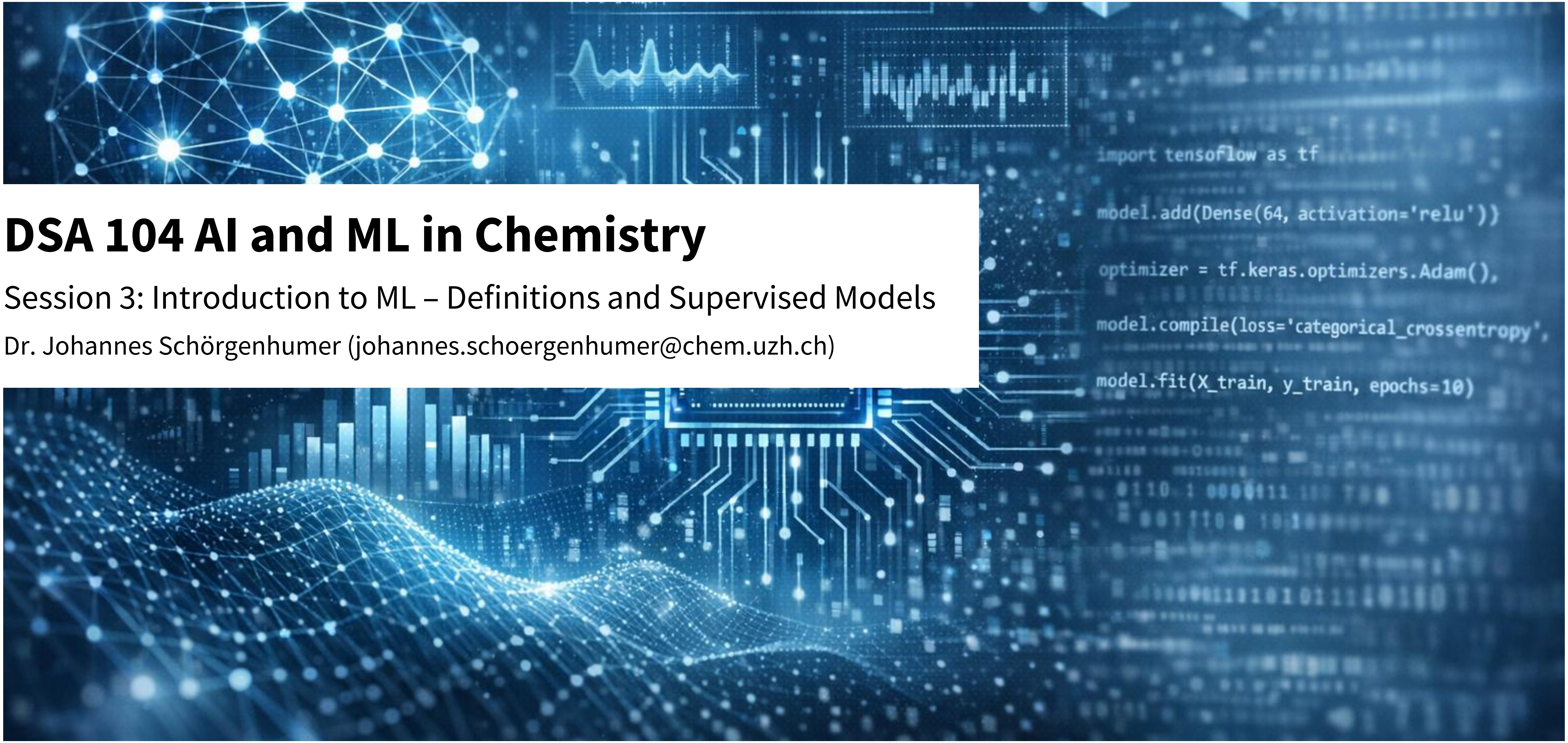
Universität Zürich[UZH]

Department of Chemistry

DSA 104 AI and ML in Chemistry

Session 3: Introduction to ML – Definitions and Supervised Models

Dr. Johannes Schörgenhumer (johannes.schoergenhumer@chem.uzh.ch)



AI – ML – DL: Definition and Distinction?

AI (Artificial Intelligence):

— A broad “umbrella term” for systems behaving intelligently

— Not all AI involves data!

— Historically includes e.g. symbolic reasoning, rule-based systems, logic systems (e.g. Classic chess engines using search, Logic-based planning systems)

ML (Machine Learning):

— Learn patterns from data to improve performance on a task

— Replaces explicit programming with learning from examples, **rules inferred from data**

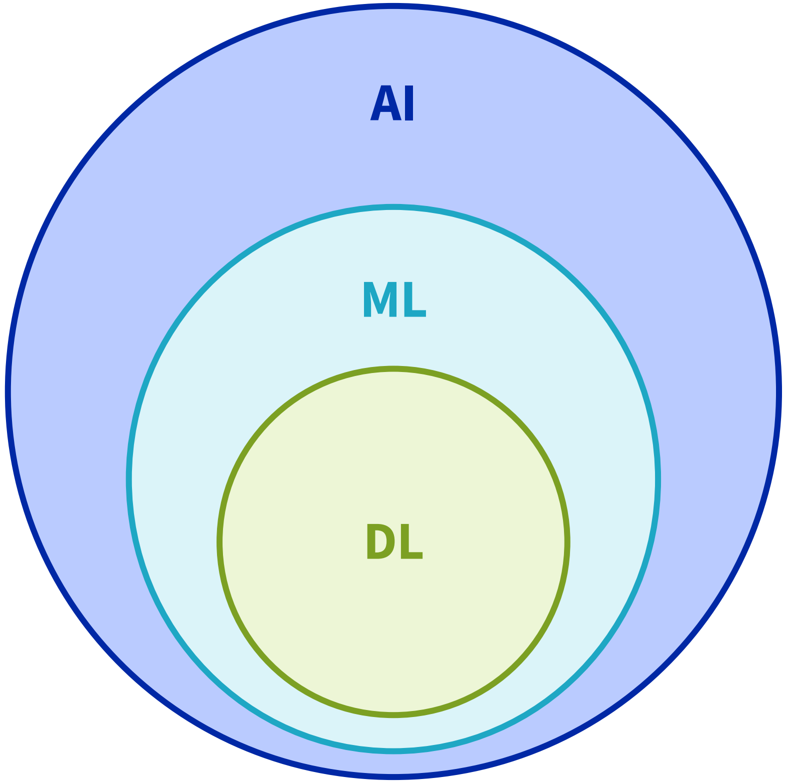
— Includes e.g. Regressors, Random Forest, GradientBoost, DeepLearning!

DL (Deep learning):

— Learn hierarchical representations using neural networks with multiple layers.

— Instead of hand-designing features, DL learns features automatically.

— Subset of ML, extremely powerful for complex data (high data demand)



Exercise: Classification quiz – AI, ML or DL?

AI 1. A thermostat

AI 2. Siri or Alexa recognizing your voice command

DL 3. ChatGPT

AI 4. A chatbot responding to customer service queries using rules

ML 5. Netflix recommender

DL 6. Generating realistic images using GANs (Generative Adversarial Networks)

AI 7. Expert system for medical diagnosis

DL 8. CNN for image recognition

ML 9. Detecting spam emails using a classifier trained on labeled emails

AI 10. A chess computer

DL 11. Self-driving car detecting pedestrians

Types of ML

Supervised Learning:

— Labelled data

— Direct feedback

— Predict outcome

Unsupervised Learning:

— No labels/targets

— No feedback

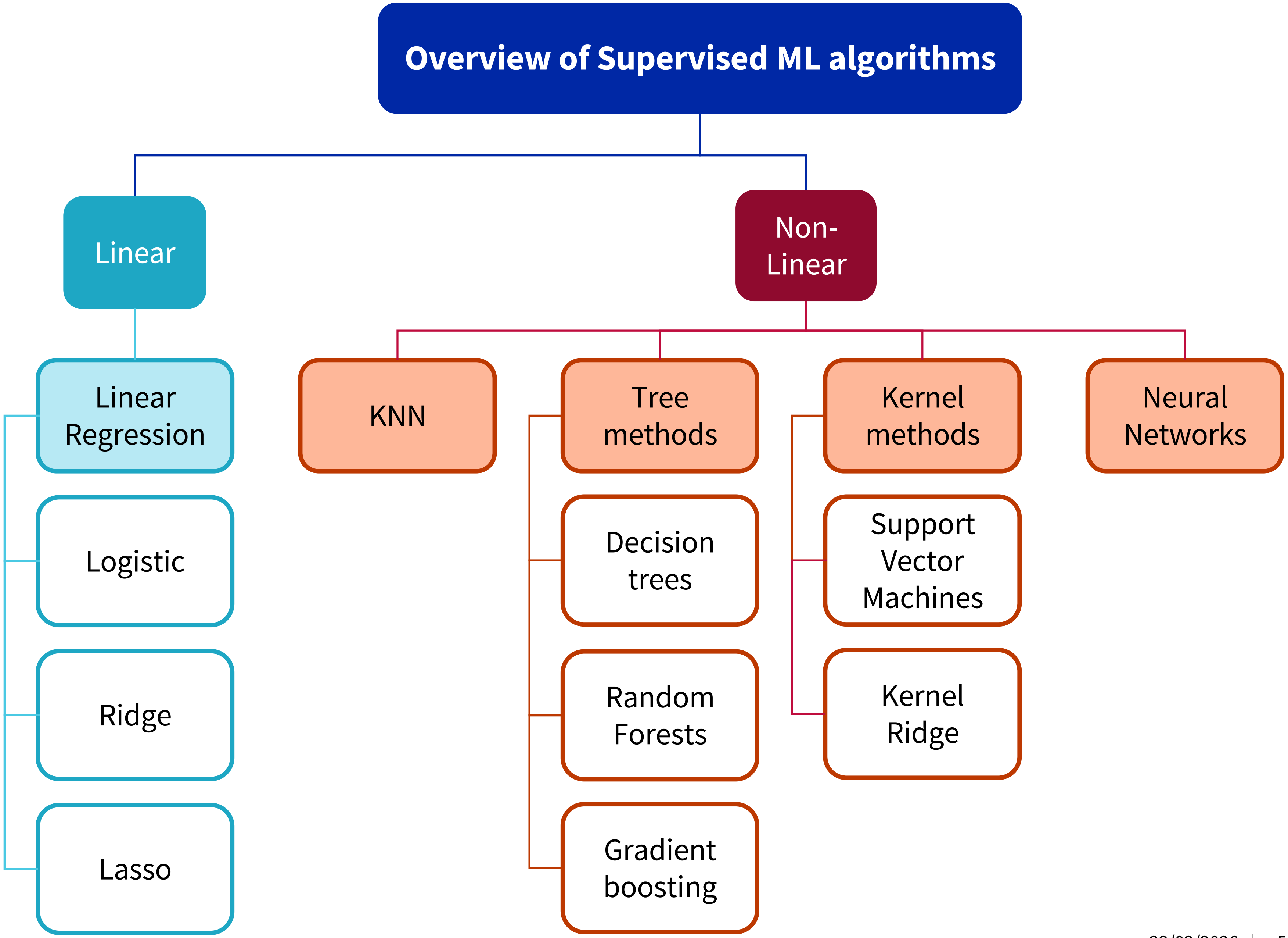
— Find hidden structure in data

Reinforcement learning:

— Decision process

— Reward system

— Learn series of actions



Supervised ML

Learning from **input values** and corresponding **target values**:

— E.g. image + object type, DNA sequence + phenotype, . . .

Typical usage: **predictive modelling**

— **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.

— Classic predictions:

  — **Classification**: target value is class **label**

  — **Regression**: target value is **numerical** value

Example:

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

Salmon

Sea bass

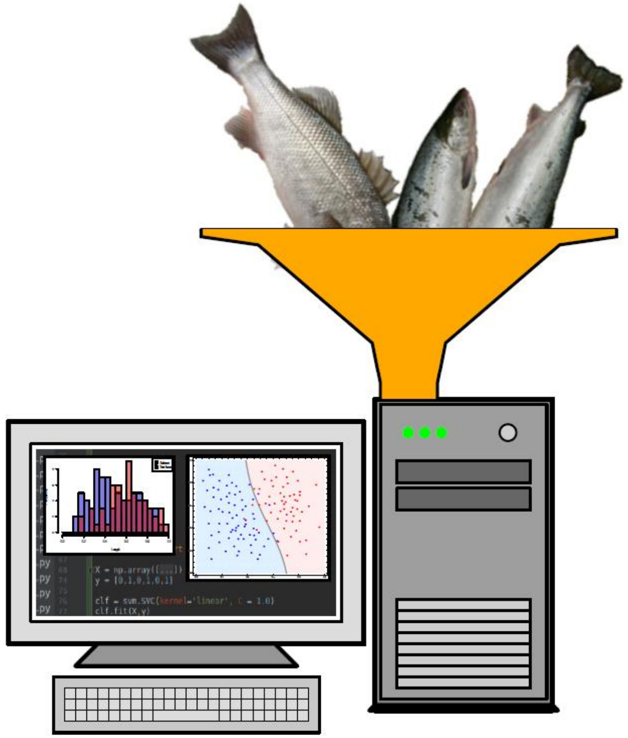
Example from R. O. Duda, P. E. Hart, and D. G. Stork. Pattern Classification. 2nd edition. John Wiley & Sons, 2001. ISBN 0-471-05669-3.

Terminology: model

- **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

Example:

- 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**: train a **model** to distinguish between salmons and sea bass.

Adapted from A. Schörgenhuber, *Hands-on AI I*, Lecture materials, **2023**, JKU Linz.

Terminology: data

We can represent an object by a **vector *x*** of feature values (=**feature vector**) of **length *d*** and **label *y*:**

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

— 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”)

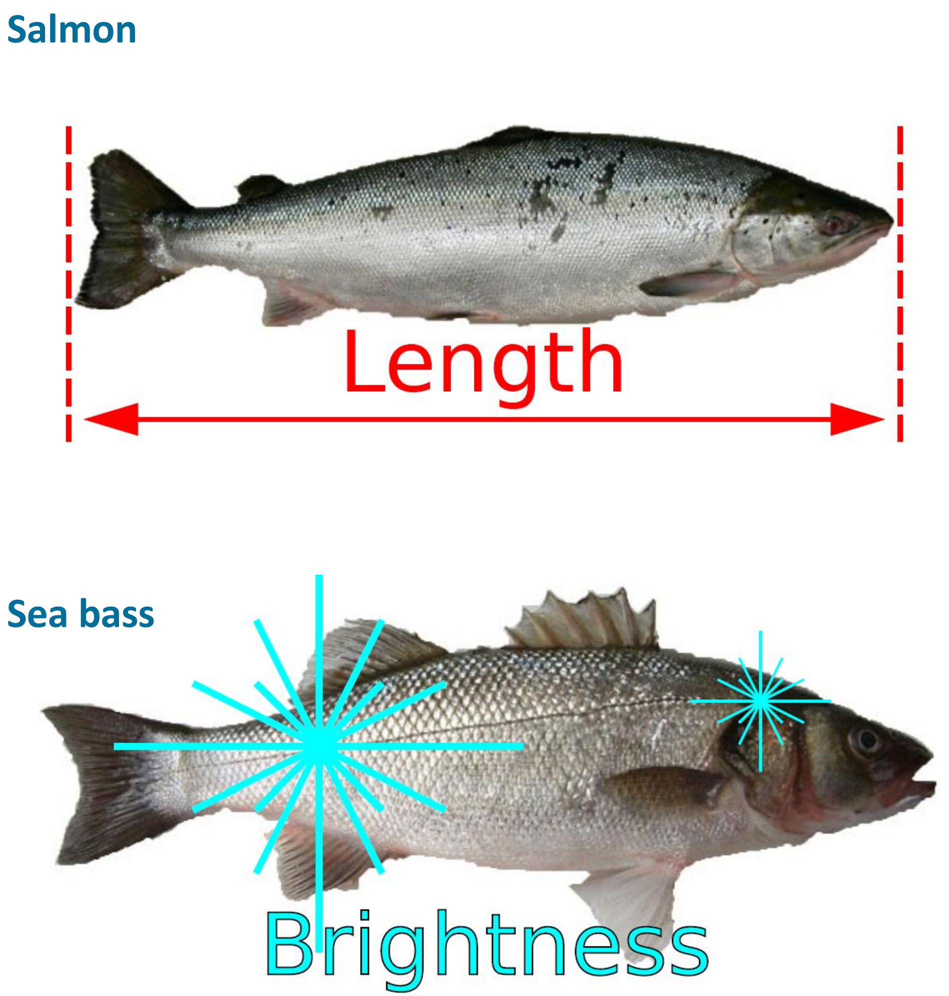
— An object described by one feature vector and one label is referred to as **sample**: (x, y).

— n samples each with feature vectors $x_1, \ldots, x_n$ of length d and label $y_1, \ldots, y_n$, thus, **matrix of feature vectors *X*** and the labels in a corresponding **labels vector *y***:

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

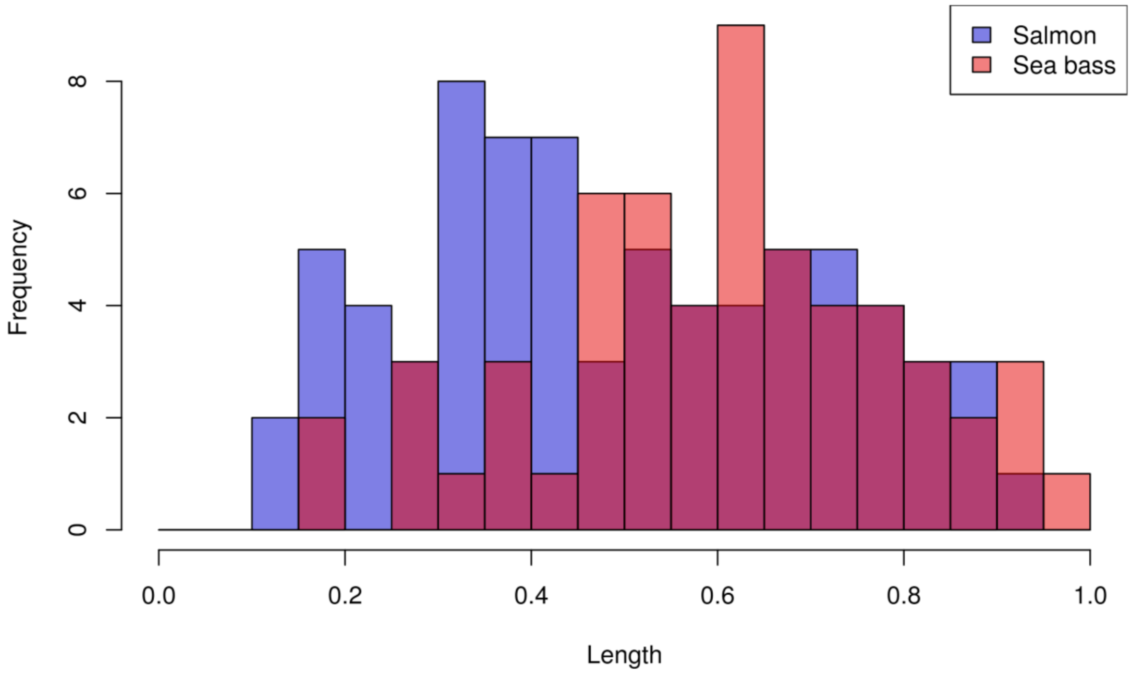
— Our labeled data is thus described by: (X, y).

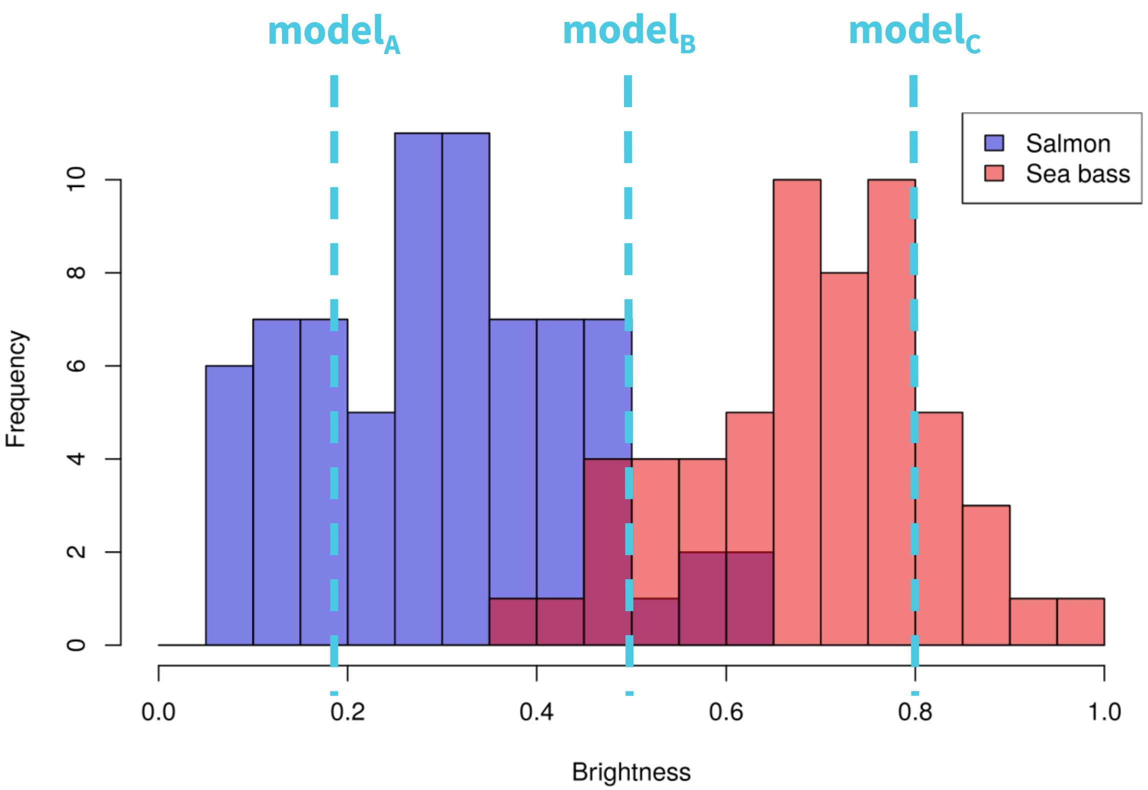
Example:



Adapted from A. Schörgenhumer, *Hands-on AI I*, Lecture materials, **2023**, JKU Linz.

Fish example: Look at the data





— Brightness looks more useful for fish classification.

— 3 different models based on brightness threshold:

— model_A: brightness < 0.18 → Salmon

— model_B: brightness < 0.5 → Salmon

— model_C: brightness < 0.8 → Salmon

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?

— **Generalisation error / risk**

Adapted from A. Schörgenhumer, *Hands-on AI I*, Lecture materials, **2023**, JKU Linz.

Loss functions

— Assume we have a model $\boldsymbol{g}$ with parameter $\boldsymbol{w}$:

— $\boldsymbol{g}(\boldsymbol{x}; \boldsymbol{w})$ maps an input vector x to an output value $\hat{\boldsymbol{y}}$

— $\hat{\boldsymbol{y}}$ needs to be as close as possible to the true target value $\boldsymbol{y}$

— We can use a **loss function** $\boldsymbol{L}$ to measure how close the prediction is to the true target for a sample with (x, y):

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

— Quantifies the error: Lower loss = better model performance

— Adjust the parameters/weights $\boldsymbol{w}$ to reduce loss

Loss functions: Examples

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

— Zero-one loss: $L_{\mathrm{ZO}}\big(y, g(x; w)\big) = \begin{cases} 0 & y = g(x; w) \\ 1 & y \neq g(x; w) \end{cases}$

Classification errors

— MSE: $L_{\mathrm{MSE}}\big(y, g(x; w)\big) = \frac{1}{n}\sum_{i=1}^{n}(y_i - g(x_i; w))^2$

Quite common, heavily punishes outliers (as compared to MAE), not for robustness

— Many other loss functions available

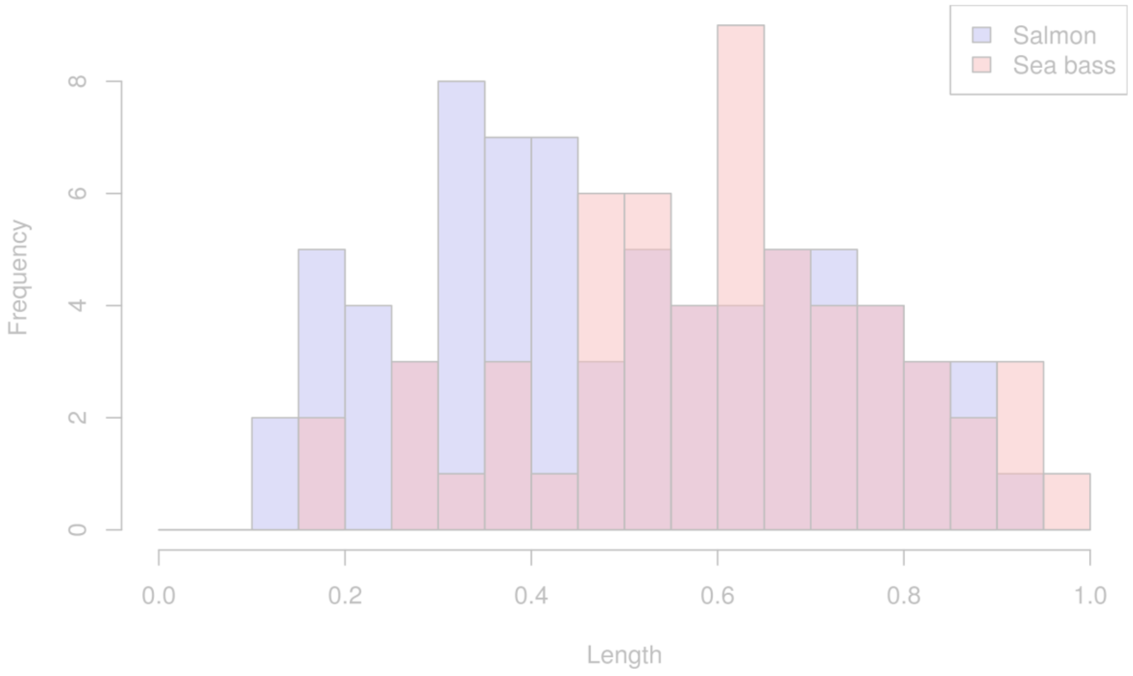
— **Different loss functions are needed for different data, tasks and model classes, e.g.**

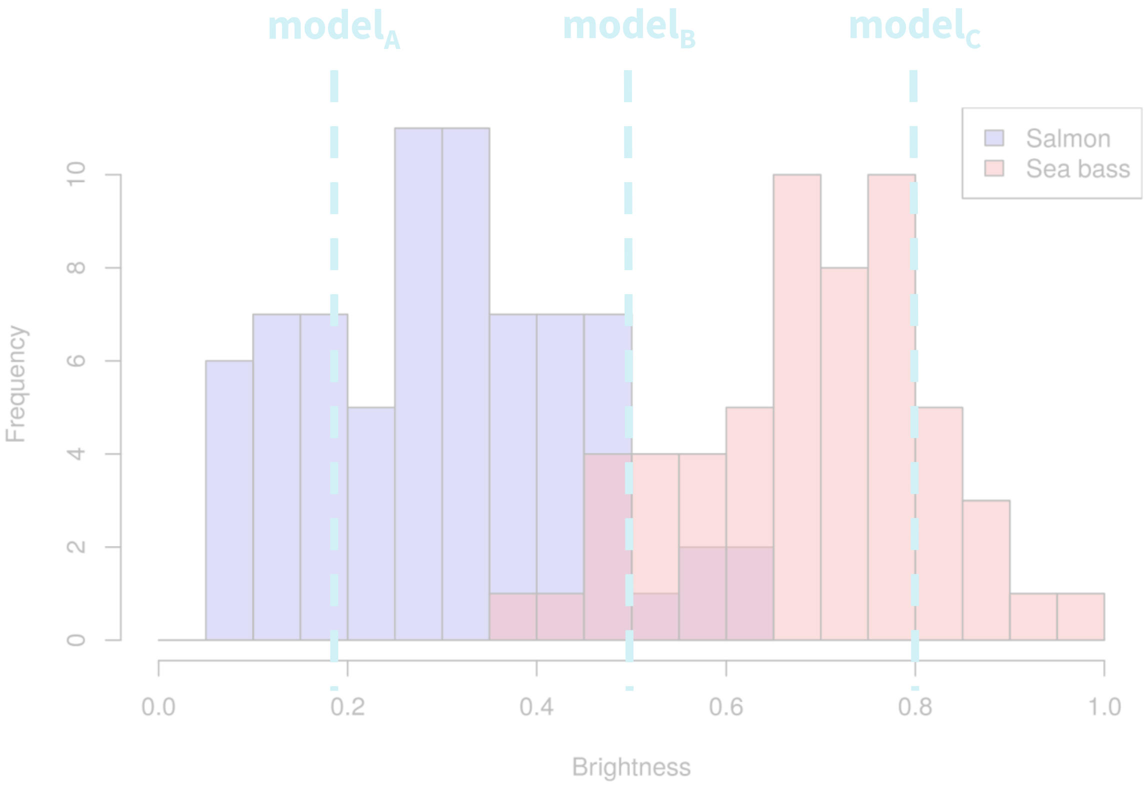
  — Regression: Mean Squared Error (MSE), Mean Absolute Error (MAE)

  — Classification: Cross-Entropy Loss (log loss), Hinge Loss (SVMs)

  — Ranking/Boosting: Exponential loss (AdaBoost), Logistic loss (Gradient Boosting)

Fish example: Look at the data





- Brightness looks more useful for fish classification.
- 3 different models based on brightness threshold:
  - model$_A$: brightness < 0.18 → Salmon
  - model$_B$: brightness < 0.5 → Salmon
  - model$_C$: brightness < 0.8 → Salmon

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?
  - **Generalisation error / risk**

Adapted from A. Schörgenhumer, *Hands-on AI I*, Lecture materials, **2023**, JKU Linz.

Generalisation error / risk

The **generalisation error** or **risk** is the expected loss on **future / unseen data** for a given model $\boldsymbol{g}(\boldsymbol{x}; \boldsymbol{w})$

$$R\big(g(x;w)\big) = \iint_{Xy} L(y, g(x;w) \cdot p(x,y) \cdot dydx$$

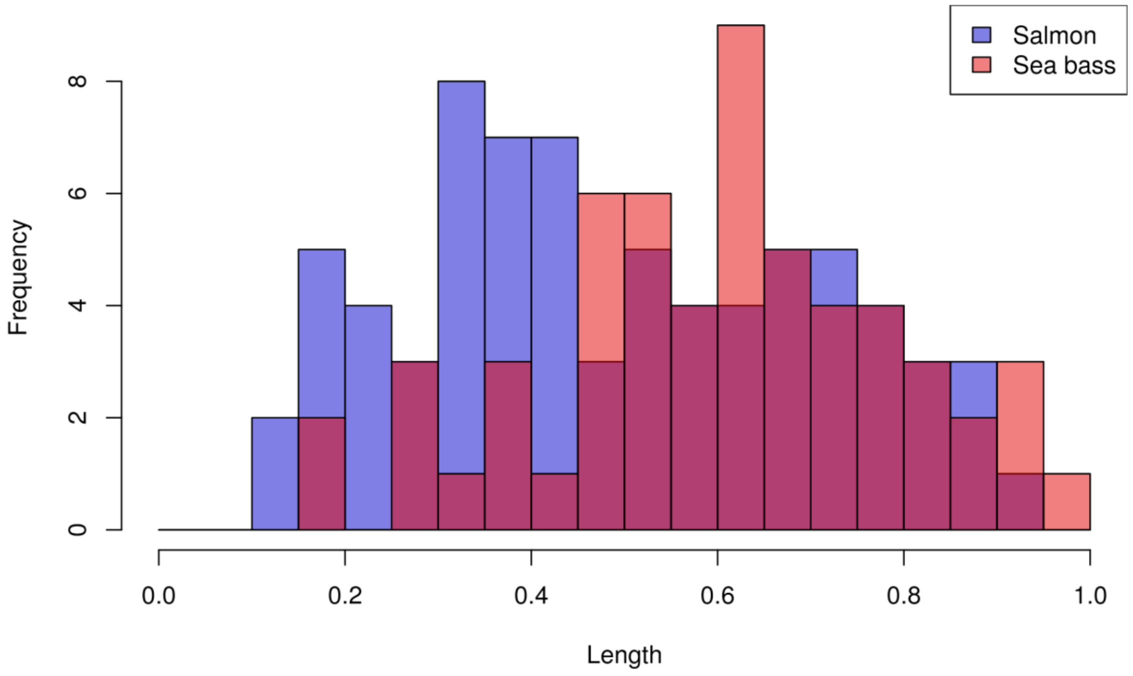
$R\big(g(x;w)\big)$ denotes the **expected loss** for input x, and $p(x,y)$ is the joint probability distribution for x and y.

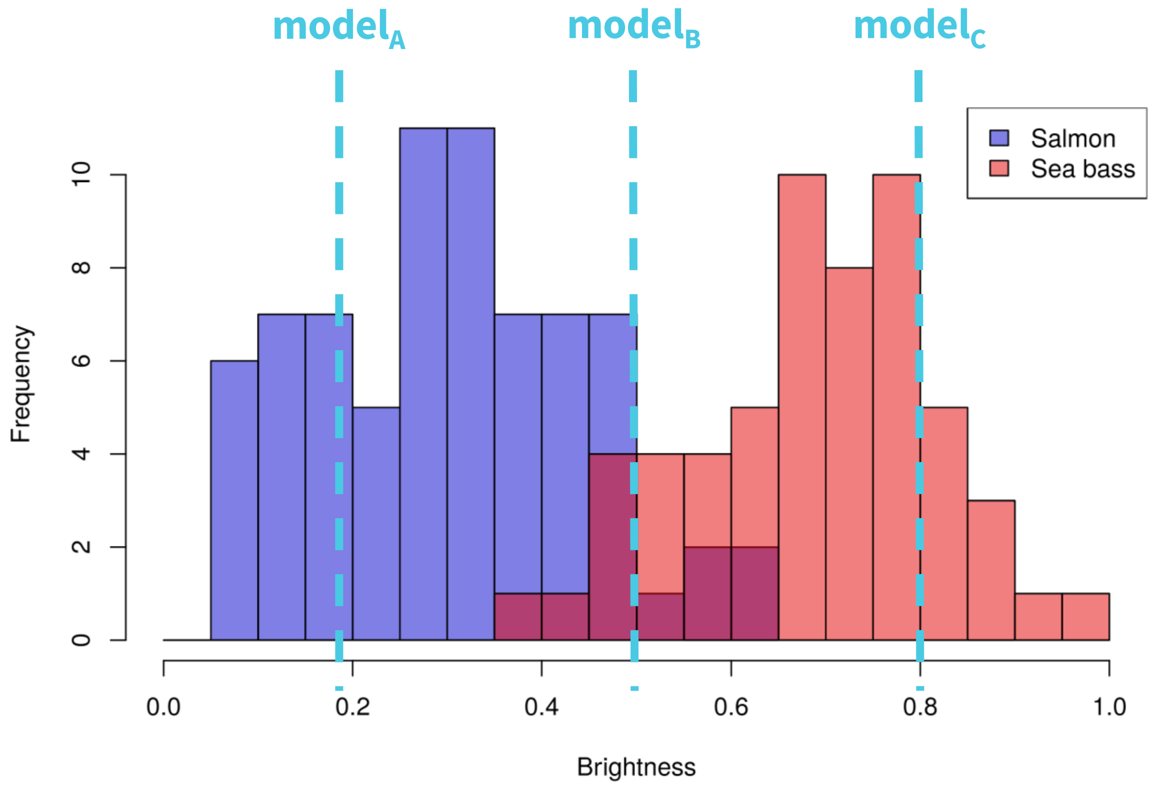
- In practice, joint probability distribution $p(x,y)$ unknown:
- **Estimate** the generalisation error: **Empirical Risk Minimisation (ERM)**
  - Use data set (X, y) instead of $p(x,y)$:

$$R_{\mathrm{E}}(g(x;w), (X,y)) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, g(x_i;w)$$

  - Law of large numbers: for $\mathrm{n} \to \infty$: $R_{\mathrm{E}}\big(g(x;w)\big) \to R\big(g(x;w)\big)$
  - Minimisation: $g_{\mathrm{ERM}} = \arg\min_{g \in G} R_{\mathrm{E}}(g)$

Fish example

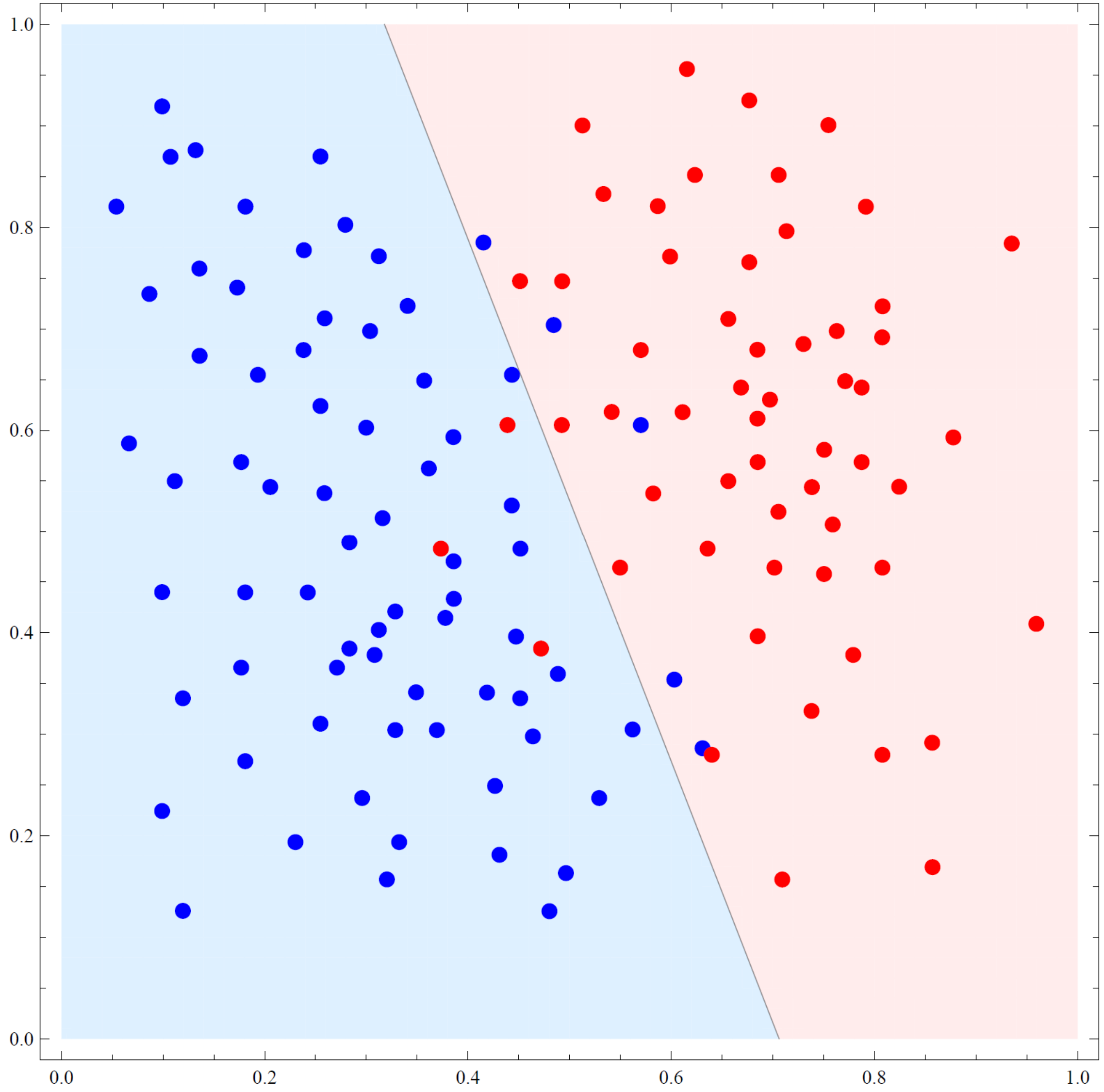




ERM optimises model only based on training data.
Individual features (especially length) do not separate classes super well
Combine our features and use a different model class

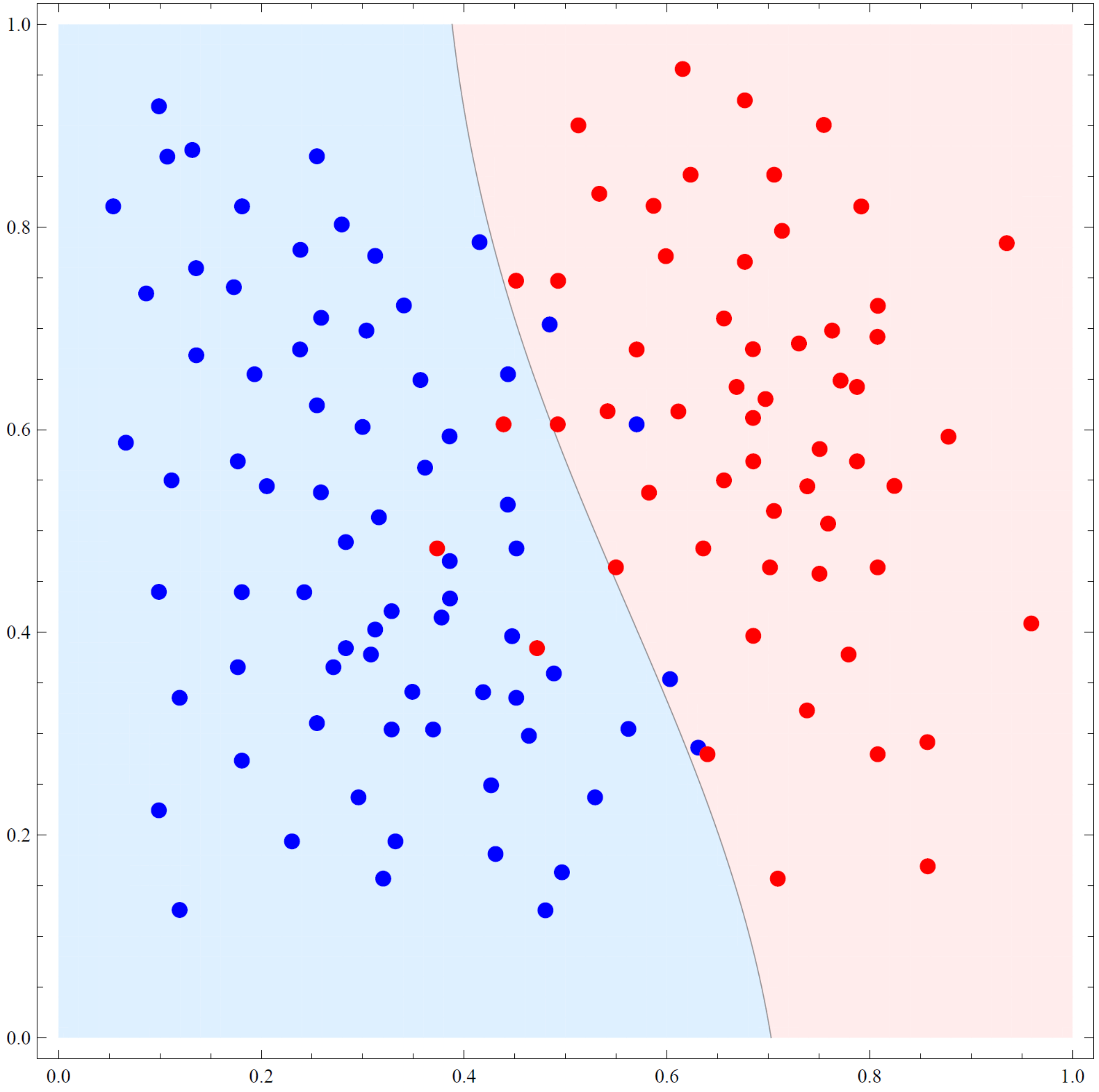
Adapted from A. Schörgenhumer, *Hands-on AI I*, Lecture materials, **2023**, JKU Linz.

Fish example – linear separation:



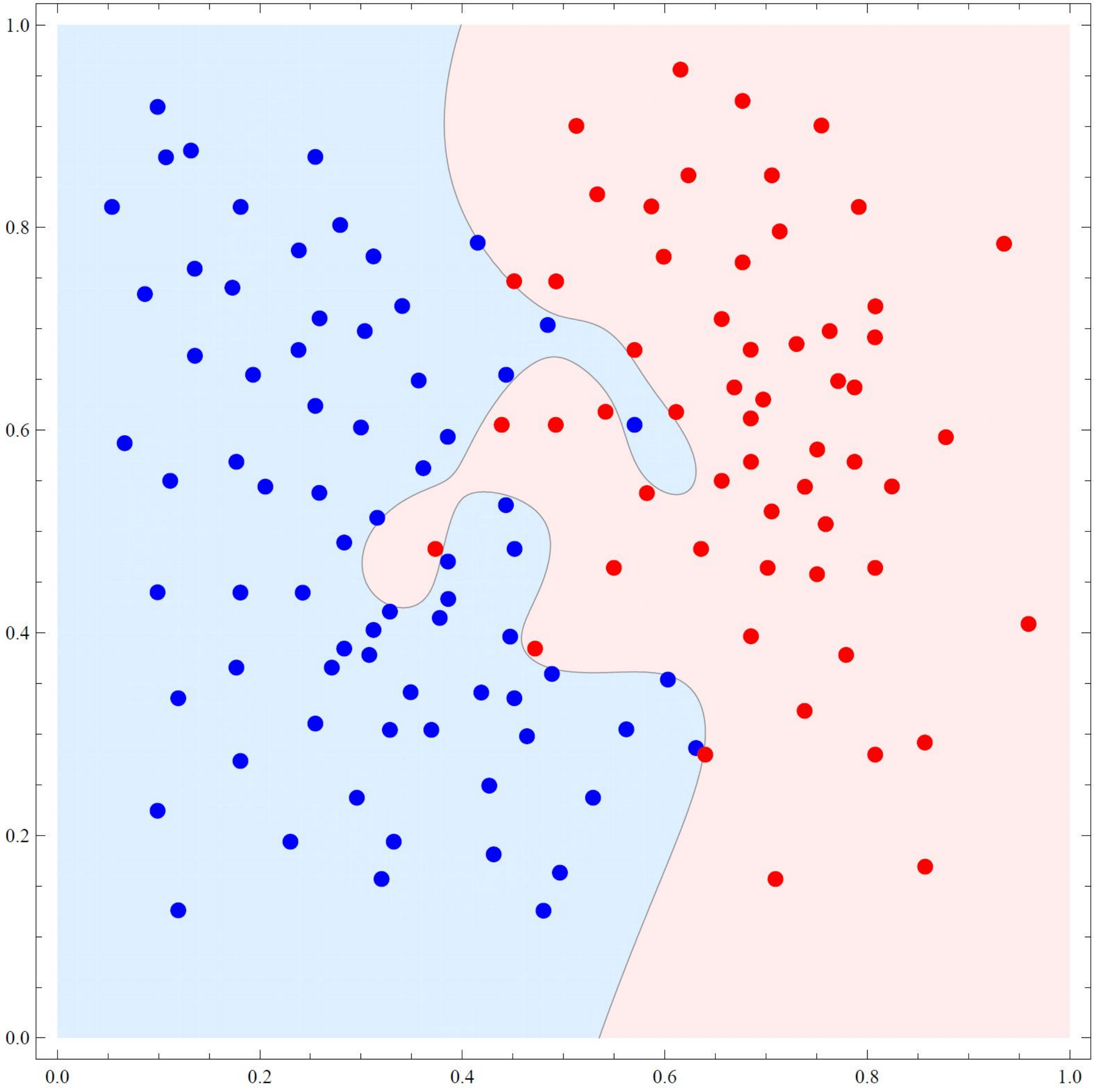
Adapted from A. Schörgenhumer, *Hands-on AI I*, Lecture materials, **2023**, JKU Linz.

Fish example – mildly non-linear separation:



Adapted from A. Schörgenhumer, *Hands-on AI I*, Lecture materials, **2023**, JKU Linz.

Fish example – highly non-linear separation:



Overfitting?

Adapted from A. Schörgenhumer, *Hands-on AI I*, Lecture materials, **2023**, JKU Linz.

Better risk estimation

Test Set Method

— Assumption: Samples are **independently and identically distributed**

— Split data set of n samples into **two non-overlapping subsets**:

  — **Training set**: for ERM (i.e. to optimise parameters), typically 80% of n

  — **Test set**: use to estimate the risk (test data = approximation of future, unseen data), typically 20% of n

— RE on the test set identifies overfitting!

Better risk estimation

Validation Set

— Hyperparameters often changed, model will be retrained and evaluated (**hyperparameter tuning**)

— May lead to overfitting:

  — Hyperparameter tuning based on test set evaluation results → Test set indirectly used for training

  — Not non-overlapping subsets - estimation of the generalisation error compromised

— 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.

More on better RE and Generalisation:

— Good generalisation requires combination of ERM with…

  — Regularisation (e.g. add penalties, limit parameters): $L'(y, \hat{y}) = L(y, \hat{y}) + \lambda \cdot \Omega(\hat{y})$

  — Appropriate model capacity

  — **Enough data!!**

…where $\Omega(\hat{y})$ is a **regularisation function**, e.g.

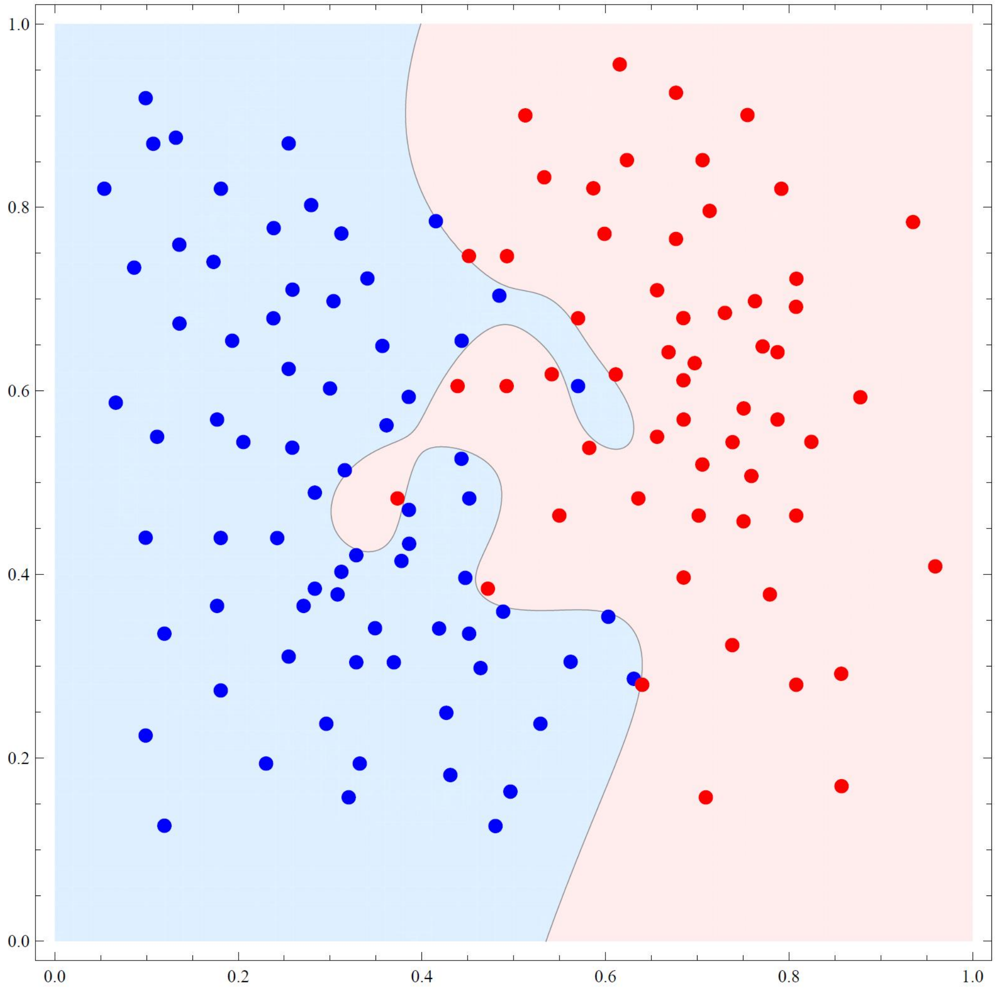
$$\Omega(\hat{y}) = ||\theta||_2^2 = w_1^2 + w_2^2 + \ ... + w_j^2$$

…with θ being a the parameter vector of j parameters w

Fish example – best fit

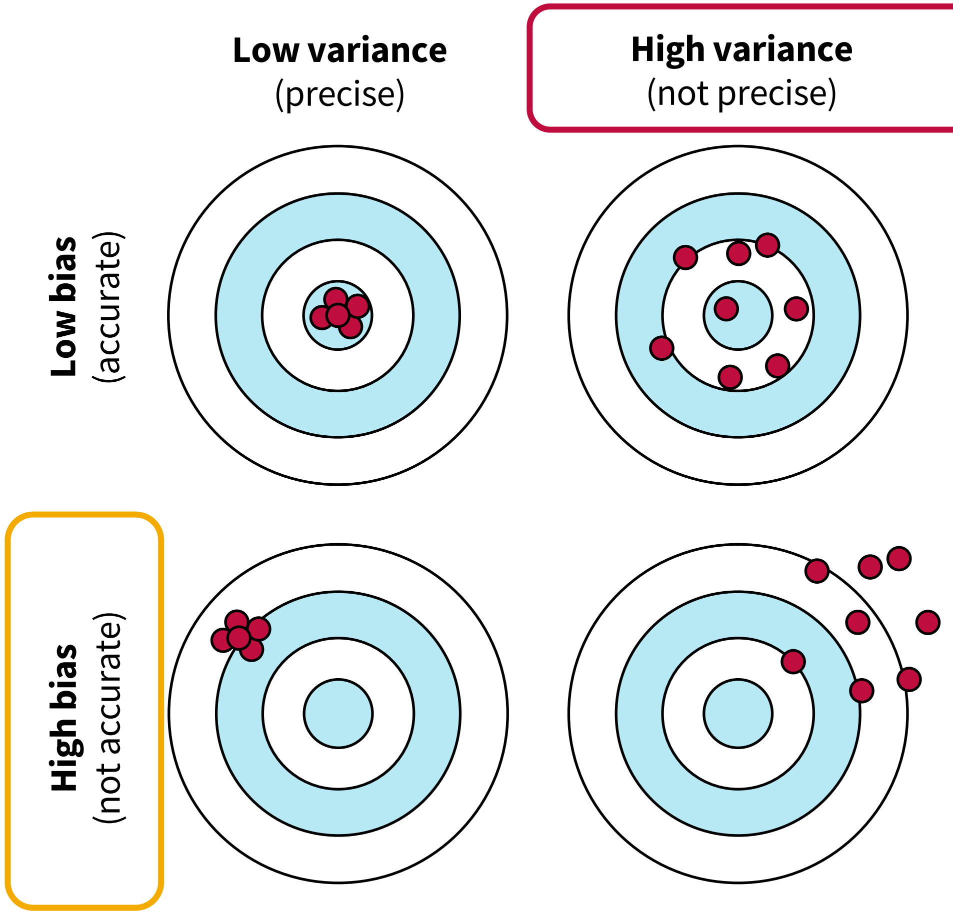
- Now, we can use **ERM** to optimise a model on our **training data set** (optionally, including a validation set)
- A held-out **test set** will allow us to get an estimate about the performance on future data (i.e. **generalisation**)
- If overfitting is detected, we can reduce the model complexity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

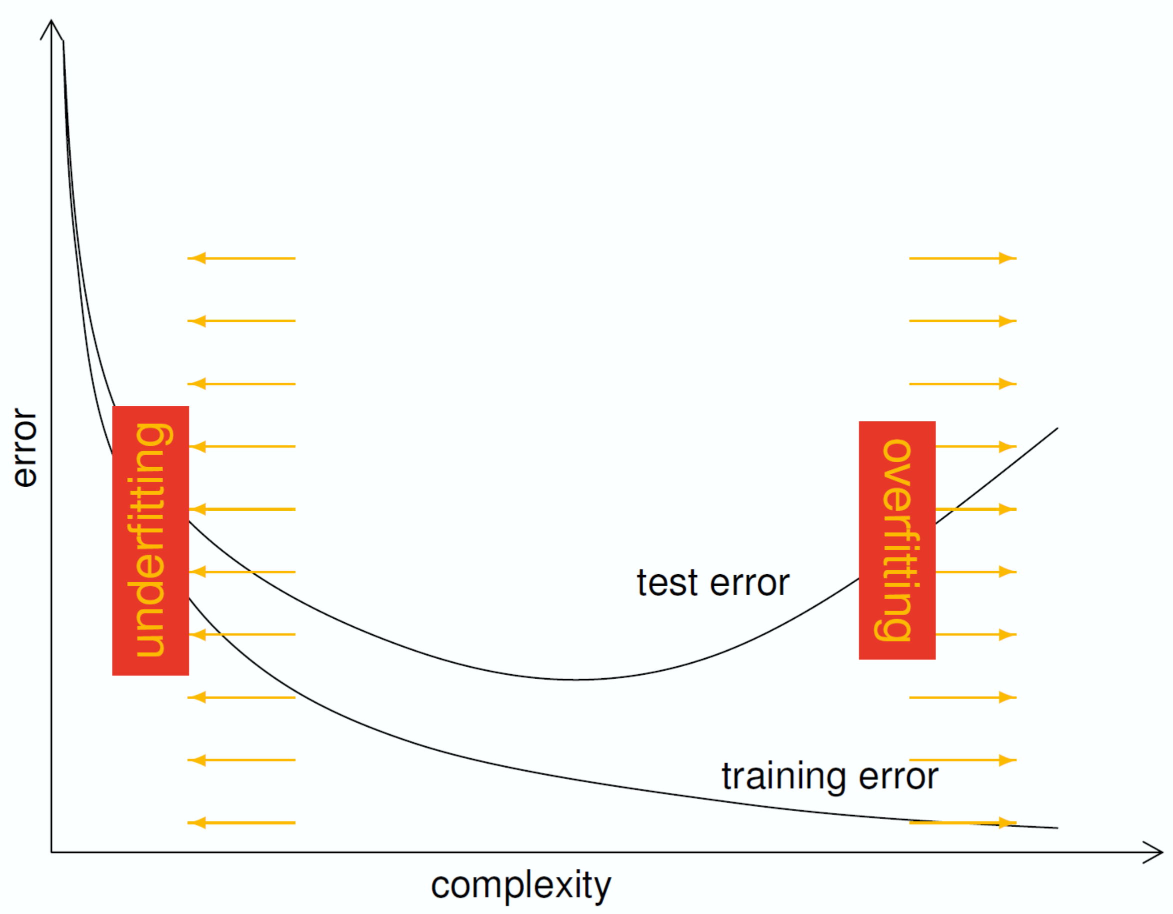
- But what if the model is **too simple**? → Underfitting!



Adapted from A. Schörgenhumer, *Hands-on AI I*, Lecture materials, **2023**, JKU Linz.

Bias-variance tradeoff



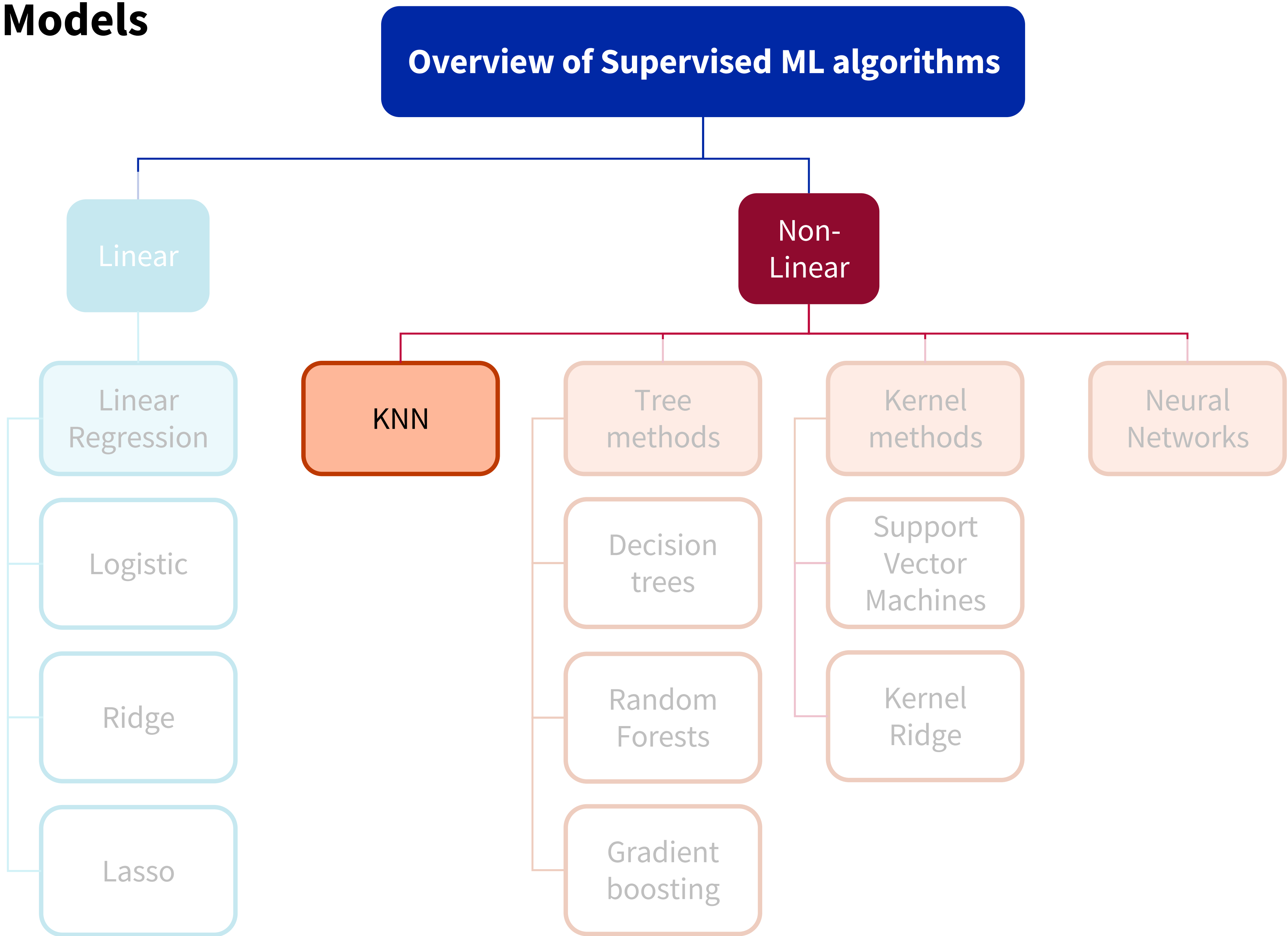


- **Underfitting** (high bias): The model is too coarse to fit training data and also too coarse to fit test data. The model complexity is too low.
- **Overfitting** (high variance): The model fits (too) well to training data but not to future/test data. The model complexity is too high.

Exercise: Over- or Underfitting? Or just right?

1. A linear regression model is used to predict house prices, but it consistently misses trends and has high error on both training and test sets.	**Underfitting**
2. A decision tree model perfectly predicts the training data but performs worse on unseen data.	**Overfitting**
3. A model achieves similar accuracy on both training and test data.	**Well-fitted**
4. Using a polynomial regression of degree 5 on a small dataset, the model perfectly fits training points but fluctuates wildly on new data.	**Overfitting**
5. A linear model predicts sales poorly because it cannot capture the underlying pattern.	**Underfitting**
6. A neural network achieves 98% accuracy on the training set, 90% on the test set, but shows high variance between predictions on different validation batches	**Overfitting**

Intro to ML Models



k-Nearest Neighbors Classifier (KNN)

Assume we have a labeled data set (X, y) and a **distance measure on the input space**.

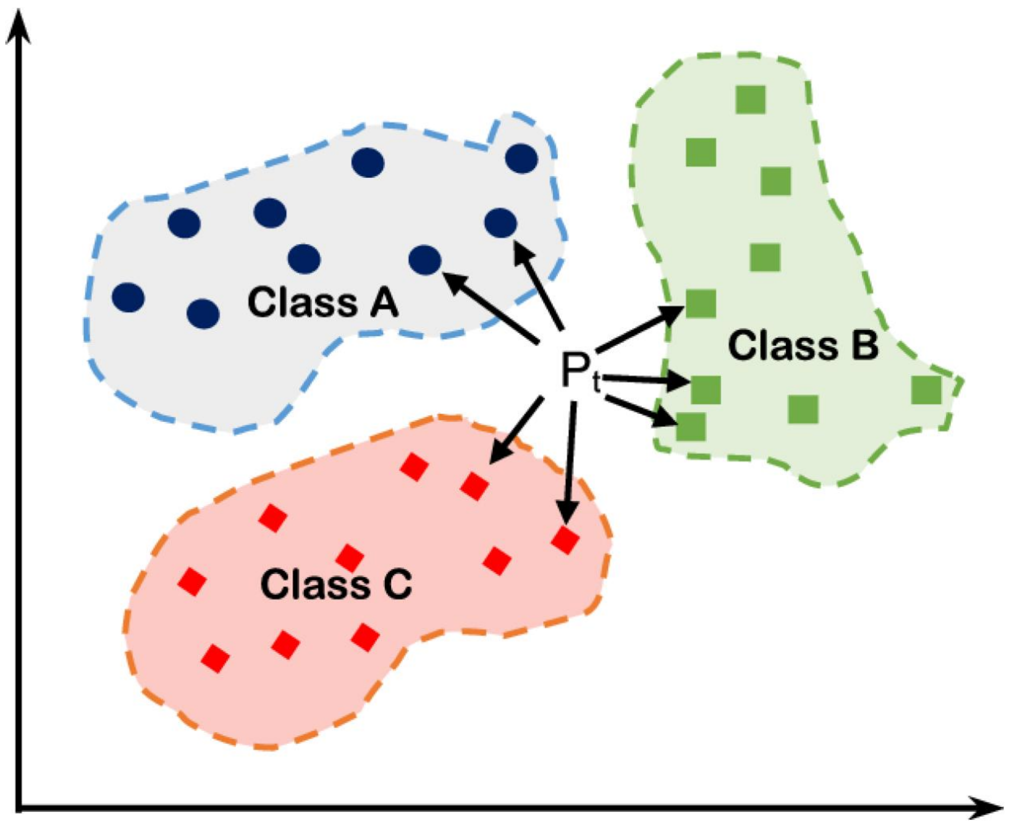
— The k-**nearest neighbors classifier** is defined as follows:

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

— For k = 1: **nearest neighbor classifier**:

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

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

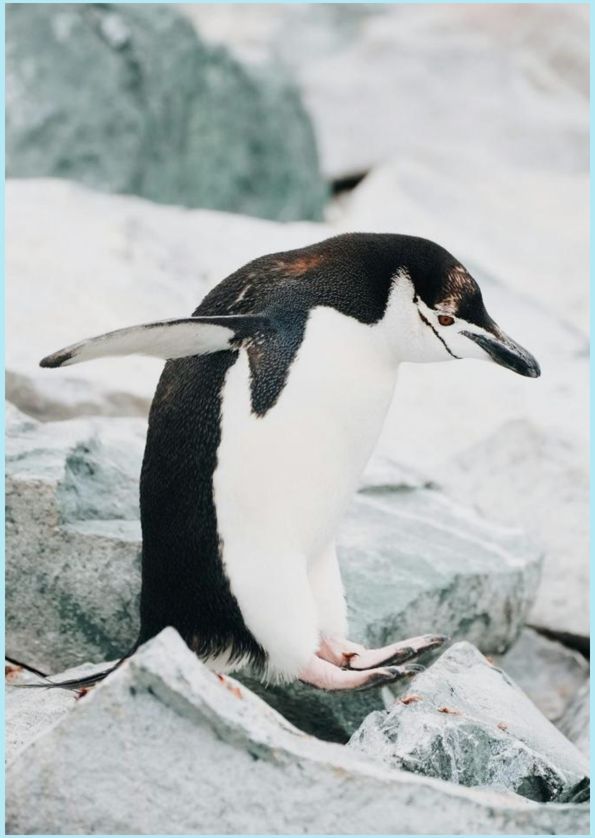


Atallah, D.M., Badawy, M., El-Sayed, A. *et al.* Predicting kidney transplantation outcome based on hybrid feature selection and KNN classifier. *Multimed Tools Appl* **2019**, *78*, 20383–20407. https://doi.org/10.1007/s11042-019-7370-5

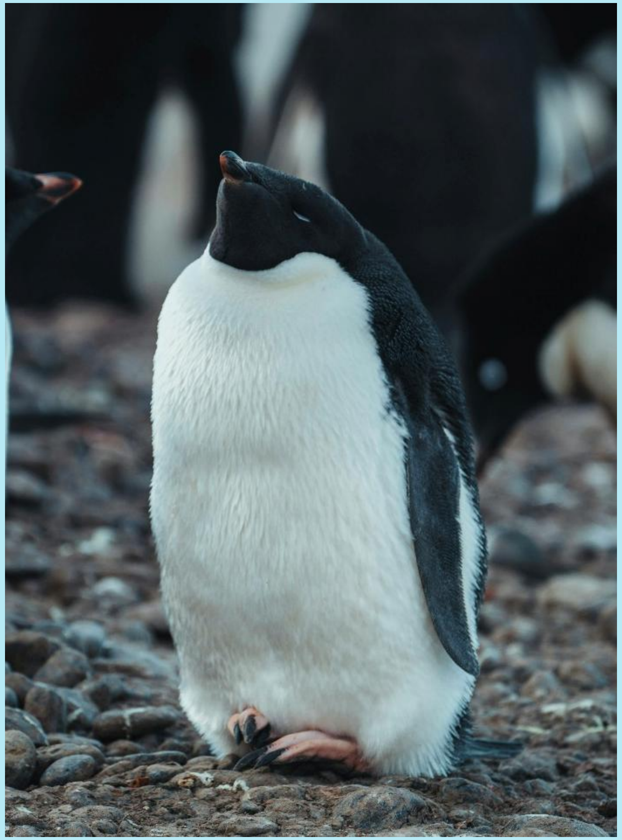
Classification model KNN

— Explore the impact of different numbers for K (*KNN-penguins-decisionboundaries*)

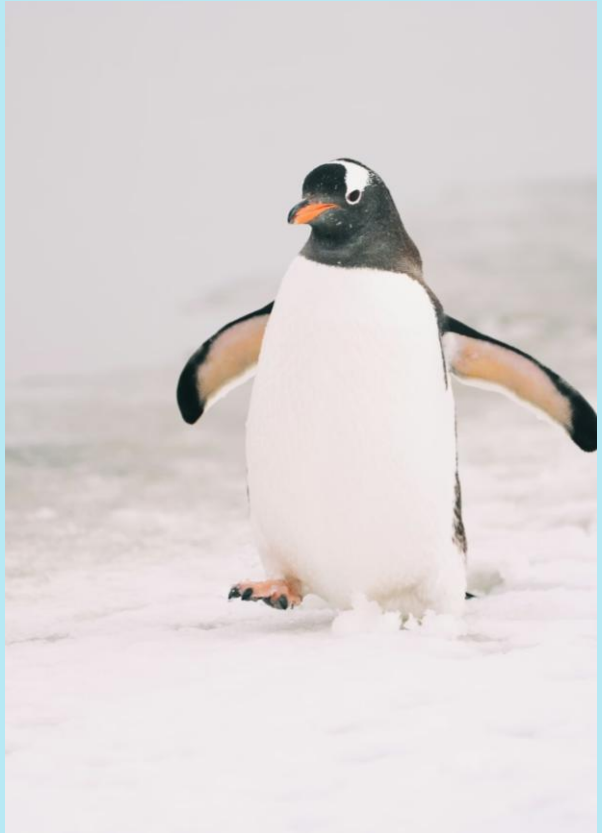
— Use a KNN Classifier to predict different penguin species for unknown data! (*KNN-penguins*)



Chinstrap



Adélie



Gentoo

Your penguin?

?