLER
UNIVERSITY LINZ

Institute for
Machine Learning

Copyright Statement

Content of Unit 3

- Introduction to machine learning
- Some machine learning algorithms

INTRODUCTION TO MACHINE LEARNING

How to Solve These Tasks?

- Prediction of trajectory of a space shuttle
- Translation of one language into another
- Prediction of protein function
- Automatic recognition of handwritten digits
- Object detection in images

Explicit Models

Traditional approach: **Explicit model**

- Use explicit knowledge to design model **deductively**.

Explicit Models

Traditional approach: **Explicit model**

- Use explicit knowledge to design model **deductively**.
- Pros:
  - Knowledge about behavior of model and environment/problem.
  - Knowledge about restrictions of model and reasons for design choices.

Explicit Models

Traditional approach: **Explicit model**

- Use explicit knowledge to design model **deductively**.
- Pros:
  - Knowledge about behavior of model and environment/problem.
  - Knowledge about restrictions of model and reasons for design choices.
- Cons:
  - Sometimes problem is too complex to model.
  - Consequences of simplifications of problem/model hard to assess.
  - Insufficient knowledge about problem/environment.

Supervised Machine Learning

Supervised Machine Learning:

- Learning from **input values** and corresponding **target values**:
  - E.g., image + object type, DNA sequence + phenotype, ...

Supervised Machine Learning

Supervised Machine Learning:

- Learning from **input values** and corresponding **target values**:
  - E.g., image + object type, DNA sequence + phenotype, ...
- Typical usage: **predictive modeling**
  - **Train** model on data set with input + target values.
  - Use trained model to **predict** target values for other (new) inputs where the targets are not known yet.

Supervised Machine Learning

Supervised Machine Learning:

- Learning from **input values** and corresponding **target values**:
  - E.g., image + object type, DNA sequence + phenotype, ...
- Typical usage: **predictive modeling**
  - **Train** model on data set with input + target values.
  - Use trained model to **predict** target values for other (new) inputs where the targets are not known yet.
- **Classification**: target value is class label

Supervised Machine Learning

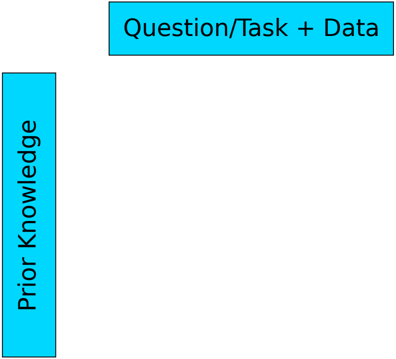
Supervised Machine Learning:

- Learning from **input values** and corresponding **target values**:
  - E.g., image + object type, DNA sequence + phenotype, ...
- Typical usage: **predictive modeling**
  - **Train** model on data set with input + target values.
  - Use trained model to **predict** target values for other (new) inputs where the targets are not known yet.
- **Classification**: target value is class label
- **Regression**: target value is numerical value

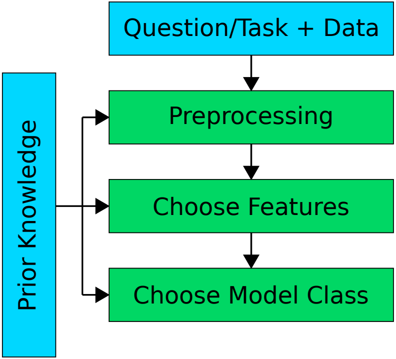
Terminology

- **Model**: parameterized function/method with specific parameter values (e.g., a trained neural network)
- **Model class**: the class of models in which we search for the model (e.g., neural networks, SVMs, ...)
- **Parameters**: what is adjusted during training (e.g., network weights)
- **Hyperparameters**: settings controlling model complexity or the training procedure (e.g., network learning rate)
- **Model selection/training**: process of finding a model (optimal parameters) from the model class

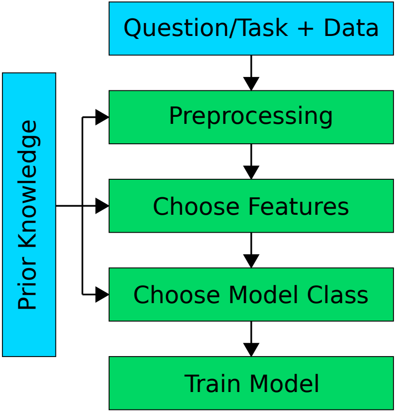
Basic Data Analysis Workflow



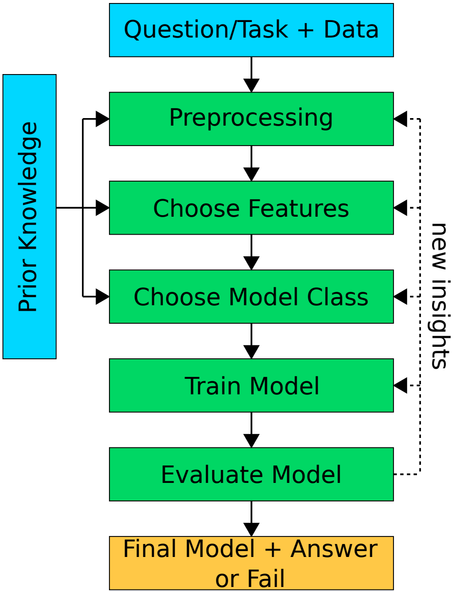
Basic Data Analysis Workflow



Basic Data Analysis Workflow



Basic Data Analysis Workflow



Introductory Example: Fish Recognition

- Example from

  R. O. Duda, P. E. Hart, and D. G. Stork. Pattern Classification*. 2nd edition. John Wiley & Sons, 2001. ISBN 0-471-05669-3.*

- Automated system to sort fish in a fish-packing company: salmons must be distinguished from sea bass optically.

Introductory Example: Fish Recognition

- Example from

  R. O. Duda, P. E. Hart, and D. G. Stork. Pattern Classification*. 2nd edition. John Wiley & Sons, 2001. ISBN 0-471-05669-3.*

- Automated system to sort fish in a fish-packing company: salmons must be distinguished from sea bass optically.
- **Given**: a set of pictures with fish labels.

Introductory Example: Fish Recognition

- Example from

  R. O. Duda, P. E. Hart, and D. G. Stork. Pattern Classification*. 2nd edition. John Wiley & Sons, 2001. ISBN 0-471-05669-3.*
- Automated system to sort fish in a fish-packing company: salmons must be distinguished from sea bass optically.
- **Given**: a set of pictures with fish labels.
- **Goal**: distinguish between salmons and sea bass.

Introductory Example: Fish Recognition

- Example from
  R. O. Duda, P. E. Hart, and D. G. Stork. Pattern Classification. *2nd edition. John Wiley & Sons, 2001. ISBN 0-471-05669-3.*

- Automated system to sort fish in a fish-packing company: salmons must be distinguished from sea bass optically.
- **Given**: a set of pictures with fish labels.
- **Goal**: distinguish between salmons and sea bass.

$\rightarrow$ **Classification** task with two labels (salmon vs. sea bass, or, alternatively, salmon vs. not salmon)

Our Data (Two Sample Images)

Salmon:

Sea bass:

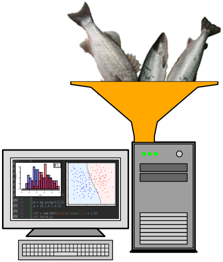
How can we distinguish these two kinds of fish?

Our Data (Two Sample Images)

Salmon:

Sea bass:

How can we distinguish these two kinds of fish?



Our Data (Two Sample Images)

Salmon:

Sea bass:

How can we distinguish these two kinds of fish?

First step: Let's take a look at our data!

Feature Selection & Preprocessing

- **Feature selection**:
  - What data do we have?
  - Removal of redundant features.
  - Removal of features the model class cannot utilize.
  - (**Deep Learning**: Feature selection mainly by neural network.)

Feature Selection & Preprocessing

- **Feature selection**:
  - What data do we have?
  - Removal of redundant features.
  - Removal of features the model class cannot utilize.
  - (**Deep Learning**: Feature selection mainly by neural network.)
- **Preprocessing**:
  - Contrast and brightness correction
  - Segmentation
  - Alignment
  - Normalization
  - ...

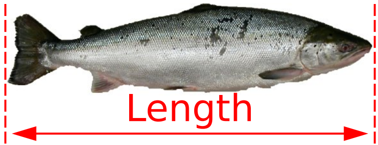
Back to Our Data

Salmon:

Sea bass:

Back to Our Data

Salmon:

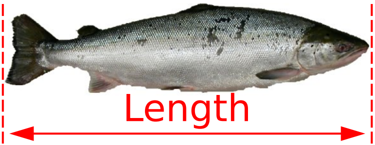


Sea bass:

- Assume we use **length** and **brightness** as features.
- For simplicity, also assume that some person extracted these features for us.

Back to Our Data

Salmon:



Sea bass:

- Assume we use **length** and **brightness** as features.
- For simplicity, also assume that some person extracted these features for us.

$\rightarrow$ How do we express/represent these features?

Input Representation

- We can represent an object by a vector $\boldsymbol{x}$ of feature values (=**feature vector**) of length d and **label** y:

$$\boldsymbol{x} = (x_1, \ldots, x_d) \qquad y$$

Input Representation

- We can represent an object by a vector $\boldsymbol{x}$ of feature values (=**feature vector**) of length d and **label** y:

$$\boldsymbol{x} = (x_1, \ldots, x_d) \qquad y$$

  - Fish example: A fish is represented as feature vector with two values *length* and *brightness* (i.e., $d = 2$) and one label ($y =$ "salmon" or $y =$ "sea bass").

Input Representation

- We can represent an object by a vector $\boldsymbol{x}$ of feature values (=**feature vector**) of length d and **label** y:

$$\boldsymbol{x} = (x_1, \ldots, x_d) \qquad y$$

  - Fish example: A fish is represented as feature vector with two values *length* and *brightness* (i.e., $d = 2$) and one label ($y =$ "salmon" or $y =$ "sea bass").
- An object described by one feature vector and one label is referred to as **sample**: $(\boldsymbol{x}, y)$.

Input Representation

- Assume our data set consists of n objects with feature vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ of length d, and each object has a corresponding label $y_1, \ldots, y_n$.

Input Representation

- Assume our data set consists of n objects with feature vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ of length d, and each object has a corresponding label $y_1, \ldots, y_n$.
- Then, we can write the feature vectors of all objects in a **matrix of feature vectors** $\boldsymbol{X}$ and the labels in a corresponding labels vector $\boldsymbol{y}$:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_n \end{bmatrix} = \begin{bmatrix} x_{11} & \cdots & x_{1d} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nd} \end{bmatrix} \qquad \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

Input Representation

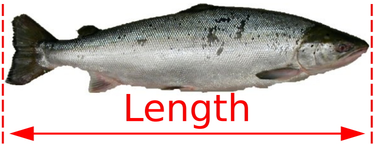
- Assume our data set consists of n objects with feature vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ of length d, and each object has a corresponding label $y_1, \ldots, y_n$.
- Then, we can write the feature vectors of all objects in a **matrix of feature vectors** $\boldsymbol{X}$ and the labels in a corresponding labels vector $\boldsymbol{y}$:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_n \end{bmatrix} = \begin{bmatrix} x_{11} & \cdots & x_{1d} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nd} \end{bmatrix} \qquad \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

- Our labeled data is thus described by: $(\boldsymbol{X}, \boldsymbol{y})$.

Back to Our Data

Salmon:

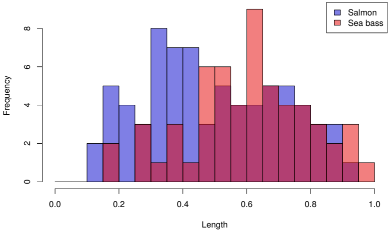


Sea bass:

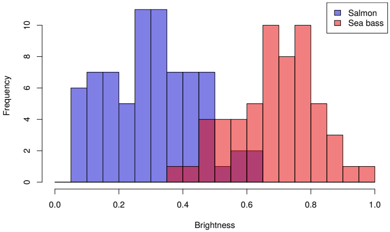
- We now know how to represent our data (i.e., using **features** and **labels**) and will take a look at it via histograms.

Back to Our Data

Length:



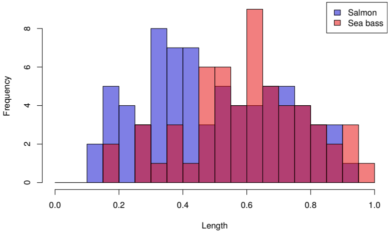
Brightness:



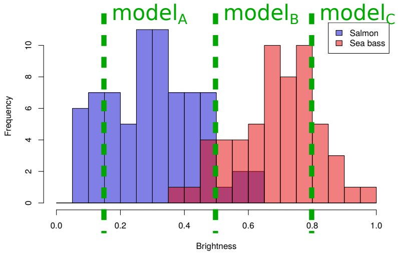
- Brightness looks more useful for fish classification.

Back to Our Data

Length:



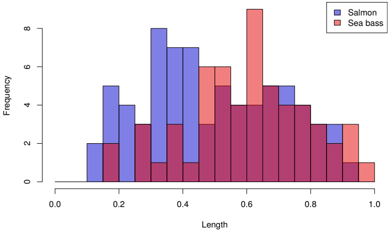
Brightness:



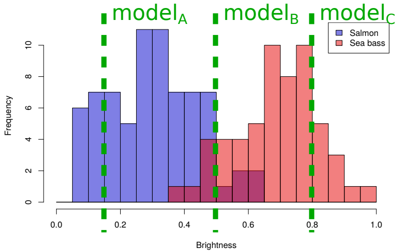
- Brightness looks more useful for fish classification.
- 3 different models based on brightness threshold:
  - model_A: brightness $< 0.18 \rightarrow$ Salmon
  - model_B: brightness $< 0.5 \rightarrow$ Salmon
  - model_C: brightness $< 0.8 \rightarrow$ Salmon

Back to Our Data

Length:



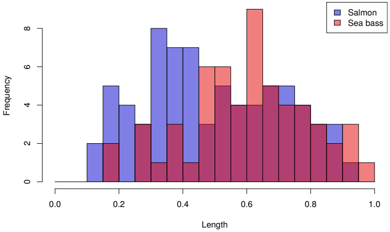
Brightness:



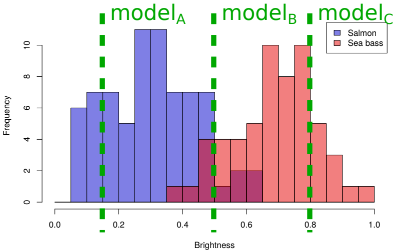
- How do we get the “best” model?

Back to Our Data

Length:



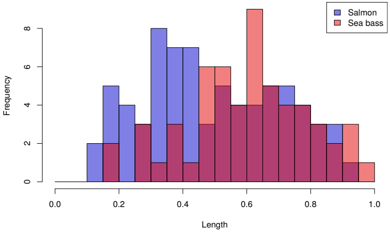
Brightness:



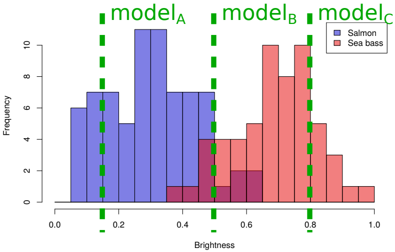
- How do we get the "best" model?
  - How does our model perform on our data?
    – Loss function

Back to Our Data

Length:



Brightness:



- How do we get the “best” model?
  - How does our model perform on our data?
    – **Loss function**
  - How will it perform on (unseen) future data?
    – **Generalization error/risk**

LOSS FUNCTION

Scoring Our Models: Loss Function

- Assume we have a model g, parameterized by w.

Scoring Our Models: Loss Function

- Assume we have a model g, parameterized by $\boldsymbol{w}$.
- $g(\boldsymbol{x}; \boldsymbol{w})$ maps an input vector $\boldsymbol{x}$ to an output value $\hat{y}$.

Scoring Our Models: Loss Function

- Assume we have a model g, parameterized by $\boldsymbol{w}$.
- $g(\boldsymbol{x}; \boldsymbol{w})$ maps an input vector $\boldsymbol{x}$ to an output value $\hat{y}$.
- We want $\hat{y}$ to be as close as possible to the true target value y.

Scoring Our Models: Loss Function

- Assume we have a model g, parameterized by $\boldsymbol{w}$.
- $g(\boldsymbol{x}; \boldsymbol{w})$ maps an input vector $\boldsymbol{x}$ to an output value $\hat{y}$.
- We want $\hat{y}$ to be as close as possible to the true target value y.
- We can use a **loss function**

$$L(y, g(\boldsymbol{x}; \boldsymbol{w})) = L(y, \hat{y})$$

  to measure how close our prediction is to the true target for a given sample with $(\boldsymbol{x}, y)$.

Scoring Our Models: Loss Function

- Assume we have a model g, parameterized by $\boldsymbol{w}$.
- $g(\boldsymbol{x}; \boldsymbol{w})$ maps an input vector $\boldsymbol{x}$ to an output value $\hat{y}$.
- We want $\hat{y}$ to be as close as possible to the true target value y.
- We can use a **loss function**

$$L(y, g(\boldsymbol{x}; \boldsymbol{w})) = L(y, \hat{y})$$

  to measure how close our prediction is to the true target for a given sample with $(\boldsymbol{x}, y)$.
- **The smaller the loss/cost, the better our prediction**.

Examples of Loss Functions

Zero-one loss: $L_{\mathbf{zo}}(y, g(\boldsymbol{x}; \boldsymbol{w})) = \begin{cases} 0 & y = g(\boldsymbol{x}; \boldsymbol{w}) \\ 1 & y \neq g(\boldsymbol{x}; \boldsymbol{w}) \end{cases}$

Quadratic loss: $L_{\mathbf{q}}(y, g(\boldsymbol{x}; \boldsymbol{w})) = (y - g(\boldsymbol{x}; \boldsymbol{w}))^2$

Examples of Loss Functions

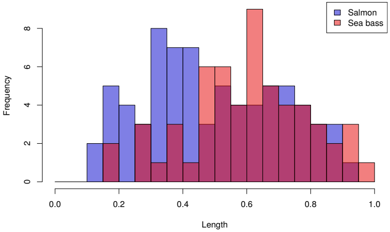
Zero-one loss: $L_{\mathbf{zo}}(y, g(\boldsymbol{x}; \boldsymbol{w})) = \begin{cases} 0 & y = g(\boldsymbol{x}; \boldsymbol{w}) \\ 1 & y \neq g(\boldsymbol{x}; \boldsymbol{w}) \end{cases}$

Quadratic loss: $L_{\mathbf{q}}(y, g(\boldsymbol{x}; \boldsymbol{w})) = (y - g(\boldsymbol{x}; \boldsymbol{w}))^2$

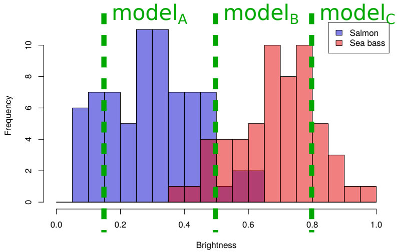
- Many other loss functions available with different justifications.
- Not every loss function is suitable for every task.
- Choice of loss function depends on data, task, and model class.

Back to Our Data

Length:



Brightness:



- How do we get the "best" model?
  - How does our model perform on our data?
    – **Loss function** ✓
  - How will it perform on (unseen) future data?
    – **Generalization error/risk**

GENERALIZATION ERROR/RISK

Generalization Error/Risk

- The **generalization error** or **risk** is the expected loss on future data for a given model $g(.;\boldsymbol{w})$:

$$R(g(.;\boldsymbol{w})) = \int_{\boldsymbol{X}} \int L(y, g(\boldsymbol{x};\boldsymbol{w})) \cdot p(\boldsymbol{x}, y) dy d\boldsymbol{x}$$

- $R(g(\boldsymbol{x};\boldsymbol{w}))$ denotes the **expected loss** for input $\boldsymbol{x}$, and $p(\boldsymbol{x}, y)$ is the joint probability distribution for $\boldsymbol{x}$ and y.

Generalization Error/Risk

- The **generalization error** or **risk** is the expected loss on future data for a given model $g(.;\boldsymbol{w})$:

$$R(g(.;\boldsymbol{w})) = \int_{\boldsymbol{X}} \int L(y, g(\boldsymbol{x};\boldsymbol{w})) \cdot p(\boldsymbol{x}, y) dy d\boldsymbol{x}$$

- $R(g(\boldsymbol{x};\boldsymbol{w}))$ denotes the **expected loss** for input $\boldsymbol{x}$, and $p(\boldsymbol{x}, y)$ is the joint probability distribution for $\boldsymbol{x}$ and y.
- In practice, we hardly have any knowledge about $p(\boldsymbol{x}, y)$.

Generalization Error/Risk

- The **generalization error** or **risk** is the expected loss on future data for a given model $g(.;\boldsymbol{w})$:

$$R(g(.;\boldsymbol{w})) = \int_{\boldsymbol{X}} \int L(y, g(\boldsymbol{x};\boldsymbol{w})) \cdot p(\boldsymbol{x}, y) dy d\boldsymbol{x}$$

- $R(g(\boldsymbol{x};\boldsymbol{w}))$ denotes the **expected loss** for input $\boldsymbol{x}$, and $p(\boldsymbol{x}, y)$ is the joint probability distribution for $\boldsymbol{x}$ and y.
- In practice, we hardly have any knowledge about $p(\boldsymbol{x}, y)$.

$\rightarrow$ We have to **estimate the generalization error**:

- This is called **empirical risk minimization (ERM)**

Empirical Risk Minimization (ERM)

- We do not know the true $p(\boldsymbol{x}, y)$ but we have access to a subset of n data samples $\rightarrow$ our data set $(\boldsymbol{X}, \boldsymbol{y})$.

Empirical Risk Minimization (ERM)

- We do not know the true $p(\boldsymbol{x}, y)$ but we have access to a subset of n data samples $\rightarrow$ our data set $(\boldsymbol{X}, \boldsymbol{y})$.
- We can minimize the **empirical risk** R_E on our data set (=**Empirical Risk Minimization**):

$$R_E(g(.;\boldsymbol{w}), (\boldsymbol{X}, y)) = \frac{1}{n} \cdot \sum_{i=1}^{n} L(y_i, g(\boldsymbol{x}_i; \boldsymbol{w}))$$

Empirical Risk Minimization (ERM)

- We do not know the true $p(\boldsymbol{x}, y)$ but we have access to a subset of n data samples $\rightarrow$ our data set $(\boldsymbol{X}, \boldsymbol{y})$.
- We can minimize the **empirical risk** R_E on our data set (=**Empirical Risk Minimization**):

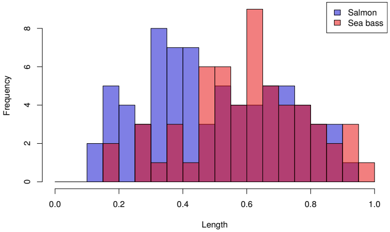
$$R_E(g(.;\boldsymbol{w}), (\boldsymbol{X}, y)) = \frac{1}{n} \cdot \sum_{i=1}^{n} L(y_i, g(\boldsymbol{x}_i; \boldsymbol{w}))$$

- Law of large numbers:

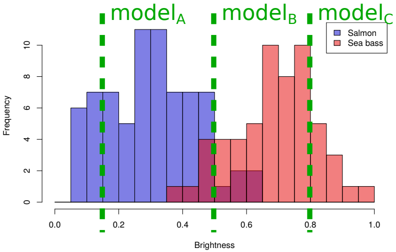
$$R_E(g(.;\boldsymbol{w})) \rightarrow R(g(.;\boldsymbol{w})) \quad \text{for } n \rightarrow \infty$$

Back to Our Data

Length:



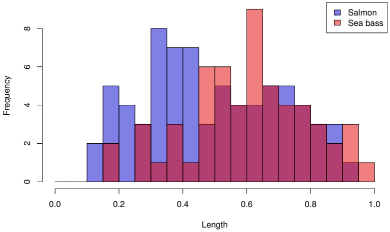
Brightness:



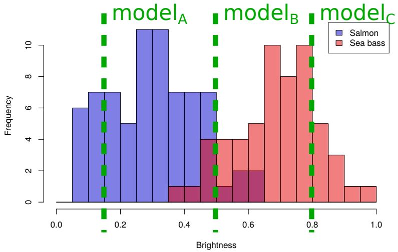
- How do we get the "best" model?
  - How does our model perform on our data?
    – **Loss function** ✓
  - How will it perform on (unseen) future data?
    – **Generalization error/risk** ✓

Back to Our Data

Length:



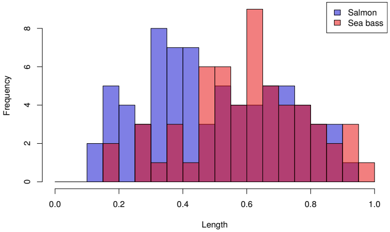
Brightness:



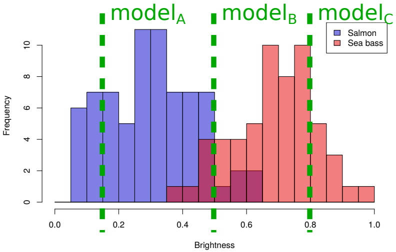
- We can now optimize our model by minimizing the risk on our (training) data set.

Back to Our Data

Length:



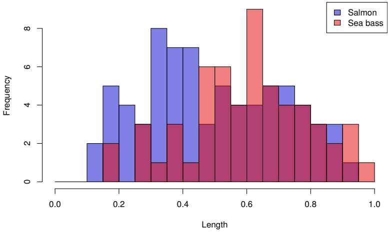
Brightness:



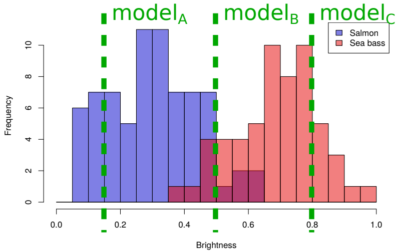
- We can now optimize our model by minimizing the risk on our (training) data set.
- But the individual features (especially length) do not separate the classes well.

Back to Our Data

Length:



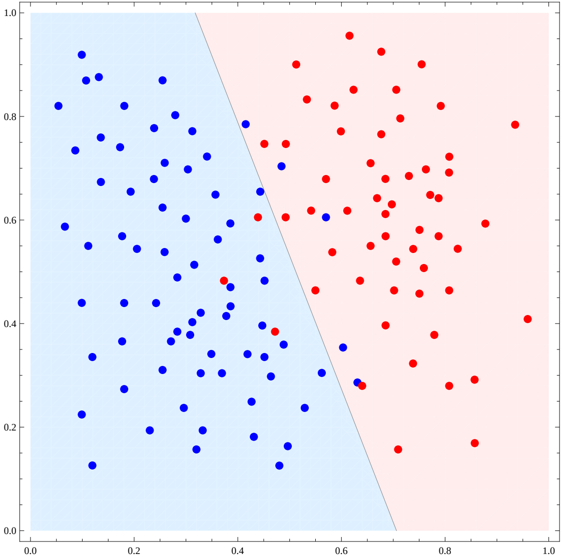
Brightness:



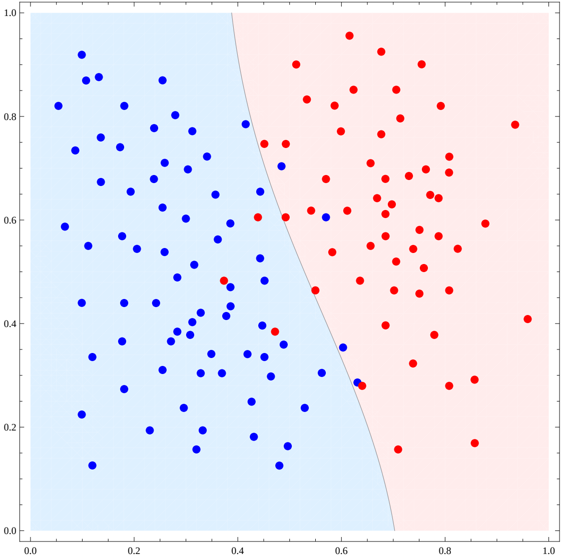
- We can now optimize our model by minimizing the risk on our (training) data set.
- But the individual features (especially length) do not separate the classes well.

$\rightarrow$ **Combine our features** and use a different model class.

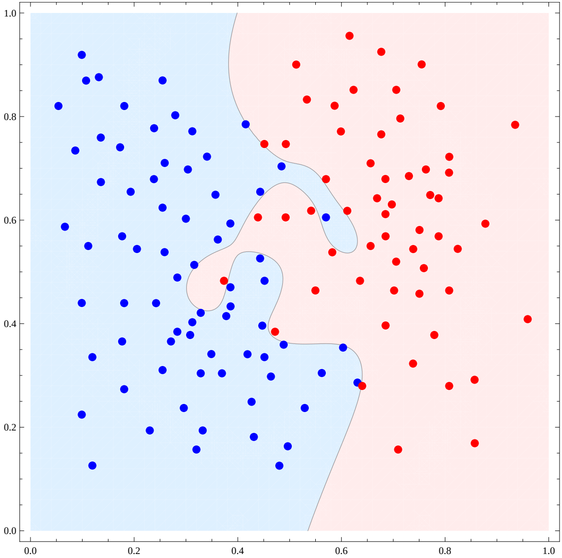
Combination: Linear Separation



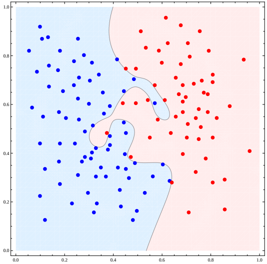
Combination: Mildly Non-linear Separation



Combination: Highly Non-linear Separation

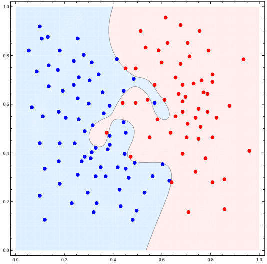


The Problem of Overfitting



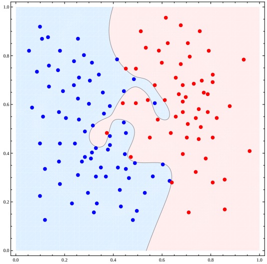
- With ERM, we can optimize our model by minimizing the loss on our data set.

The Problem of Overfitting



- With ERM, we can optimize our model by minimizing the loss on our data set.
- Problem: We might fit our parameters to noise specific to our data set (=**overfitting**).

The Problem of Overfitting



- With ERM, we can optimize our model by minimizing the loss on our data set.
- Problem: We might fit our parameters to noise specific to our data set (=**overfitting**).

$\rightarrow$ We need to get a better estimate for the (true) risk.

Risk Estimation: Test Set Method

- Assume our data samples are **independently and identically distributed (i.i.d.)**[1]

[1] i.i.d.: Each sample has the same probability distribution as the others, and all samples are mutually independent.

Risk Estimation: Test Set Method

- Assume our data samples are **independently and identically distributed (i.i.d.)**[1]
- We can split our data set of n samples into **two non-overlapping subsets**:

[1] i.i.d.: Each sample has the same probability distribution as the others, and all samples are mutually independent.

Risk Estimation: Test Set Method

- Assume our data samples are **independently and identically distributed (i.i.d.)**[1]
- We can split our data set of n samples into **two non-overlapping subsets**:
  - **Training set**: a subset with l samples we perform ERM on (i.e., optimize parameters on)

[1] i.i.d.: Each sample has the same probability distribution as the others, and all samples are mutually independent.

Risk Estimation: Test Set Method

- Assume our data samples are **independently and identically distributed (i.i.d.)**[1]
- We can split our data set of n samples into **two non-overlapping subsets**:
  - **Training set**: a subset with l samples we perform ERM on (i.e., optimize parameters on)
  - **Test set**: a subset with m samples we use to estimate the risk (test data = approximation of future, unseen data)

[1] i.i.d.: Each sample has the same probability distribution as the others, and all samples are mutually independent.

Risk Estimation: Test Set Method

- Assume our data samples are **independently and identically distributed (i.i.d.)**[1]
- We can split our data set of n samples into **two non-overlapping subsets**:
  - **Training set**: a subset with l samples we perform ERM on (i.e., optimize parameters on)
  - **Test set**: a subset with m samples we use to estimate the risk (test data = approximation of future, unseen data)
- Our estimate R_E on the test set will show if we overfit to noise in the training set.

[1] i.i.d.: Each sample has the same probability distribution as the others, and all samples are mutually independent.

Risk Estimation: Validation Set

- Often, we repeatedly change hyperparameters in our model, retrain it and evaluate it (**hyperparameter tuning**).

Risk Estimation: Validation Set

- Often, we repeatedly change hyperparameters in our model, retrain it and evaluate it (**hyperparameter tuning**).
- This actually leads to overfitting again:

Risk Estimation: Validation Set

- Often, we repeatedly change hyperparameters in our model, retrain it and evaluate it (**hyperparameter tuning**).
- This actually leads to overfitting again:
  - We change our hyperparameters according to the test set evaluation results.

Risk Estimation: Validation Set

- Often, we repeatedly change hyperparameters in our model, retrain it and evaluate it (**hyperparameter tuning**).
- This actually leads to overfitting again:
  - We change our hyperparameters according to the test set evaluation results.

  → We indirectly use the test set in our training process!

Risk Estimation: Validation Set

- Often, we repeatedly change hyperparameters in our model, retrain it and evaluate it (**hyperparameter tuning**).
- This actually leads to overfitting again:
  - We change our hyperparameters according to the test set evaluation results.
  - → We indirectly use the test set in our training process!
  - This violates our non-overlapping-subset rule and, in turn, the estimation of the generalization error.[2]

[2] We are likely to overestimate the actual performance because we tuned the model towards the test set.

Risk Estimation: Validation Set

- Often, we repeatedly change hyperparameters in our model, retrain it and evaluate it (**hyperparameter tuning**).
- This actually leads to overfitting again:
  - We change our hyperparameters according to the test set evaluation results.
  - → We indirectly use the test set in our training process!
  - This violates our non-overlapping-subset rule and, in turn, the estimation of the generalization error.[2]
- Solution: Create a third non-overlapping **validation set**.

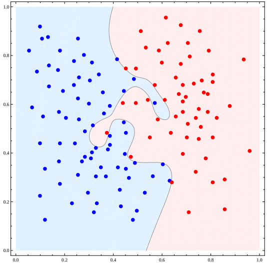
[2] We are likely to overestimate the actual performance because we tuned the model towards the test set.

Risk Estimation: Validation Set

- Often, we repeatedly change hyperparameters in our model, retrain it and evaluate it (**hyperparameter tuning**).
- This actually leads to overfitting again:
  - We change our hyperparameters according to the test set evaluation results.
  - → We indirectly use the test set in our training process!
  - This violates our non-overlapping-subset rule and, in turn, the estimation of the generalization error.[2]
- Solution: Create a third non-overlapping **validation set**.
- Use the validation set for hyperparameter tuning and the test set (once) for the final model evaluation.

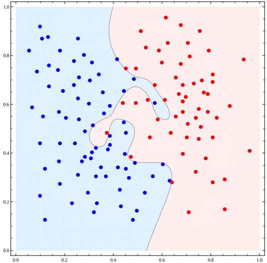
[2] We are likely to overestimate the actual performance because we tuned the model towards the test set.

Back to Our Data



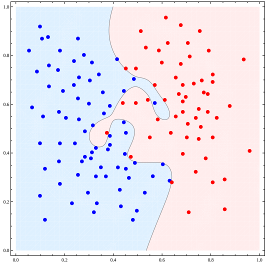
- Now, we can use **ERM** to optimize a model on our **training data set** (optionally, including a validation set).

Back to Our Data



- Now, we can use **ERM** to optimize a model on our **training data set** (optionally, including a validation set).
- A held-out **test set** will allow us to get an **estimate** about the performance on future data.

Back to Our Data



- Now, we can use **ERM** to optimize a model on our **training data set** (optionally, including a validation set).
- A held-out **test set** will allow us to get an **estimate** about the performance on future data.
- If overfitting is detected, we can reduce the model complexity via hyperparameters.

Bias-Variance Tradeoff

- We have introduced ERM and the test-set method to counteract overfitting. But **what if the model is too simple**?

Bias-Variance Tradeoff

- We have introduced ERM and the test-set method to counteract overfitting. But **what if the model is too simple**?
- **Bias-variance tradeoff**:

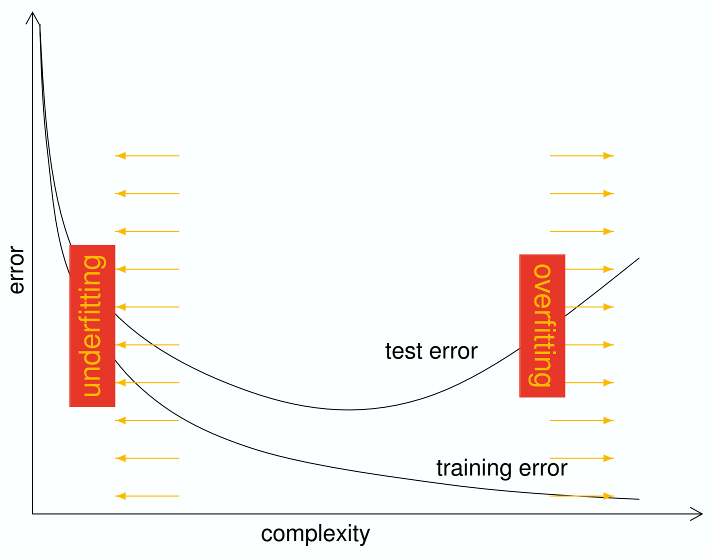
Bias-Variance Tradeoff

- We have introduced ERM and the test-set method to counteract overfitting. But **what if the model is too simple**?
- **Bias-variance tradeoff**:
  - **Underfitting** (high bias): The model is too coarse to fit training data and also too coarse to fit test data. The model complexity is too low.

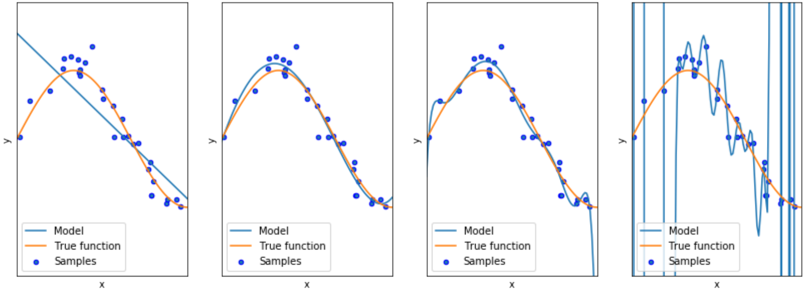
Bias-Variance Tradeoff

- We have introduced ERM and the test-set method to counteract overfitting. But **what if the model is too simple**?
- **Bias-variance tradeoff**:
  - **Underfitting** (high bias): The model is too coarse to fit training data and also too coarse to fit test data. The model complexity is too low.
  - **Overfitting** (high variance): The model fits (too) well to training data but not to future/test data. The model complexity is too high.

Bias-Variance Tradeoff



Bias-Variance Tradeoff



SOME MACHINE LEARNING ALGORITHMS

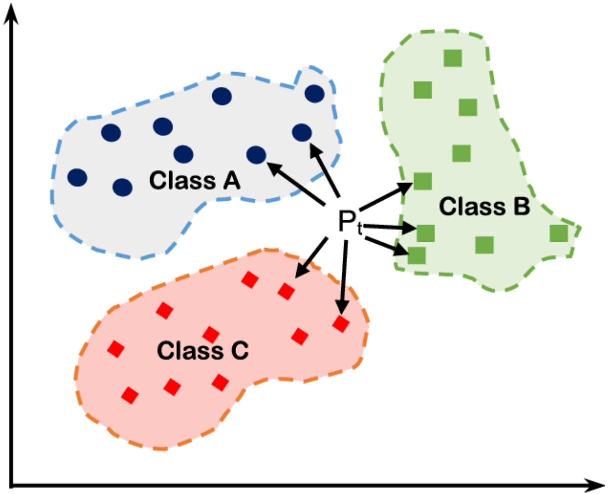
Some Machine Learning Algorithms

- We now look more closely at three different machine learning algorithms:
  - k-nearest neighbors
  - Decision trees
  - Random forest
- This should provide a rough idea on the diversity of ideas and algorithms on the market.

Some Machine Learning Algorithms

- We now look more closely at three different machine learning algorithms:
  - k-nearest neighbors
  - Decision trees
  - Random forest
- This should provide a rough idea on the diversity of ideas and algorithms on the market.
- A profound mathematical treatment of the algorithms is given, e.g., in **Machine Learning: Supervised Techniques**.

k-Nearest Neighbors Classifier



Picture taken from: https://www.researchgate.net/publication/331424423

k-Nearest Neighbors Classifier (1)

- Assume we have a labeled data set $(\boldsymbol{X}, y)$ and a **distance measure on the input space**. Then the **k-nearest neighbors classifier** is defined as follows:

$$g_{k\text{-NN}}(\boldsymbol{x}; \boldsymbol{w}) = \text{class that occurs most often among } k \text{ samples closest to } \boldsymbol{x}$$

k-Nearest Neighbors Classifier (1)

- Assume we have a labeled data set $(\boldsymbol{X}, y)$ and a **distance measure on the input space**. Then the **k-nearest neighbors classifier** is defined as follows:

  $g_{k\text{-NN}}(\boldsymbol{x}; \boldsymbol{w})$ = class that occurs most often among k samples closest to $\boldsymbol{x}$

- For $k = 1$: **nearest neighbor classifier**:

  $g_{\text{NN}}(\boldsymbol{x}; \boldsymbol{w})$ = class of the sample that is closest to $\boldsymbol{x}$

k-Nearest Neighbors Classifier (1)

- Assume we have a labeled data set $(\boldsymbol{X}, y)$ and a **distance measure on the input space**. Then the **k-nearest neighbors classifier** is defined as follows:

  $g_{k\text{-NN}}(\boldsymbol{x}; \boldsymbol{w}) =$ class that occurs most often among k samples closest to $\boldsymbol{x}$

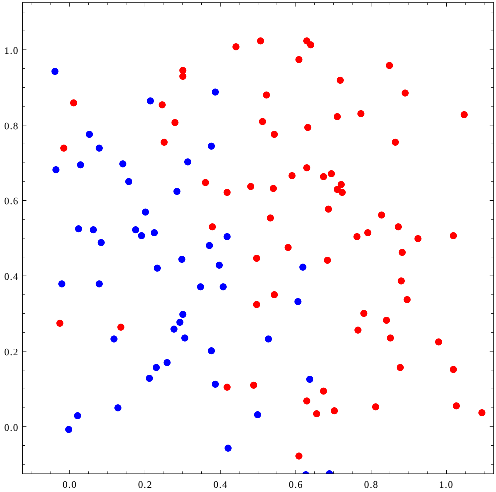
- For $k = 1$: **nearest neighbor classifier**:

  $g_{\text{NN}}(\boldsymbol{x}; \boldsymbol{w}) =$ class of the sample that is closest to $\boldsymbol{x}$

- In case of ties: e.g., random class assignment or class with larger number of samples is assigned.

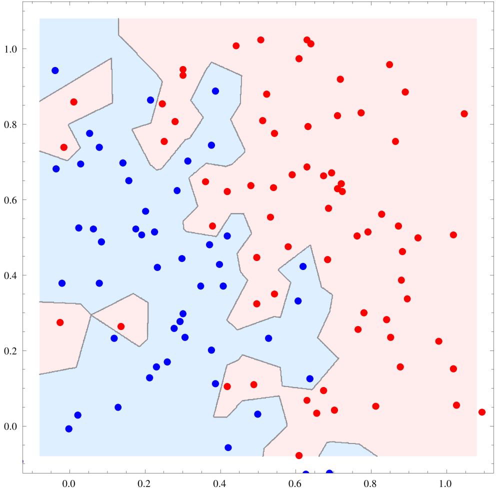
k-Nearest Neighbors Classifier (2)

Input data set



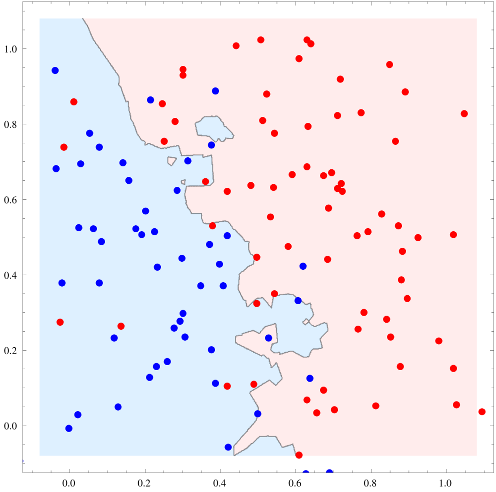
k-Nearest Neighbors Classifier (3)

$k = 1$



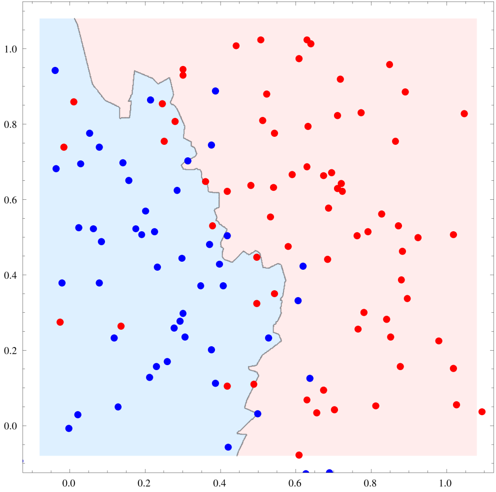
k-Nearest Neighbors Classifier (4)

$k = 5$



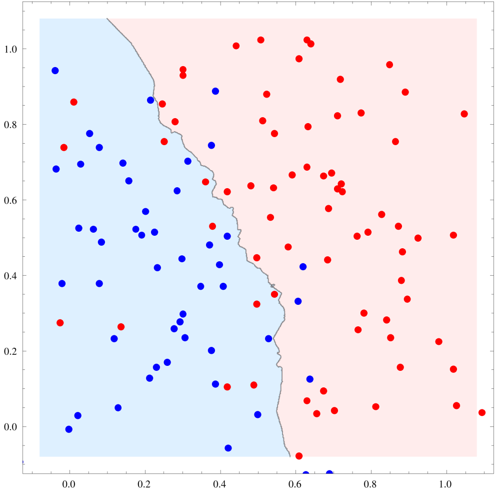
k-Nearest Neighbors Classifier (5)

$k = 13$

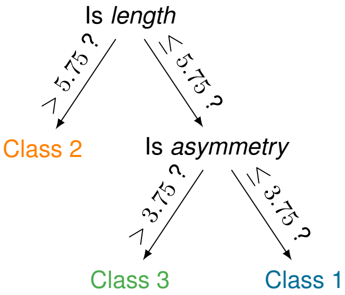


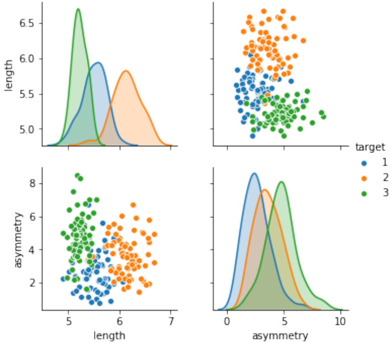
k-Nearest Neighbors Classifier (6)

$k = 25$



Decision Trees





Decision Trees

- A decision tree classifies samples “by asking questions successively”: each **non-leaf** node corresponds to a **question**, each **leaf** corresponds to a final **prediction**.

Decision Trees

- A decision tree classifies samples “by asking questions successively”: each **non-leaf** node corresponds to a **question**, each **leaf** corresponds to a final **prediction**.
- Decision tree learning partitions training data hierarchically.

Decision Trees

- A decision tree classifies samples “by asking questions successively”: each **non-leaf** node corresponds to a **question**, each **leaf** corresponds to a final **prediction**.
- Decision tree learning partitions training data hierarchically.
- There are three main design issues:

Decision Trees

- A decision tree classifies samples "by asking questions successively": each **non-leaf** node corresponds to a **question**, each **leaf** corresponds to a final **prediction**.
- Decision tree learning partitions training data hierarchically.
- There are three main design issues:
  - Splitting criterion: which splits to choose?

Decision Trees

- A decision tree classifies samples “by asking questions successively”: each **non-leaf** node corresponds to a **question**, each **leaf** corresponds to a final **prediction**.
- Decision tree learning partitions training data hierarchically.
- There are three main design issues:
  - Splitting criterion: which splits to choose?
  - Stopping criterion: when to stop further growing of the tree?

Decision Trees

- A decision tree classifies samples “by asking questions successively”: each **non-leaf** node corresponds to a **question**, each **leaf** corresponds to a final **prediction**.
- Decision tree learning partitions training data hierarchically.
- There are three main design issues:
  - Splitting criterion: which splits to choose?
  - Stopping criterion: when to stop further growing of the tree?
  - Pruning: whether/how to collapse unnecessarily deep sub-trees?

Decision Trees

- A decision tree classifies samples “by asking questions successively”: each **non-leaf** node corresponds to a **question**, each **leaf** corresponds to a final **prediction**.
- Decision tree learning partitions training data hierarchically.
- There are three main design issues:
  - Splitting criterion: which splits to choose?
  - Stopping criterion: when to stop further growing of the tree?
  - Pruning: whether/how to collapse unnecessarily deep sub-trees?
- Two famous algorithms use decision trees:
  - Random forest
  - Gradient boosting

Random Forest

- **Several decision trees** are made, i.e., a random forest is a collection of multiple models (=ensemble learning).

Random Forest

- **Several decision trees** are made, i.e., a random forest is a collection of multiple models (=ensemble learning).
- Two sources of randomness (to reduce overfitting):

Random Forest

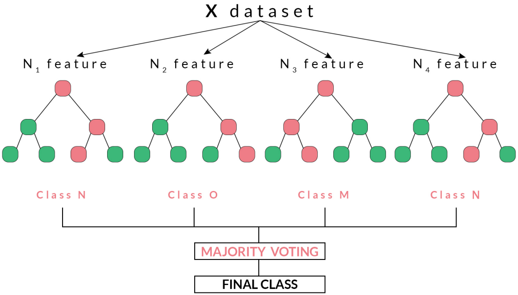
- **Several decision trees** are made, i.e., a random forest is a collection of multiple models (=ensemble learning).
- Two sources of randomness (to reduce overfitting):
  - Random subset of input data for fitting the individual trees.

Random Forest

- **Several decision trees** are made, i.e., a random forest is a collection of multiple models (=ensemble learning).
- Two sources of randomness (to reduce overfitting):
  - Random subset of input data for fitting the individual trees.
  - Features in each decision tree are chosen randomly.

Random Forest

- **Several decision trees** are made, i.e., a random forest is a collection of multiple models (=ensemble learning).
- Two sources of randomness (to reduce overfitting):
  - Random subset of input data for fitting the individual trees.
  - Features in each decision tree are chosen randomly.
- Final decision by, e.g., majority vote.



Picture taken from: https://blog.quantinsti.com