Data Science Academy

# Data Science for Business – Becoming a Data Science Expert (D)

Pilot Presentation:
for participants of and use in the pilot only

01.09.2021

Boehringer Ingelheim

## Agenda

# Agenda week one

## Schedule week one

| Week 1 | | | |
|---|---|---|---|
| | Day 1<br>Tuesday, 31.08.2021 | | Day 2<br>Wednesday, 01.09.2021 |
| Start:<br>12:00 | Introduction | Start:<br>12:00 | Recap |
| | 1 – Recap Basic Machine Learning and Python | | 3 – Model Evaluation |
| 14:00 –<br>15:00 | Break | 14:00 –<br>15:00 | Break |
| | 2 – Complex Models | | 4 – Hyperparameters |
| End:<br>18:00 | Q&A and Feedback | End:<br>18:00 | Q&A and Feedback |

We will also have several short coffee breaks in between.

## Feedback for pilot training

We aim to provide a great training experience for you and are looking forward to receiving your feedback!

You will have three different ways to give us your feedback on each training day:

1. We will have an **anonymized** feedback collection **after the last session** of each day per **Myforms.**
2. We will have an **open feedback round and discussion** at the **end of each training day.**
3. Please also **take notes** regarding your ideas during the sessions: **locally or via the Mural Board** which you can reach via LINK.

## Module 3
## **Model Evaluation**

# Agenda week one

# Measure how well the model is performing

| Level | Example | Area | Advantages | Disadvantages | Comment |
|---|---|---|---|---|---|
| **1. Loss function** | • Mean squared error loss<br>• Cross entropy loss | **Model training** | • Easy to compute the derivative<br>• Evaluates goodness of fit | • Difficult to interpret | • How well do the current model parameters work on the training set? |
| **2. Evaluation metrics** | • Mean absolute error (MAE)<br>• Accuracy | **Data Science** | • Easy to understand<br>• General applicability | • Influence on the business is not apparent | • How well does the model generalize? |
| **3. Business metrics** | • Return on investments (ROI)<br>• Click through rate (CRT) | **Management** | • Shows influence on business | • Has to be defined<br>• Strongly depends on definition | • How much money will we gain/loose from applying this model in production? |

# 1. How to measure performance of model

**A loss function $L(y, \hat{y})$ describes how much all predictions $\hat{y}$ are away from all true labels $y$**

$$L(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} l(y_i, \hat{y}_i)$$

Given a series of actual answer $y = (y_1, \ldots, y_n)$ and predictions $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_n)$

**Define the function $l(y_i, \widehat{y_i})$ as how much a prediction $\widehat{y_i}$ is away from the actual answer $y_i$**

- **Regression:** Squared error

$$l(y_i, \hat{y}_i) = (y_i - \widehat{y_i})^2$$

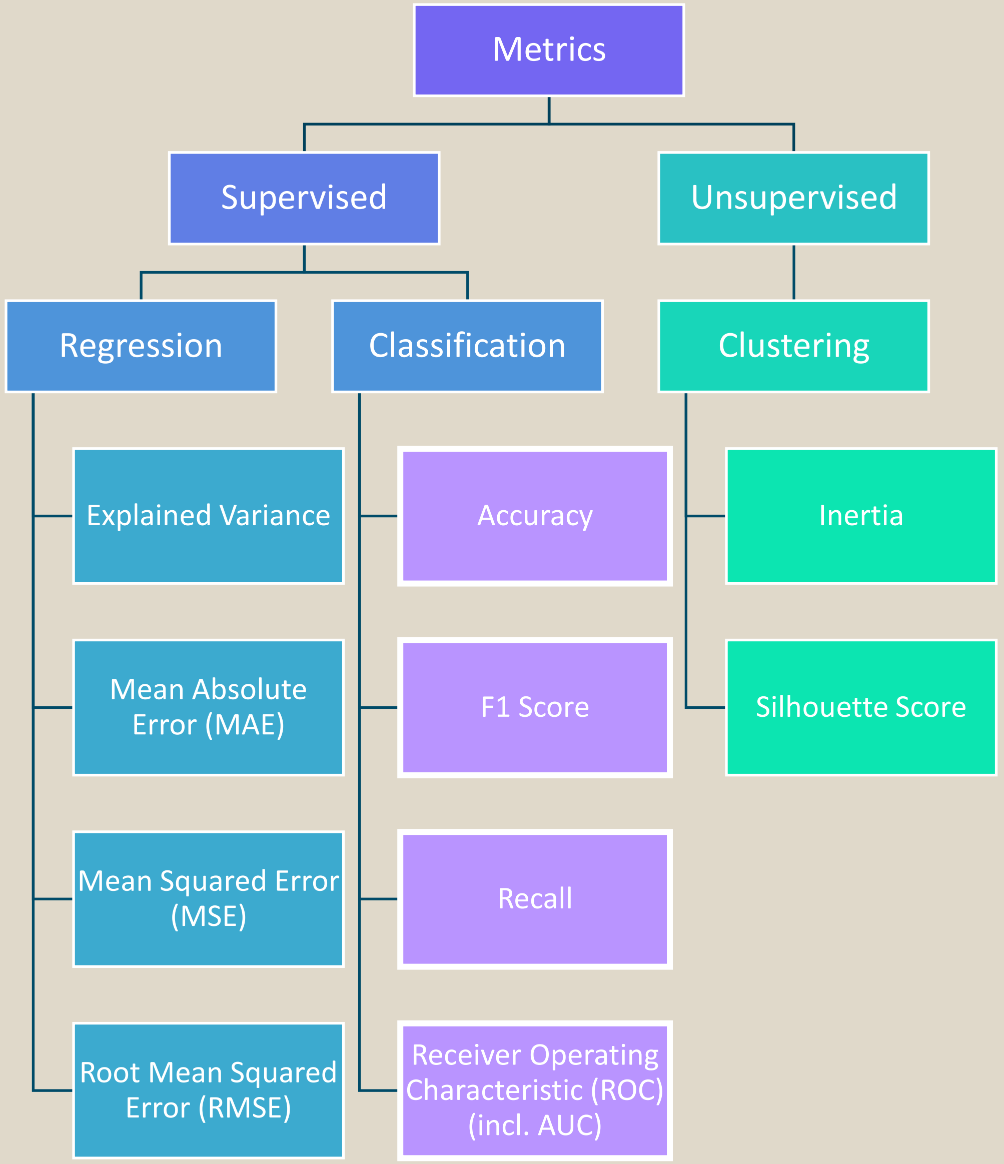
- **Boolean Classification:** Binary cross-entropy loss

$$l(y_i, \widehat{y_i}) = -[y_i \log(\widehat{y_i}) + (1 - y_i)\log(1 - \widehat{y_i})]$$

- **Multiclass Classification:** Multi-class cross-entropy loss

$$l(\text{class } c, \hat{y}_i) = -\log(\widehat{y_i}[c])$$

# Selected evaluation metrics



# Accuracy

**The accuracy metric quantifies the performance as the ratio of samples we classified correctly.**

$$accuracy = \frac{\#correctly\ labeled\ cases}{\#of\ all\ cases}$$

**Calculated accuracy score of the below example:**

$$accuracy = \frac{8}{10} = 0.8$$

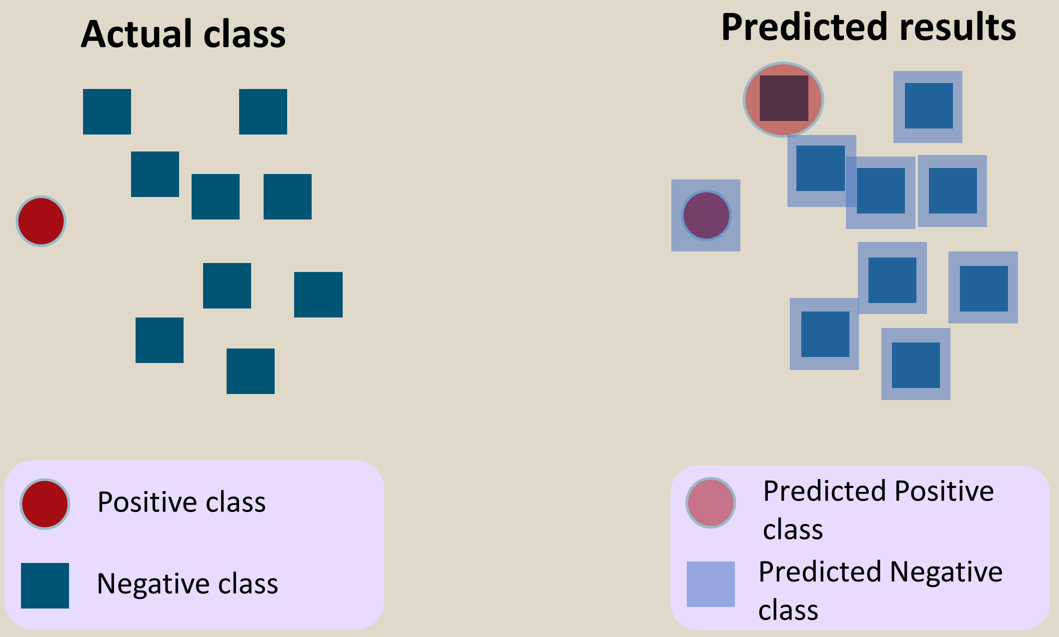
Even though we completely mis-predicted the minority class, we achieve an accuracy score of 80%.

## Pros

- Easy to interpret

## Cons

- Misleading when using unbalanced datasets



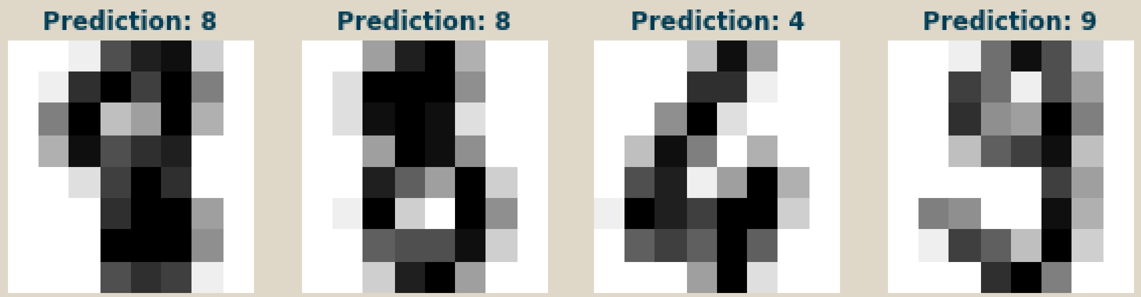
## Confusion matrix

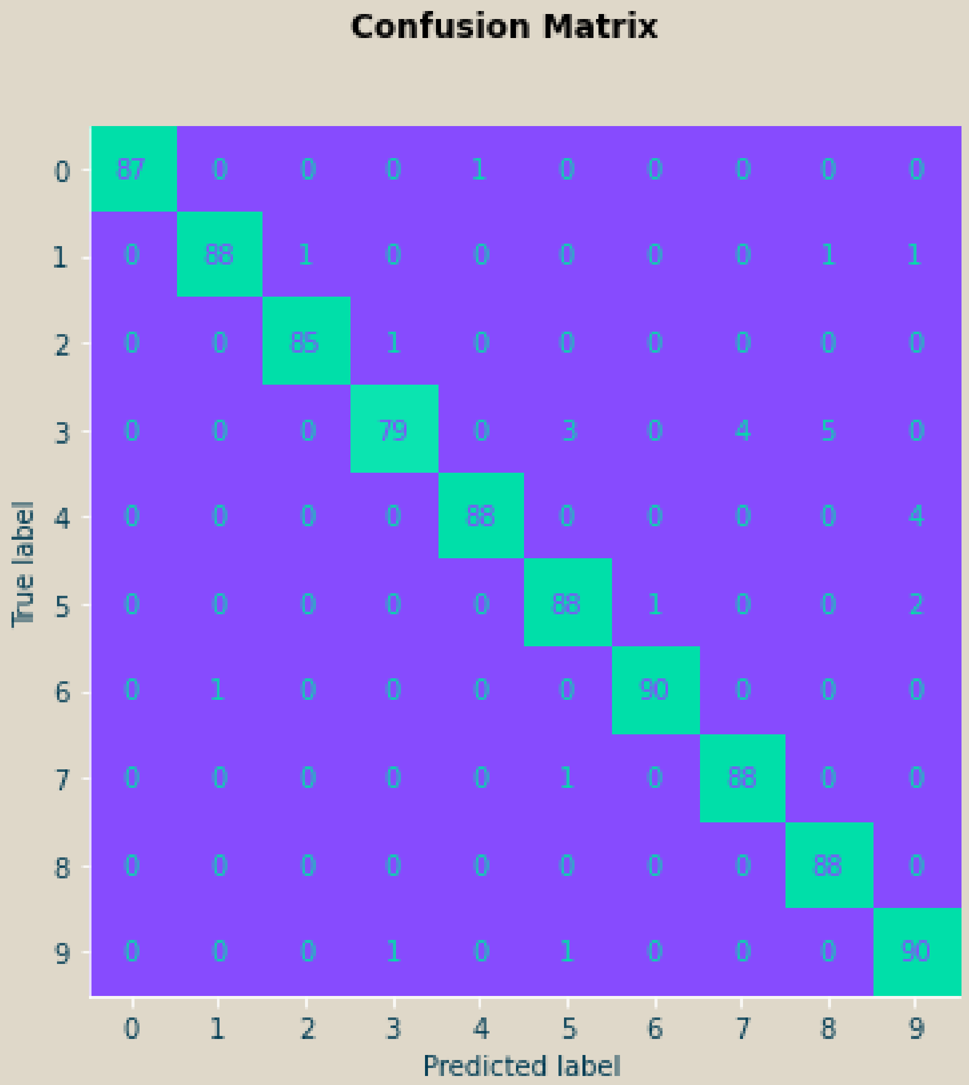
| Outputs/ Labeling | Predicted True | Predicted False | $\sum$ |
|---|---|---|---|
| **Actual True** | True positive (TP) | False negative (NP) | $P$ |
| **Actual False** | False positive (FP) | True negative (TN) | $N$ |
| $\Sigma$ | $PP$ | $PN$ | $T$ |

# Confusion matrix for multi-class problems

**Example: Optical recognition of handwritten digits**

- Popular benchmark dataset
- Contains 1797 images of handwritten number between 0 and 9
- 8x8 images with integer values for the pixel





| True label \ Predicted label | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 87 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 88 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 2 | 0 | 0 | 85 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 79 | 0 | 3 | 0 | 4 | 5 | 0 |
| 4 | 0 | 0 | 0 | 0 | 88 | 0 | 0 | 0 | 0 | 4 |
| 5 | 0 | 0 | 0 | 0 | 0 | 88 | 1 | 0 | 0 | 2 |
| 6 | 0 | 1 | 0 | 0 | 0 | 0 | 90 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 88 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 88 | 0 |
| 9 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 90 |

## When precision and recall matter

### Precision

Predicted Class

| True Class | | Positive | Negative |
|---|---|---|---|
| | Positive | TP | FN |
| | Negative | FP | TN |

Precision emphasizes **false-positives**: $\frac{\text{TP}}{\text{TP} + \text{FP}}$

- Example: The recommended video section in Youtube has only a limited capacity. Thereby it is important that the recommended videos are actually relevant to the user. Recommending all available relevant videos is not so important here.

### Recall

Predicted Class

| True Class | | Positive | Negative |
|---|---|---|---|
| | Positive | TP | FN |
| | Negative | FP | TN |

Recall accounts for **false-negative instances**: $\frac{\text{TP}}{\text{TP} + \text{FN}}$

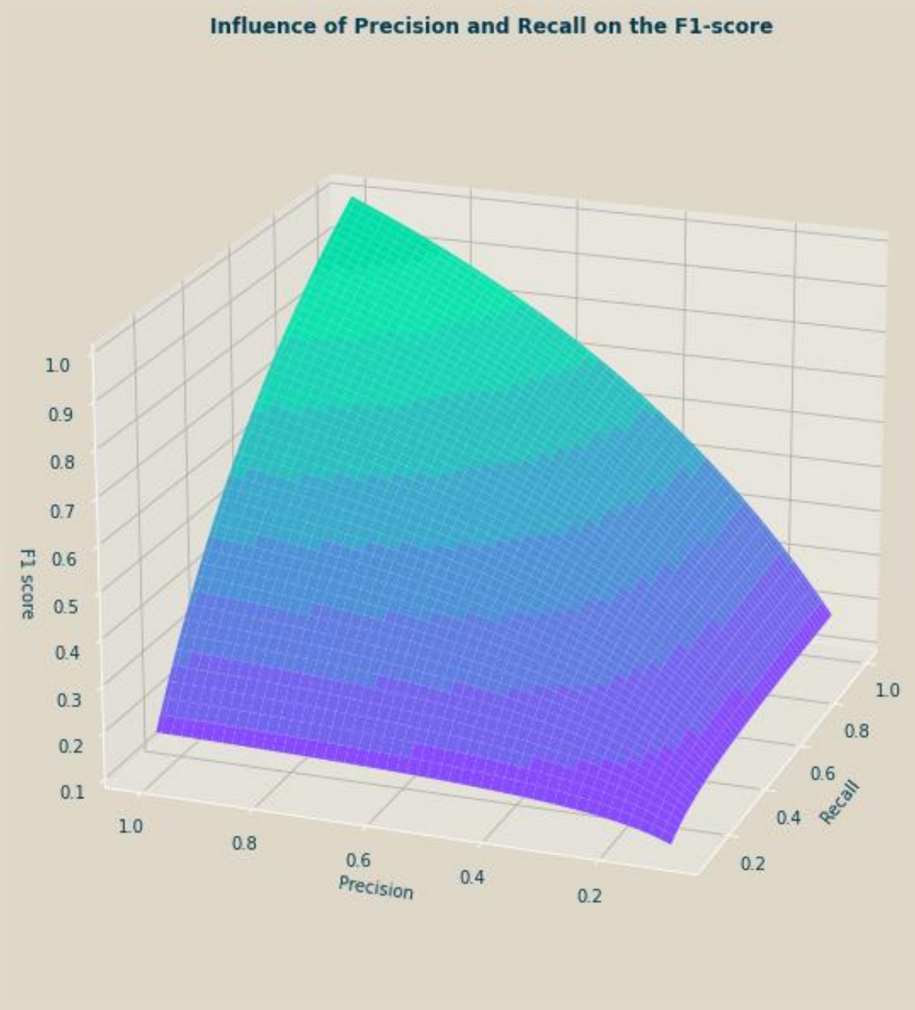
- Example: Rare cancer data modeling, any missed false-negative can be disastrous (i.e., someone is sick but not identified). It is less problematic to identify someone as a cancer patient who is not, this can be clarified during the next doctoral appointment.

# Balancing Precision and Recall: F1 Score

- F1 score: Tradeoff between precision and recall
  - **Precision:** Of all positive predicted samples which percentage were actually positive? Increase the score by only predicting positive when we are certain.
  - **Recall:** What ratio of all actual positive samples did we find? Increase the score by predicting as many samples as positive as possible.

$$F1 = \frac{2 \ * \text{precision} \ * \text{recall}}{\text{precision} + \text{recall}}$$

- Good metric, for uneven class distributions
  - Unbalanced datasets
- Difficult to interpret



# Sensitivity and False Positive Rate

- Sensitivity (TPR = True Positive Rate/Recall):
  - **Detect type II errors:** How often did we not predicted true, when the label was true?

$$Sensitivity = \frac{\#predicted\ true\ and\ actually\ true}{\#actually\ true}$$

$$Sensitivity = \frac{TP}{P}$$

- False Positive Rate (FPR):
  - **Detect type I errors:** How often did we predict positive, but when the label was actually negative?

$$\text{FPR} = \frac{\#predicted\ true\ and\ actually\ false}{\#actually\ false}$$

$$\text{FPR} = \frac{FP}{N}$$

$$Specificity = 1 - \text{FPR}$$

| Outputs/ Labeling | Predicted True | Predicted False | $\Sigma$ |
|---|---|---|---|
| Actual True | True positive (TP) | False negative (NP) | $P$ |
| Actual False | False positive (FP) | True negative (TN) | $N$ |
| $\Sigma$ | $PP$ | $PN$ | $T$ |

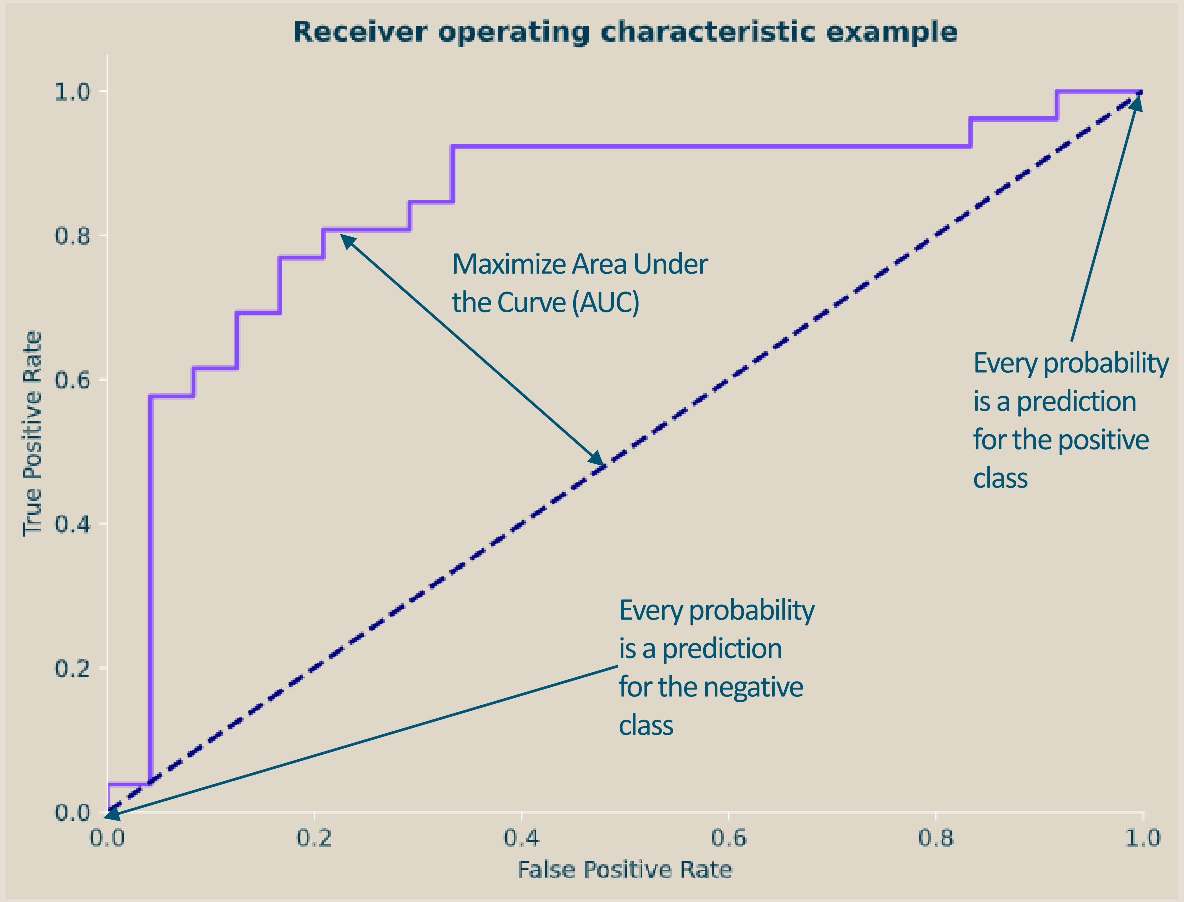
Sensitivity

| Outputs/ Labeling | Predicted True | Predicted False | $\Sigma$ |
|---|---|---|---|
| Actual True | True positive (TP) | False negative (NP) | $P$ |
| Actual False | False positive (FP) | True negative (TN) | $N$ |
| $\Sigma$ | $PP$ | $PN$ | $T$ |

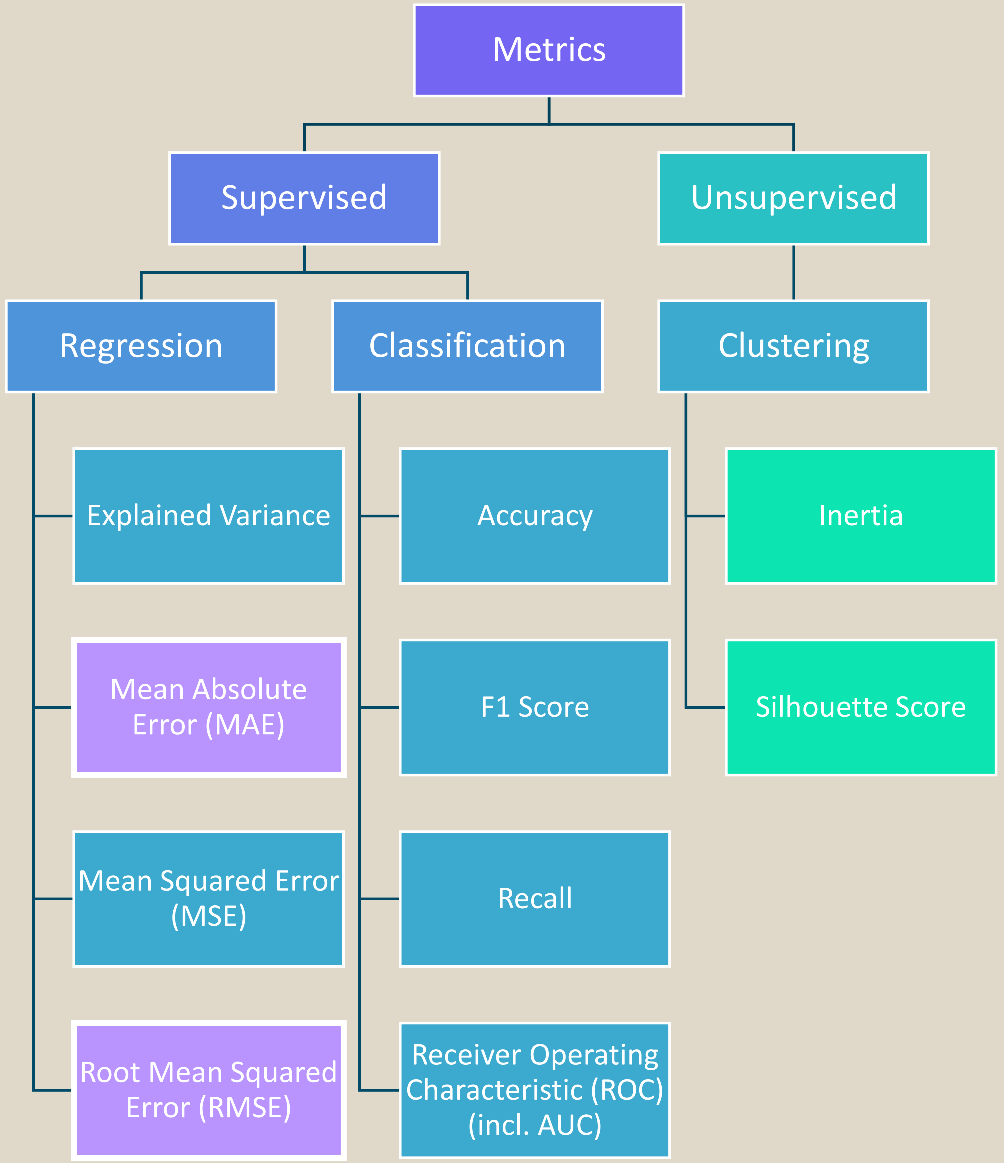
FPR

# Receiver Operating Curve (ROC)

- For decision problems we can **quantify our certainty** in the prediction
- Choosing the **threshold** to discretize the prediction is important
- We can make this decision by plotting the **True Positive Rate** against the **False Positive Rate**
  - Depending on the context we can adjust our threshold
- We can use the **area under the curve (AUC)** to determine how well the algorithm is performing in total



# Selected evaluation metrics

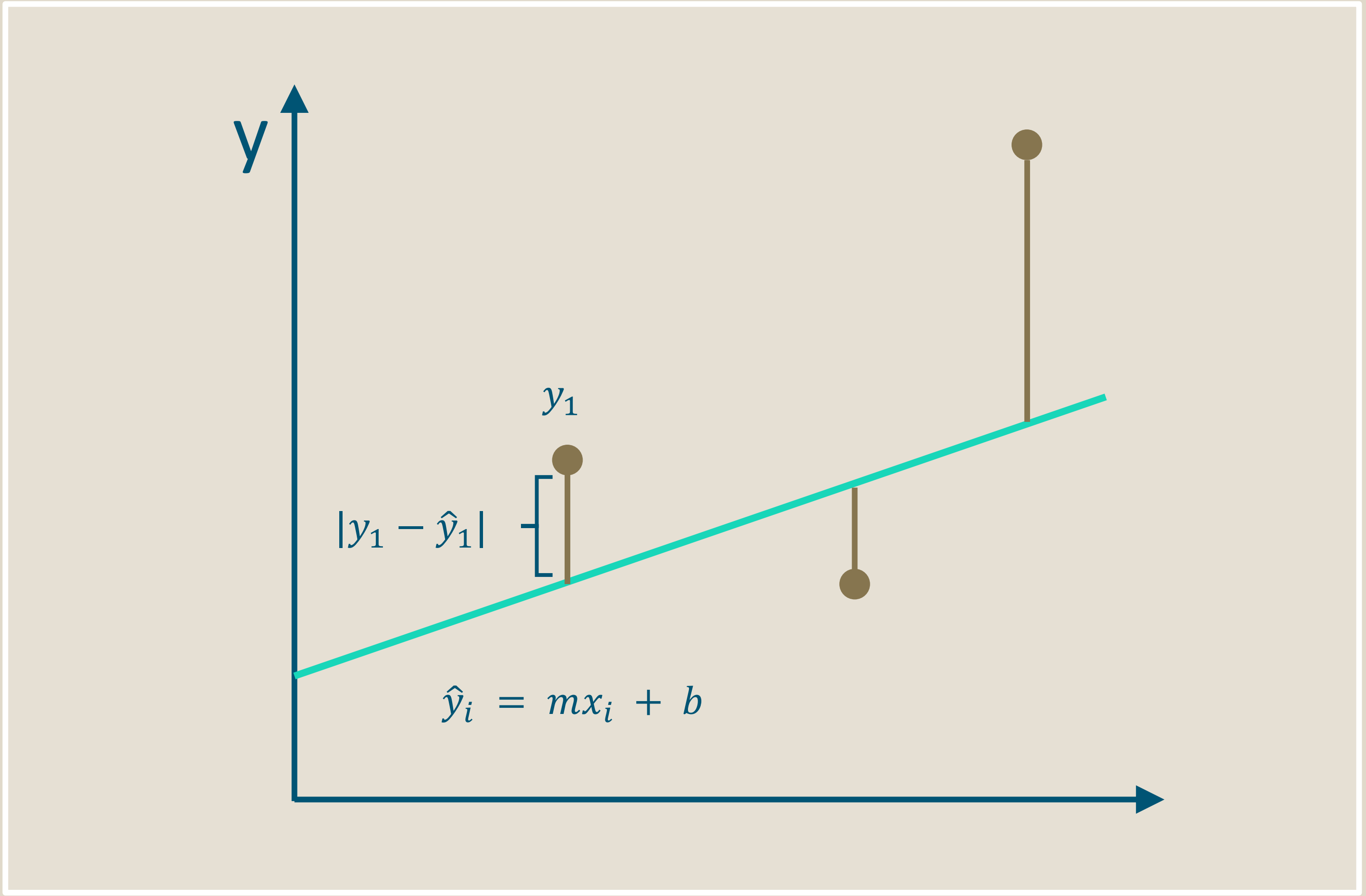


# Mean absolute error (MAE)

- MAE is the average absolute error **between the predicted and the actual outcome**
- It is mostly used to evaluate regression models

$$MAE = \frac{1}{N} \sum |y_i - \hat{y}_i|$$

  - $y$ is the actual value
  - $\hat{y}$ is the predicted value
  - $N$ is the total number of datapoints



# Root Mean Squared Error (RMSE)

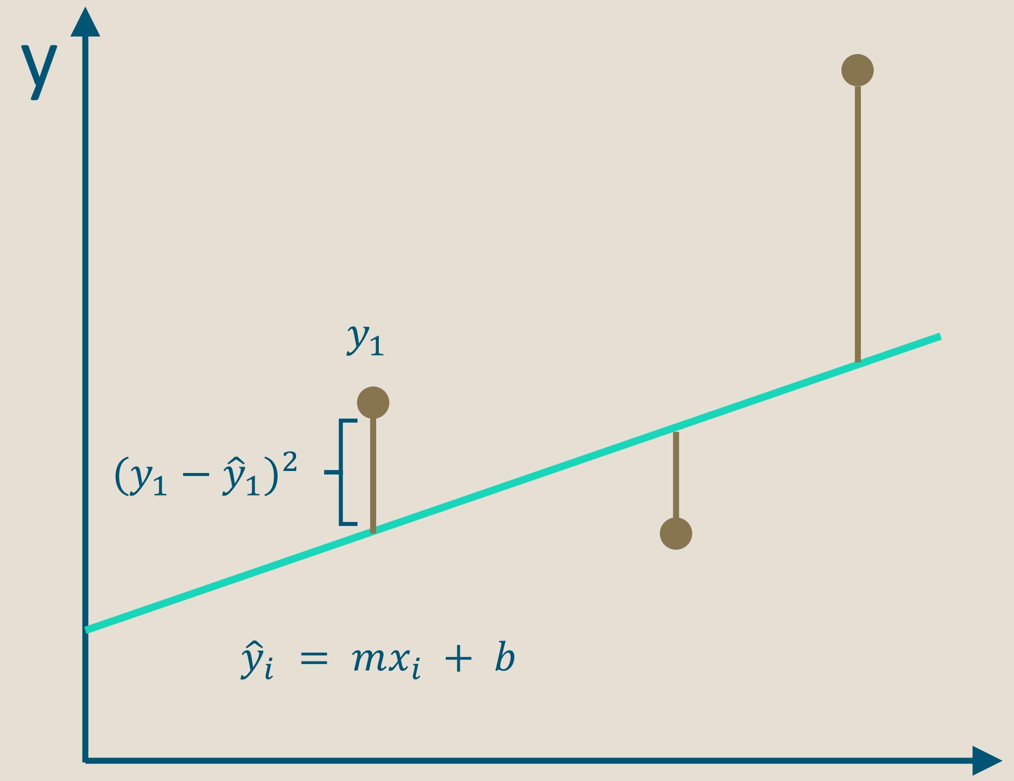
- Similar to mean absolute error, but **outliers are much more costly** due to the squared error

$$MSE = \frac{1}{N}\sum(y_i - \hat{y}_i)^2$$

- With the square root we **scale the data in the same range as the target value**
- RMSE (Root Mean Squared Error):

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{N}}$$

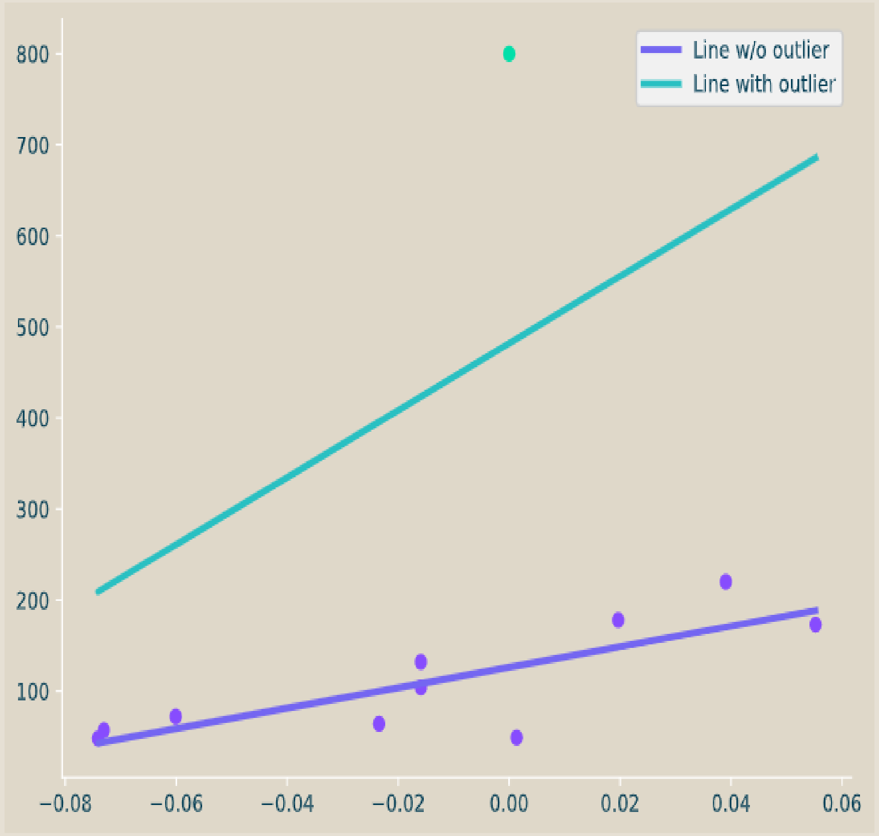
- Usually: **RMSE ~ 30% bigger than MAE**, if not then ‘very far off’ predictions!



# Overview outliers

- Outliers are **observations that diverge from an overall pattern** on a sample
- We have to **pay attention to outliers**, because models can overemphasize the importance of these outliers
  - E.g., linear regression
- **Outlier detection**
  - MAE/RMSE (next slide)
  - Isolation forest (see later in Module: Unsupervised learning)
- **Outlier handling** highly **depends on the** problem **context**
  - Use models that are robust against outliers
  - Clip values to a threshold
  - Exclude outliers

Exaggerated effect of an outlier on linear regression



# Outlier detection with MAE and RMSE

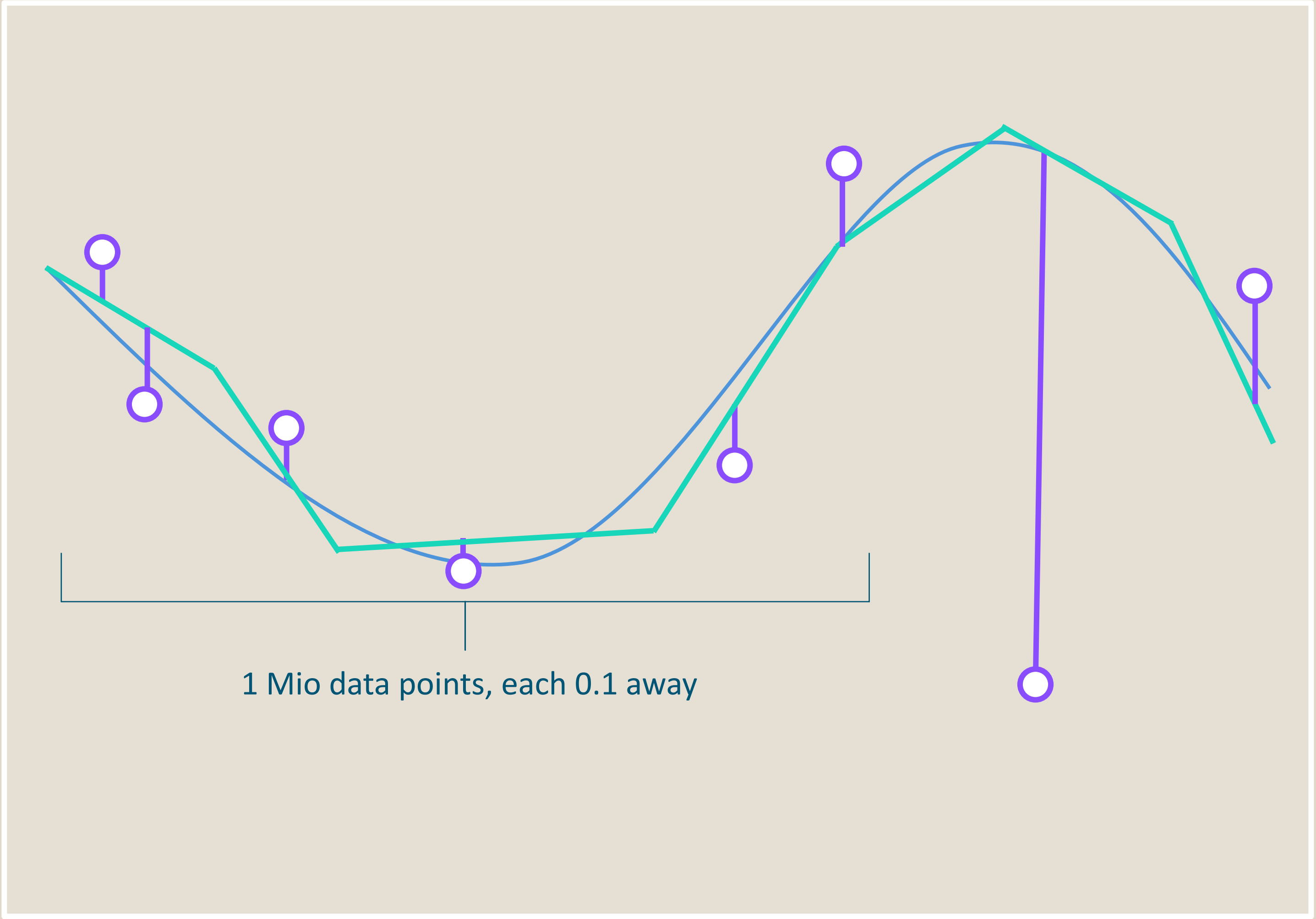
**Hypothetical case**

- 1 Mio data points, they are each 0.1 away from the generating function
- There is one outlier which has a distance of 20.000 to the target function

$$MAE = \frac{0.1 * 999K + 20.000}{1\ \text{Mio}} = 0.11$$

$$RMSE = \sqrt{\frac{999K\ (0.1)^2 + 1 * \mathbf{(20.000)^2}}{1\ Mio}}$$
$$= 20$$

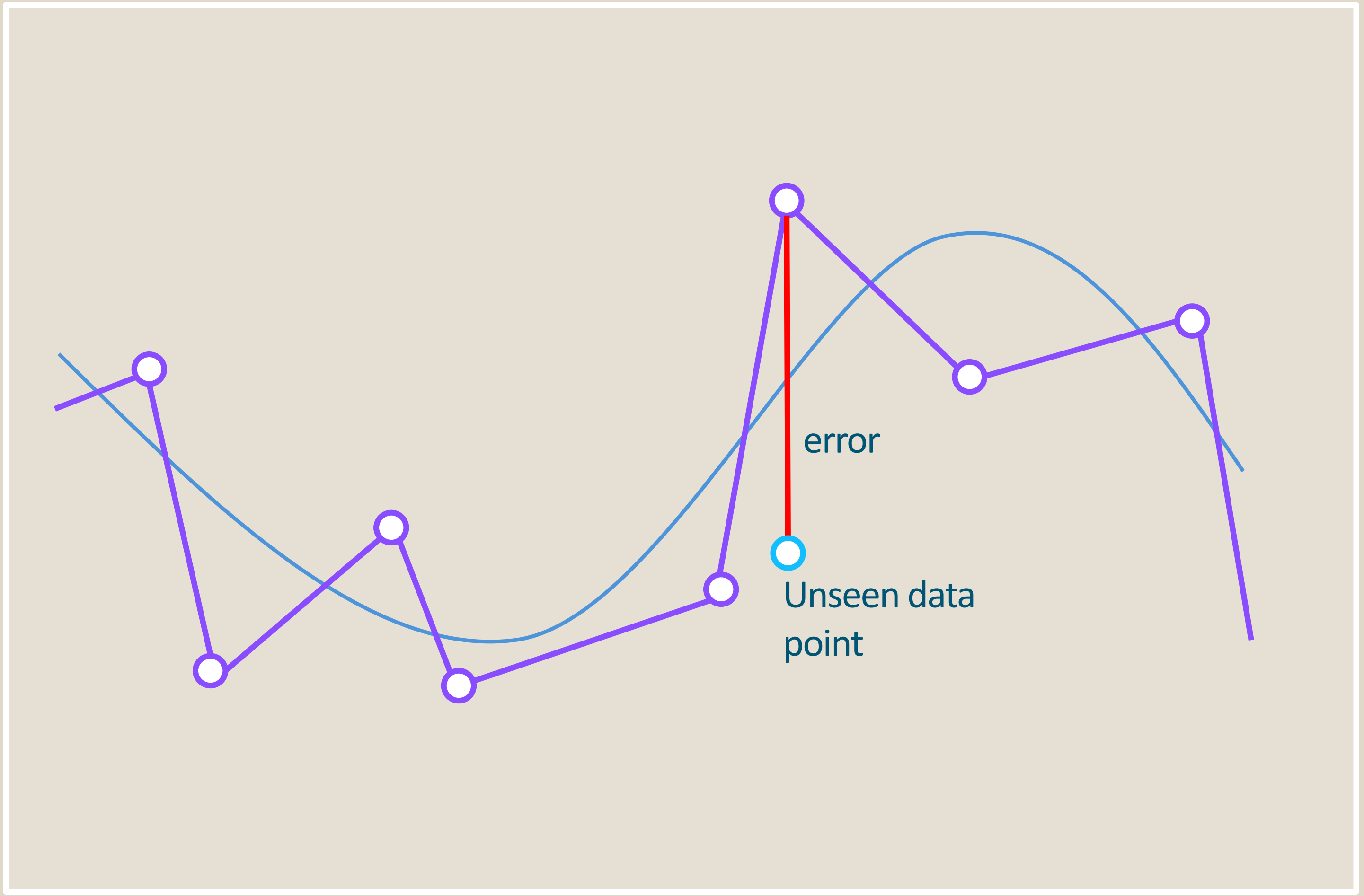
- Usually RMSE = 1.3 * MAE
- RMSE **much** higher than MAE
  - Outliers present
  - Usually error in data
  - Remove outliers, re-evaluate



## Evaluate the model performance on unseen data

To estimate the **true performance of the model** we have to **test it on unseen data**

- The most commonly used method for this is the **train-test-split**

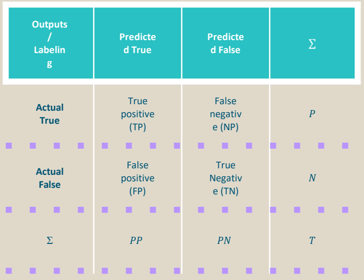


# Important takeaways

$$L(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} l(y_i, \hat{y}_i)$$

The **loss function** is used to evaluate the model performance during the training

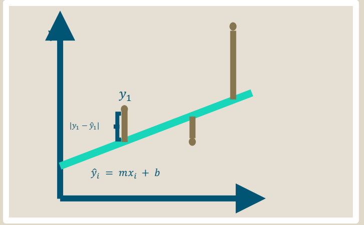
- But it is difficult to interpret



| Outputs / Labeling | Predicted True | Predicted False | $\Sigma$ |
|---|---|---|---|
| **Actual True** | True positive (TP) | False negative (NP) | $P$ |
| **Actual False** | False positive (FP) | True Negative (TN) | $N$ |
| $\Sigma$ | $PP$ | $PN$ | $T$ |

The **Confusion Matrix** collects information about the model performance for classification problems

- We can use the confusion matrix to calculate the **accuracy**, **precision**, **recall** and **F1** score
- With **true positive rate** and **false positive rate**, we can create a ROC curve



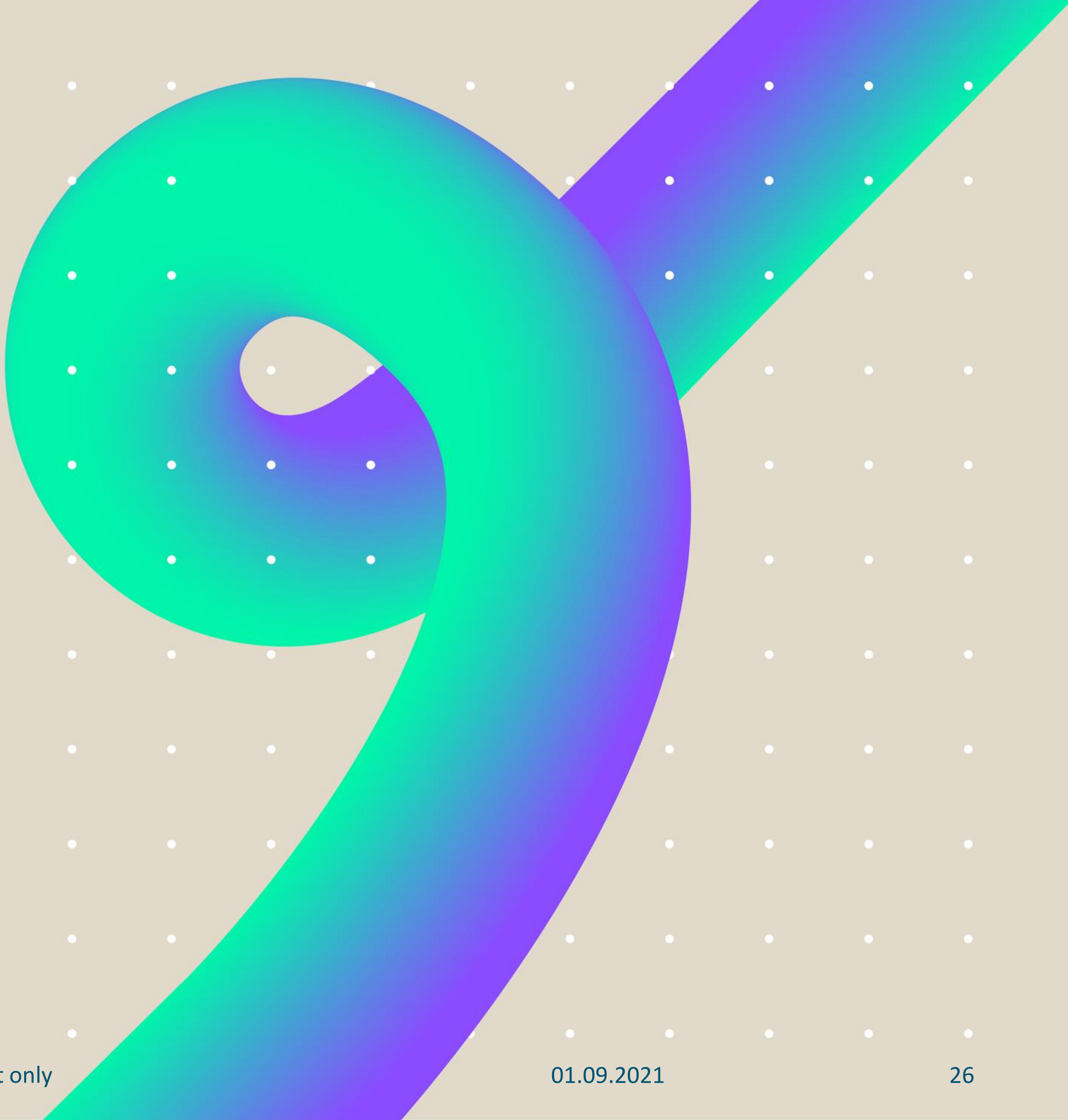
There are a lot of different metrics to evaluate regression models. Two easy to interpret metrics are:

- Mean Absolute Error **(MAE)** and
- Root Mean Squared Error **(RMSE)**

**Choosing a suitable metric depends on the context of your problem and your data.**

Try it yourself!

## In the following exercises

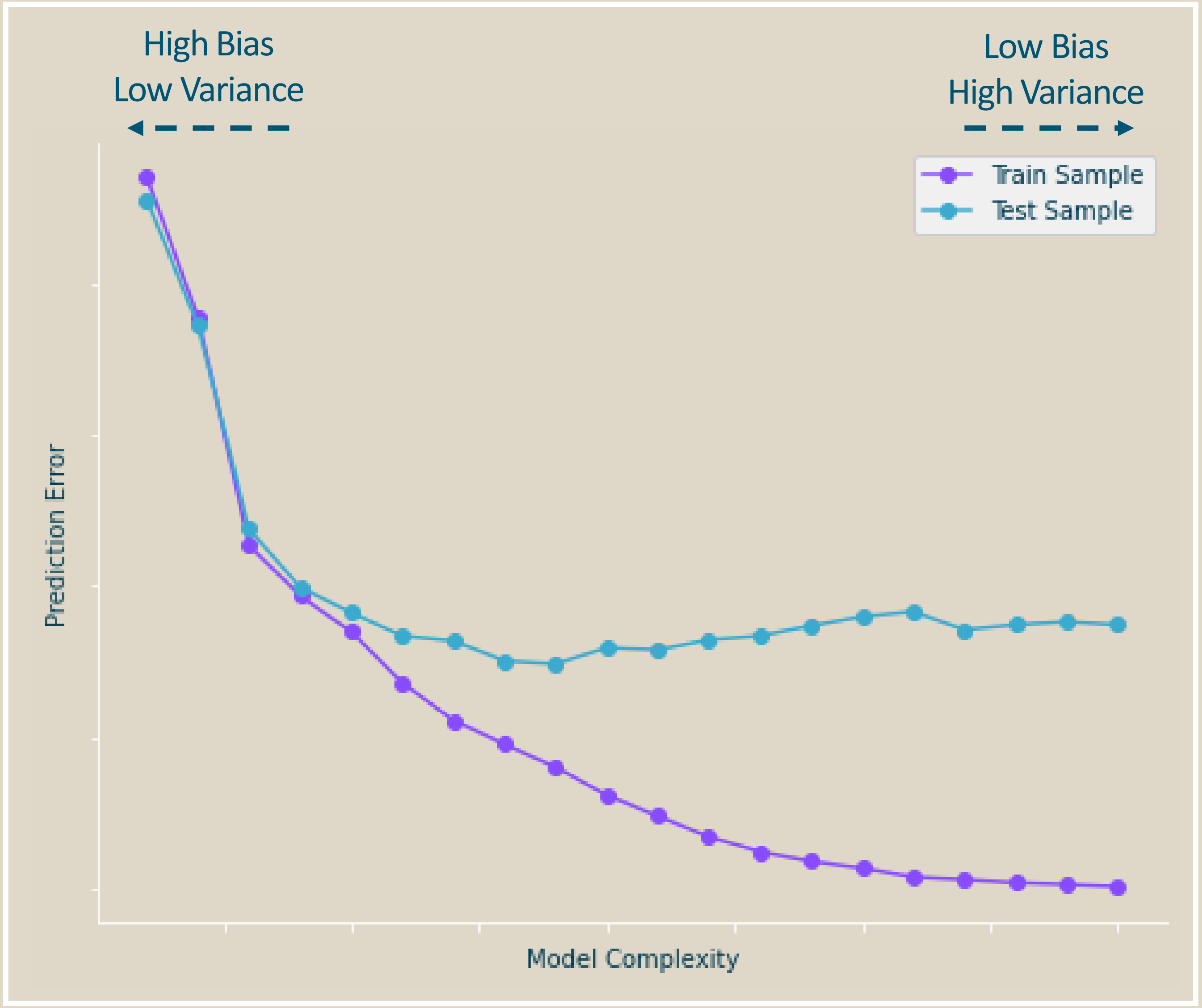


Module 4

## Hyperparameters

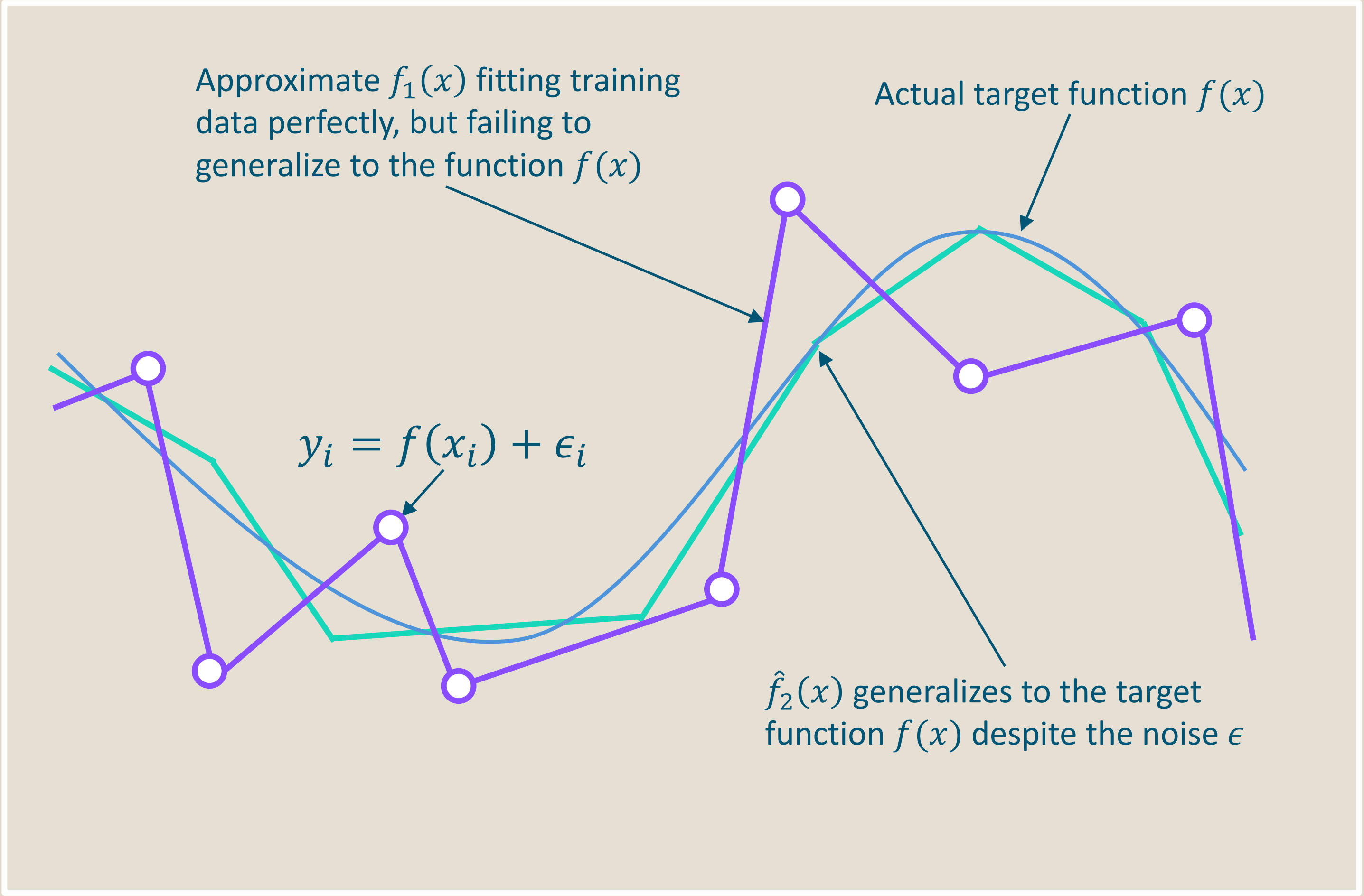
# Agenda week one

# Why do we need a test dataset?



# How does this happen?

- Assume that we have…
  - A **train dataset** with feature vectors and true answers
  - Computed a **model on this train dataset**
  - Created the model predictions
    $$\hat{y} = \hat{y}_1, \ldots, \hat{y}_n$$
  - Evaluated the model with the answers
    $$y = y_1, \ldots, y_n$$
- Using the **training set for evaluation** can lead to **misleading results**

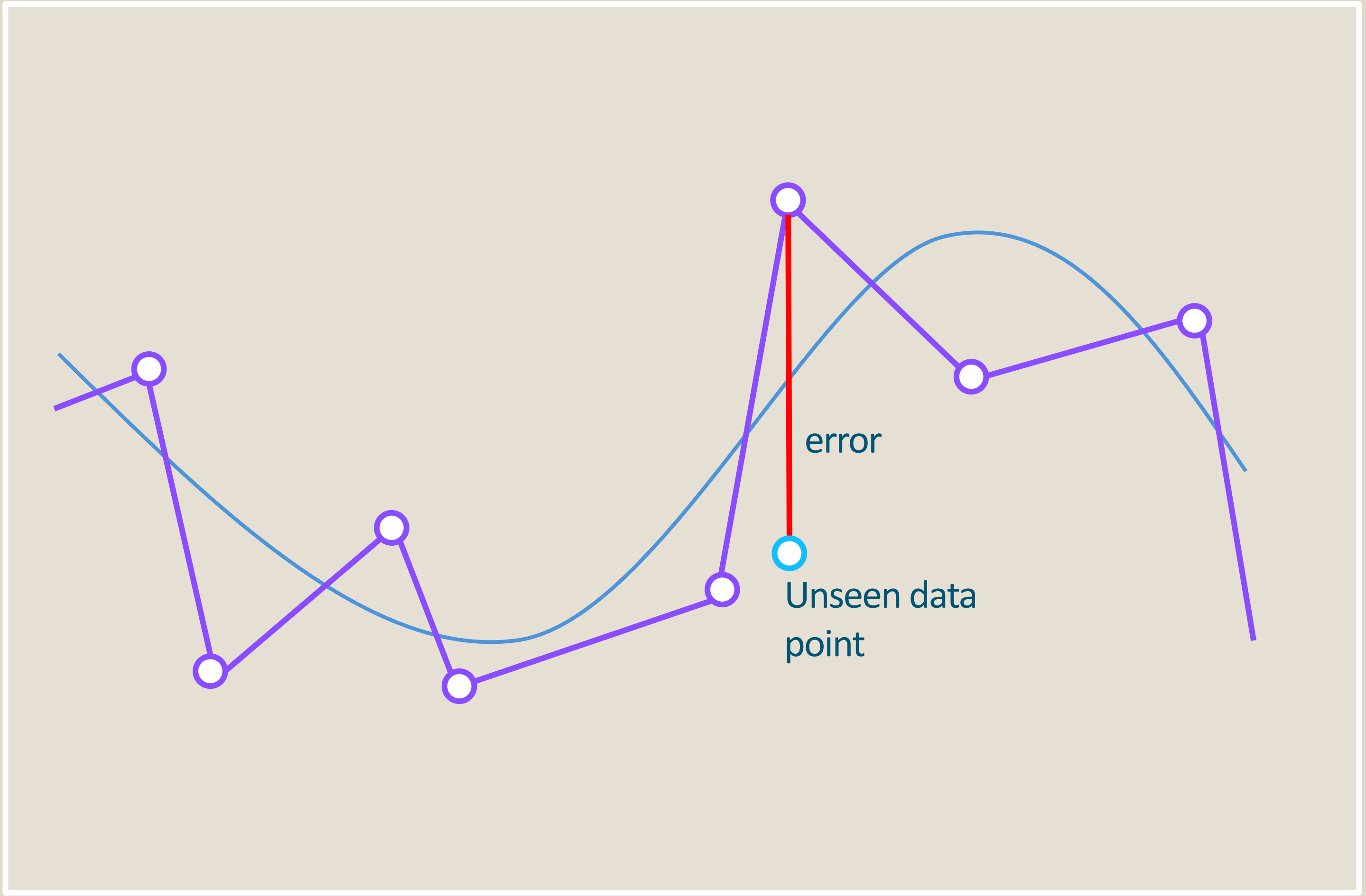


Approximate $f_1(x)$ fitting training data perfectly, but failing to generalize to the function $f(x)$
Actual target function $f(x)$
$y_i = f(x_i) + \epsilon_i$
$\hat{f}_2(x)$ generalizes to the target function $f(x)$ despite the noise $\epsilon$

# Overfitting

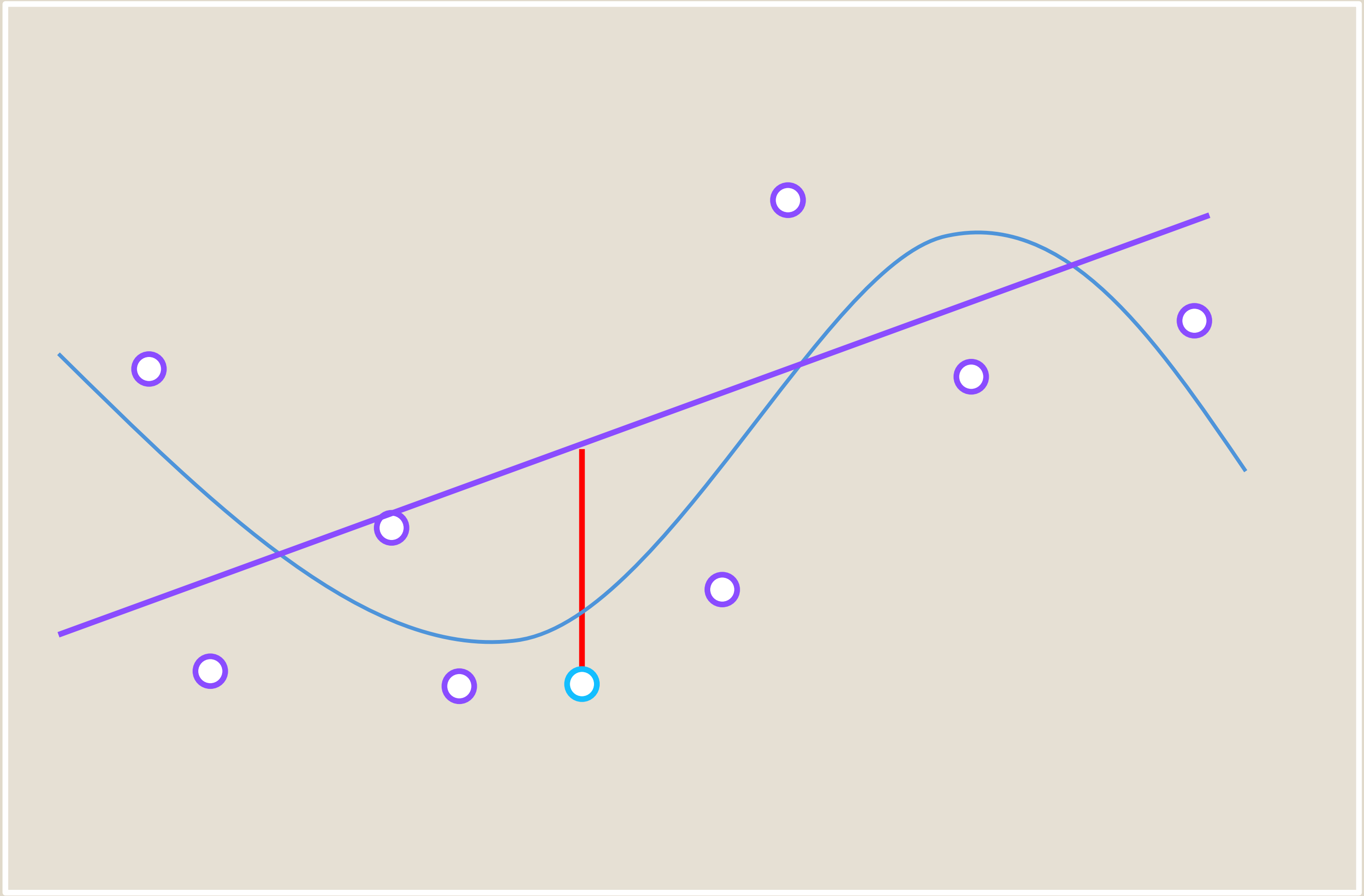
- Evaluating $\hat{f}_1(x)$ **on unseen data** reveals a **high error**, because the data is still generated by $f(x)$ and $\hat{f}_1(x)$ **fitted the noise rather than the target function**

- **The function is too complex** and thereby has too much variance around the target function



# Underfitting

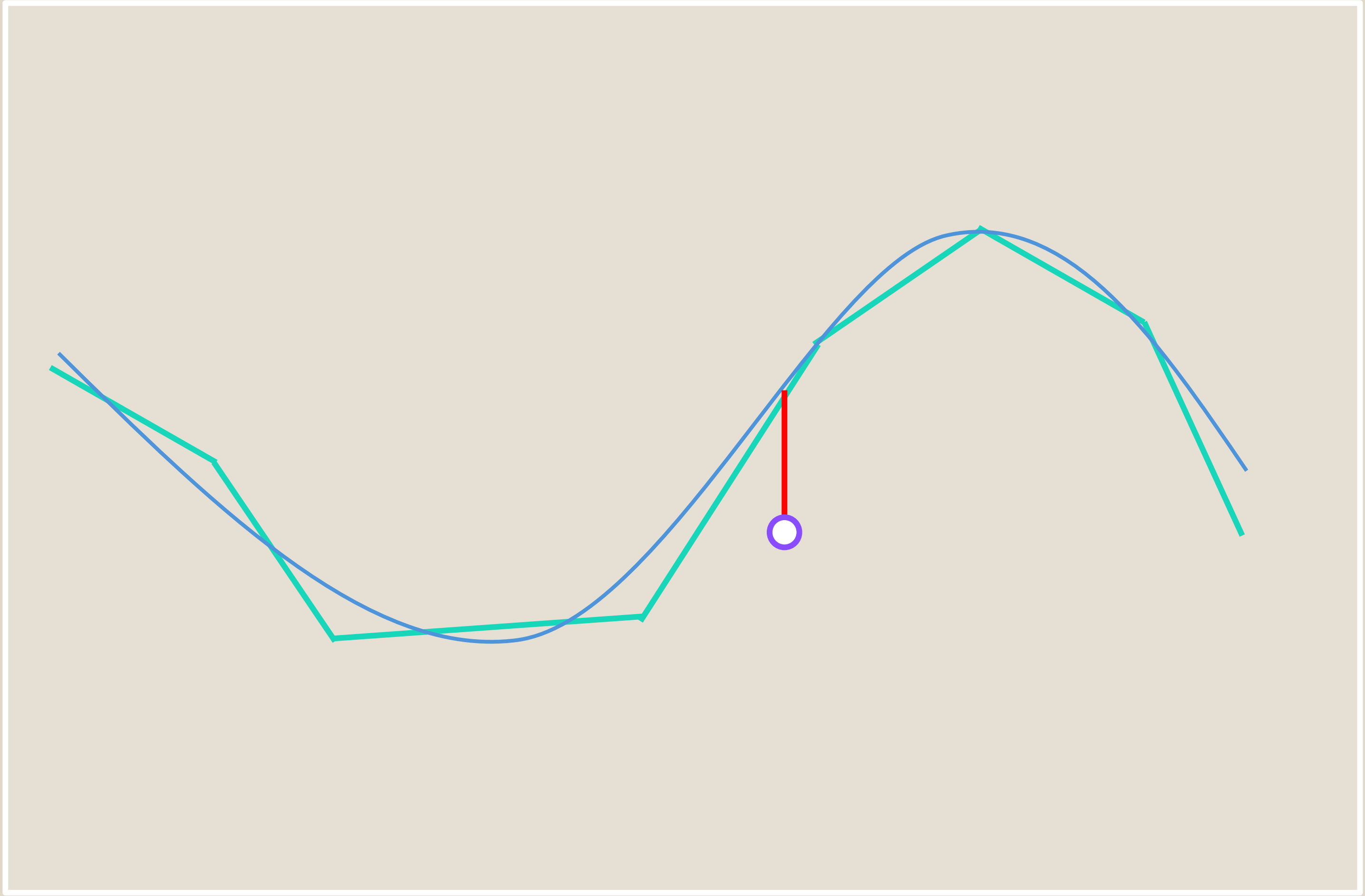
- When we allow our function **too little freedom** it **can not fit the target function properly**
- **The function is too simple** and therefore biased in one direction



# Good fit

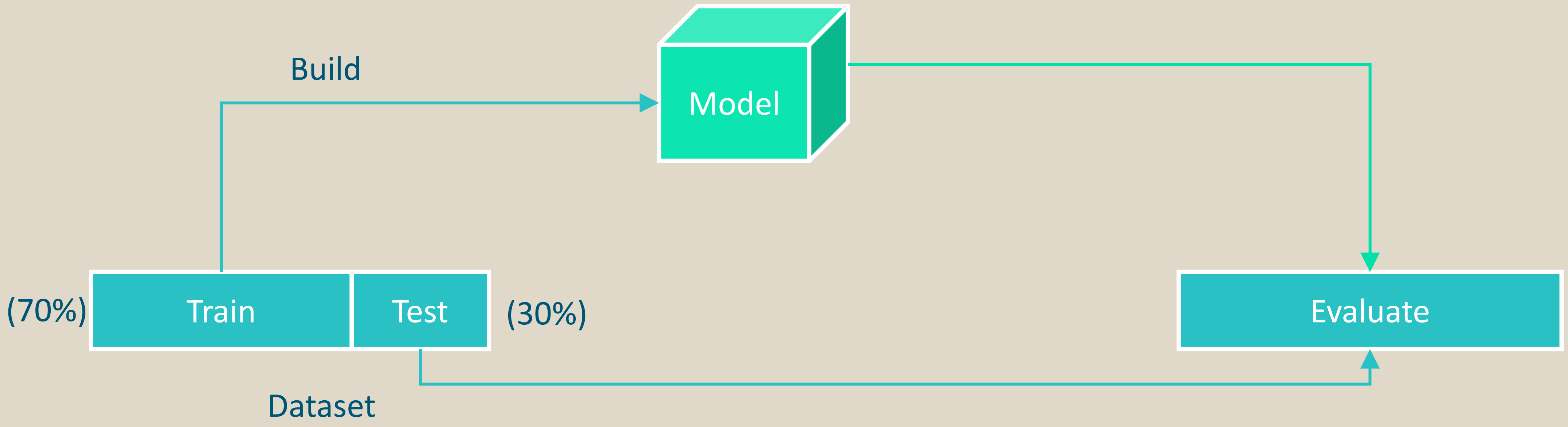
- Good **tradeoff between bias and variance**
- The approximation **function minimizes the average error to the target function**, instead of to all the data points

- **How** do we **choose the best level of complexity** a model should have?
- **How** can we **best evaluate** our decision?



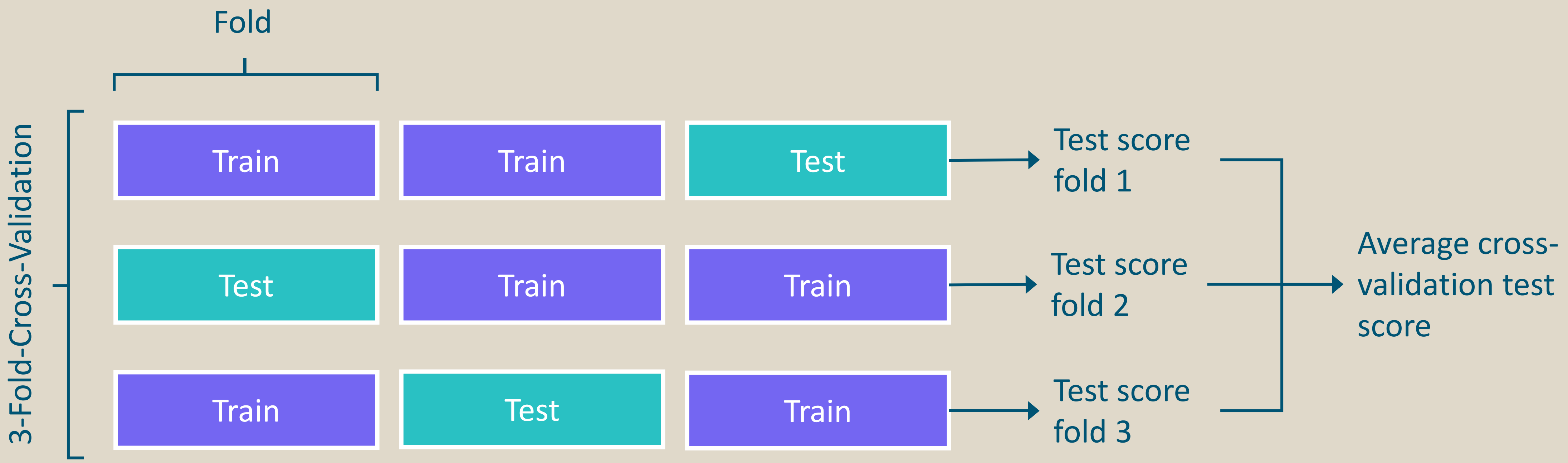
# Train-Test-Split

- Our ML model should learn **general rules** that hold true on specific cases.
- Until now, we have evaluated the model performance on the data set the model was trained on.
- **Testing** the model **on the training data** is **not sufficient to determine its generalizability**.
  - On training data, we might evaluate how strongly the model memorized the data
- To prevent that we **test the model on unseen data**



## Cross-Validation

- Mostly needed when there is **little data** available
- After we **evaluated the model on each fold once**, we have tested the generalization performance on the whole dataset
- The evaluation is **expensive**, since we must train the model k-fold times

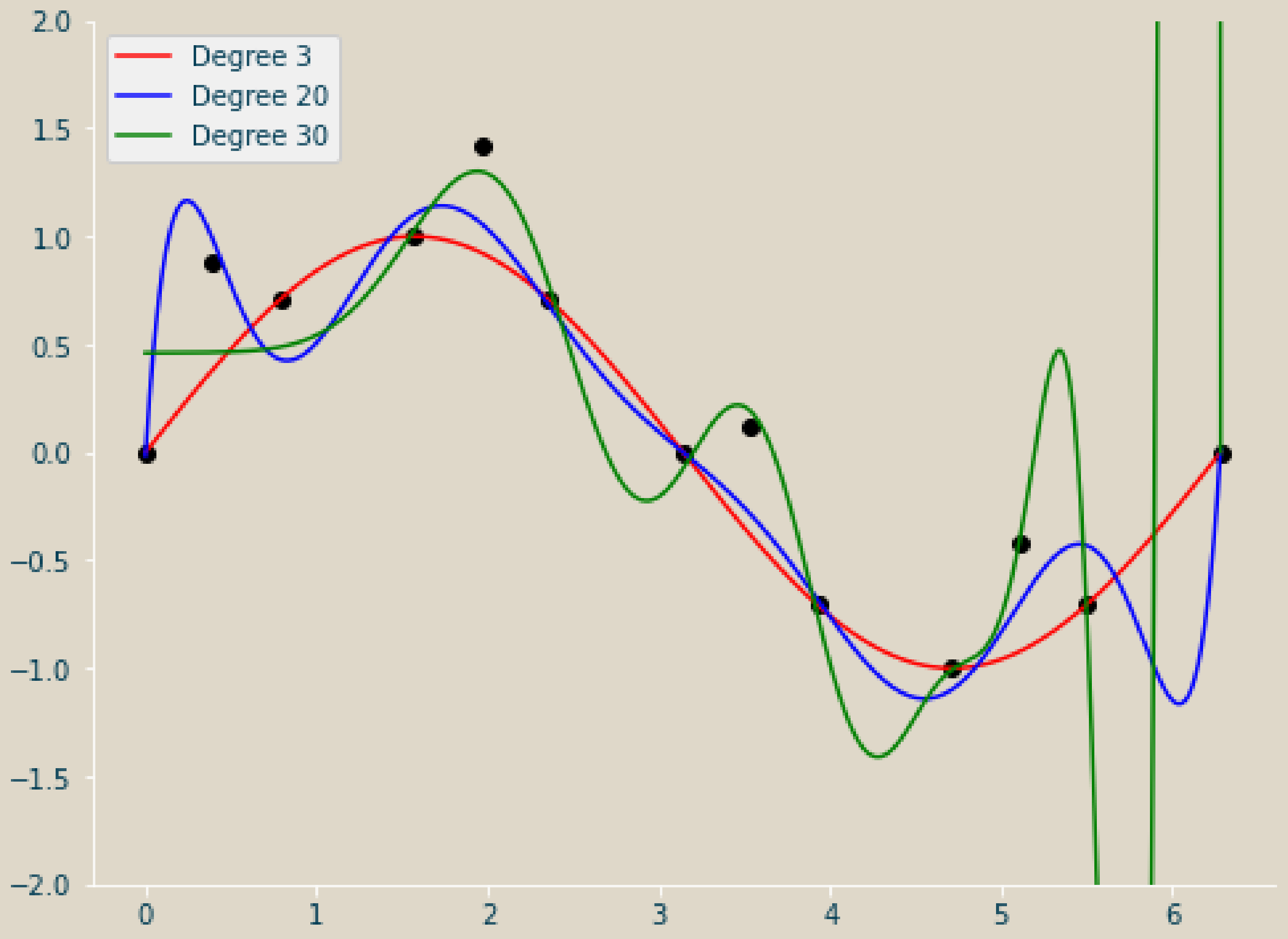


# HyperParameter Optimization (HPO)

In Machine Learning we have

- **Parameters** which
  - Specify our model and predicted outcomes
  - Are **learned based on the data**
- **Hyperparameters** which
  - Steer the **learning process**
  - Let us control the **complexity of our model**
  - Are not learned based on the data, but **need to be defined** by the Data Scientist

**But how can we define hyperparameters?**

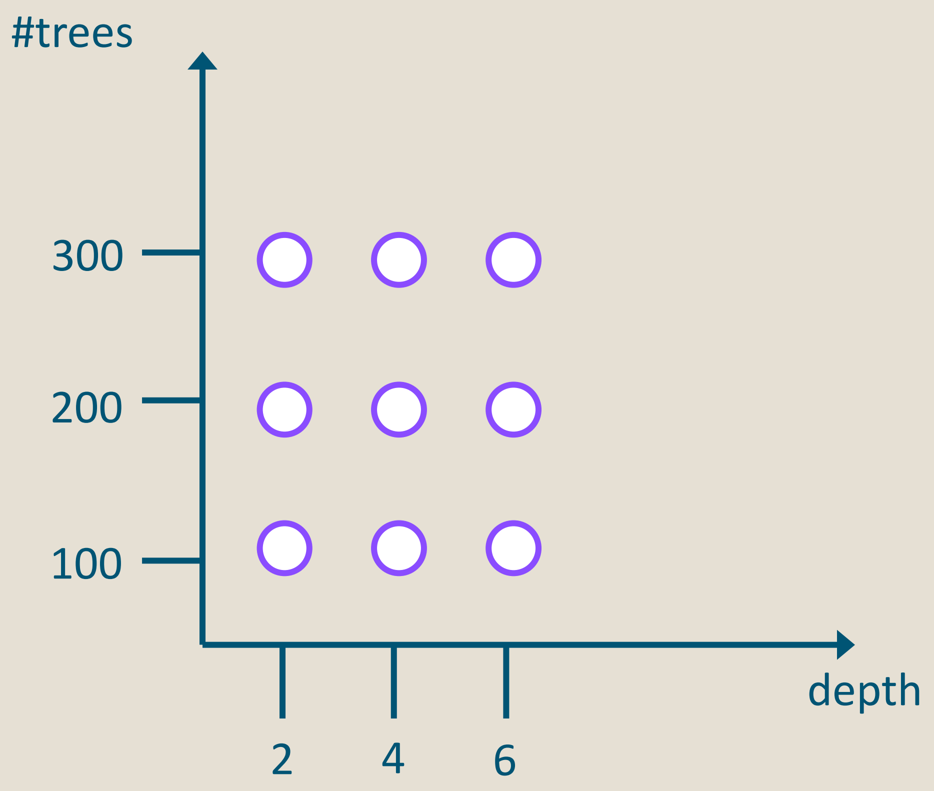


## Manual hyperparameter optimization

- For every problem and dataset the choice of hyperparameters will be different
- For each problem we have to **select a hyperparameter combination**, **train the model** and **evaluate it on the test set**
- Depending on signs of overfitting or underfitting we **increase or decrease the complexity** of the model
- Problems
  - The manual tuning with multiple interdependent hyperparameters can become **untrackable for a human really fast**.

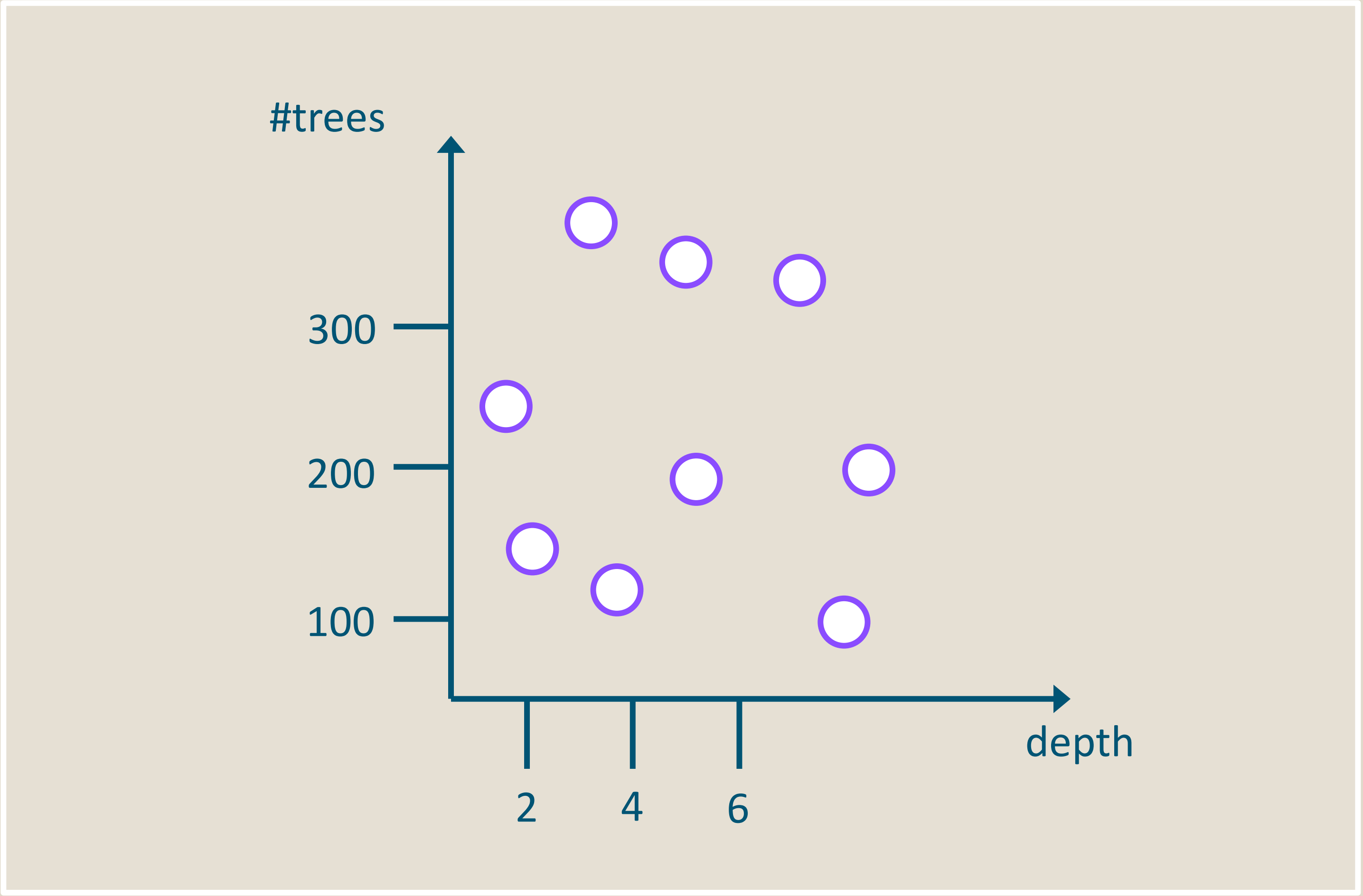
# Strategies: Grid search

- Setup a **grid** and go through it
- Example: Forests: #trees and depth
- Problem
  - We waste a lot of **computation time** checking combinations involving e.g., 100 trees!
  - What if 3 is the right depth?
  - How do we choose the range of #trees and depth?



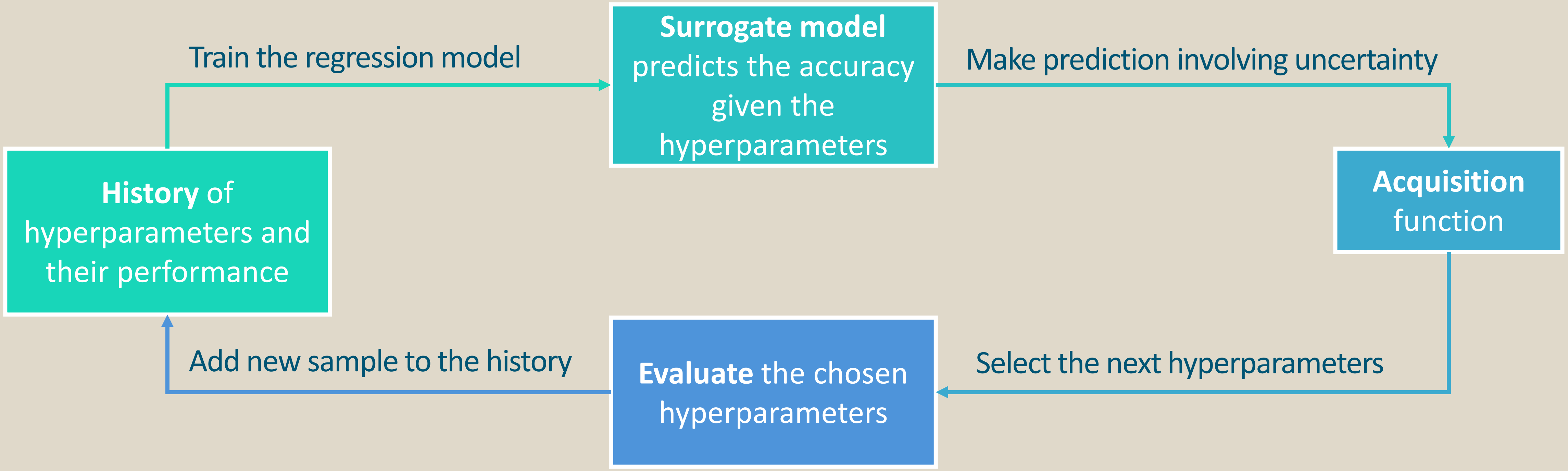
## Strategies: Random search

- Setup **boundaries and sample randomly**
- If 100 is bad, we do not waste time checking combinations with 100 trees anymore…
- We do not need to check whether 100 step size is fine enough
  - **Always prefer random search over grid search!**

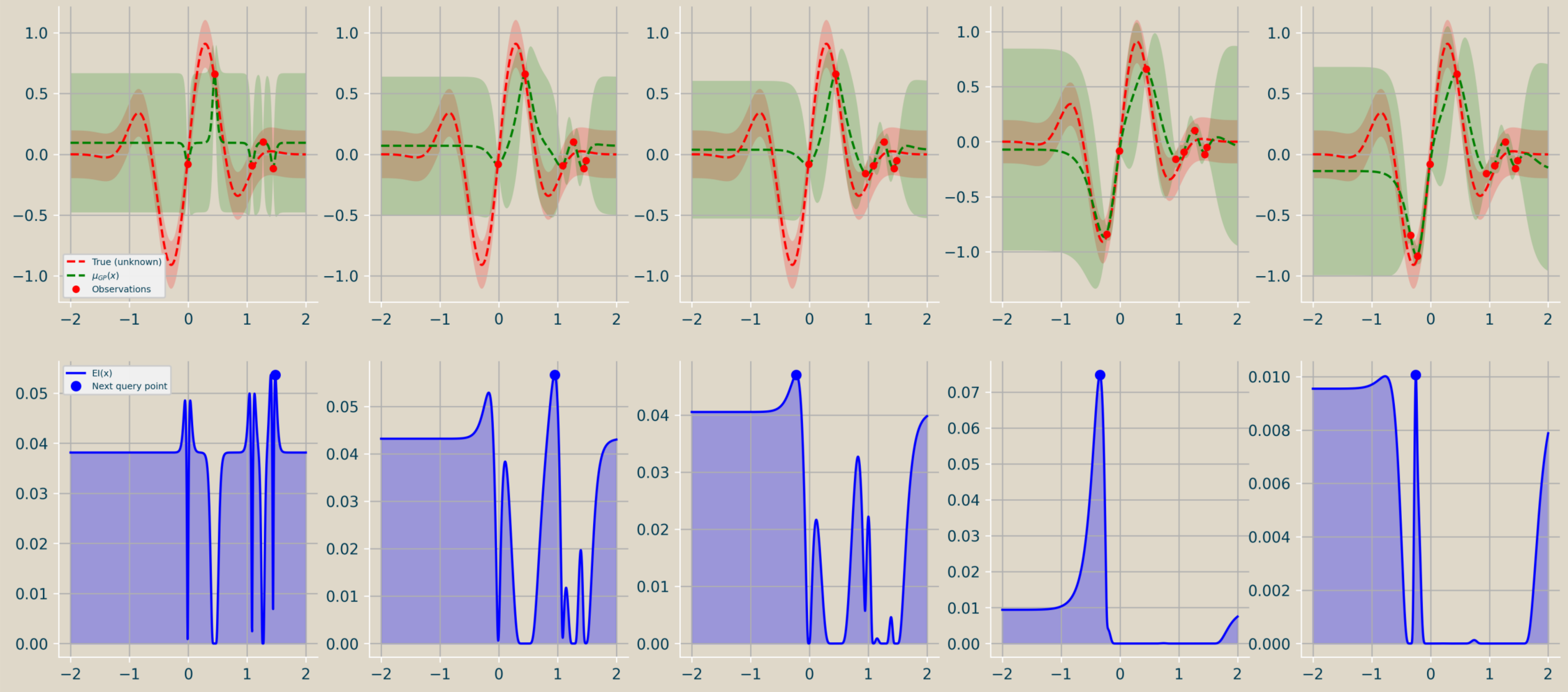


## Strategies: Bayesian Optimization (BO)

- Instead of choosing hyperparameters randomly, we want to **select hyperparameters that are likely to perform well**
- We create a regression model that predicts the test performance based on the hyperparameters
- We let the regression model choose our hyperparameters and **evaluate the hyperparameters on the validation set**
- After each evaluation we can **update the regression model** and get a better prediction the next time around

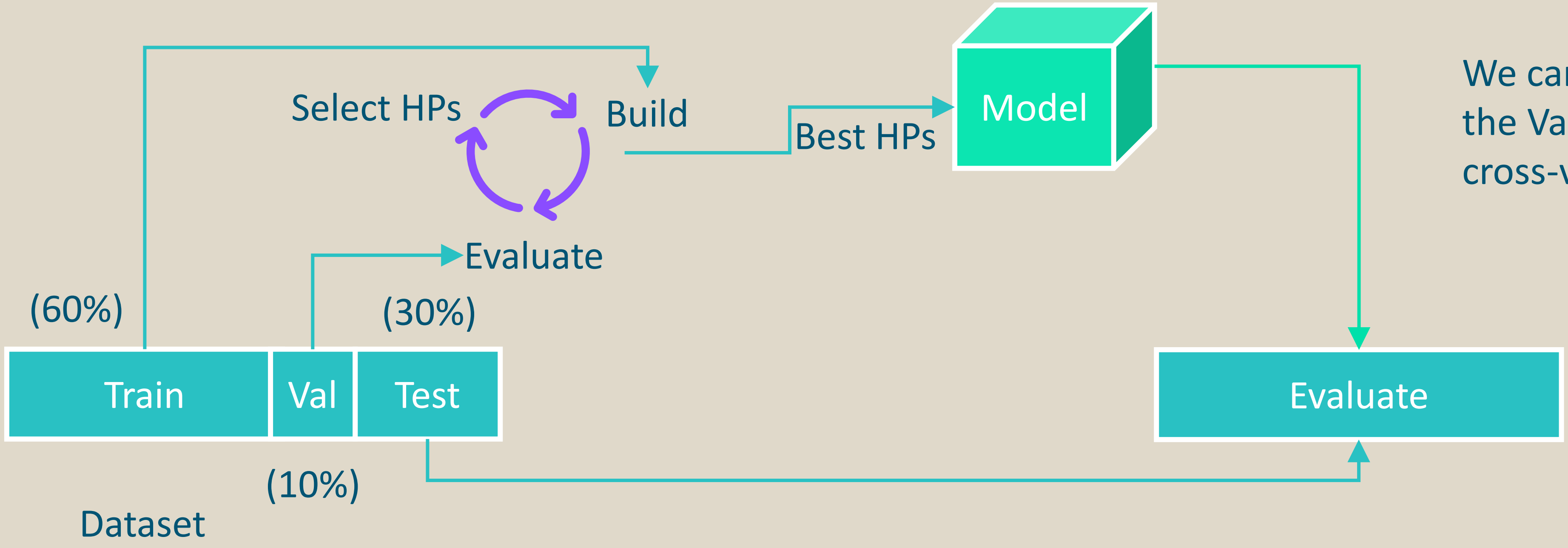

Train the regression model
Surrogate model predicts the accuracy given the hyperparameters
Make prediction involving uncertainty
History of hyperparameters and their performance
Acquisition function
Add new sample to the history
Evaluate the chosen hyperparameters
Select the next hyperparameters

# Visualization of Bayesian Optimization



# Train-Test-Validation Split

- Split the data into **train set (60%)**, **test set (30%)** and **validation set (10%)**
- Train the model on the train set, evaluate the model performance on the validation set
- After the hyperparameter tuning do **one final evaluation on the test set**



We can select the Val set with cross-validation

## Example: Models and their hyperparameters

| Model | Complexity Hyperparameters | Not Complexity Hyperparameters |
|---|---|---|
| **Decision Tree** | Max depth<br>Min split samples<br>Min impurity decrease | Sample weight<br>Impurity function |
| **Polynomial regression** | Maxiumum degree | |
| **MLP** | Number of layers<br>Number of neurons in layer | Activation function<br>Learning rate<br>Optimizer<br>Solver<br>Batch size<br>Momentum |

# Regularization: Controlling complexity

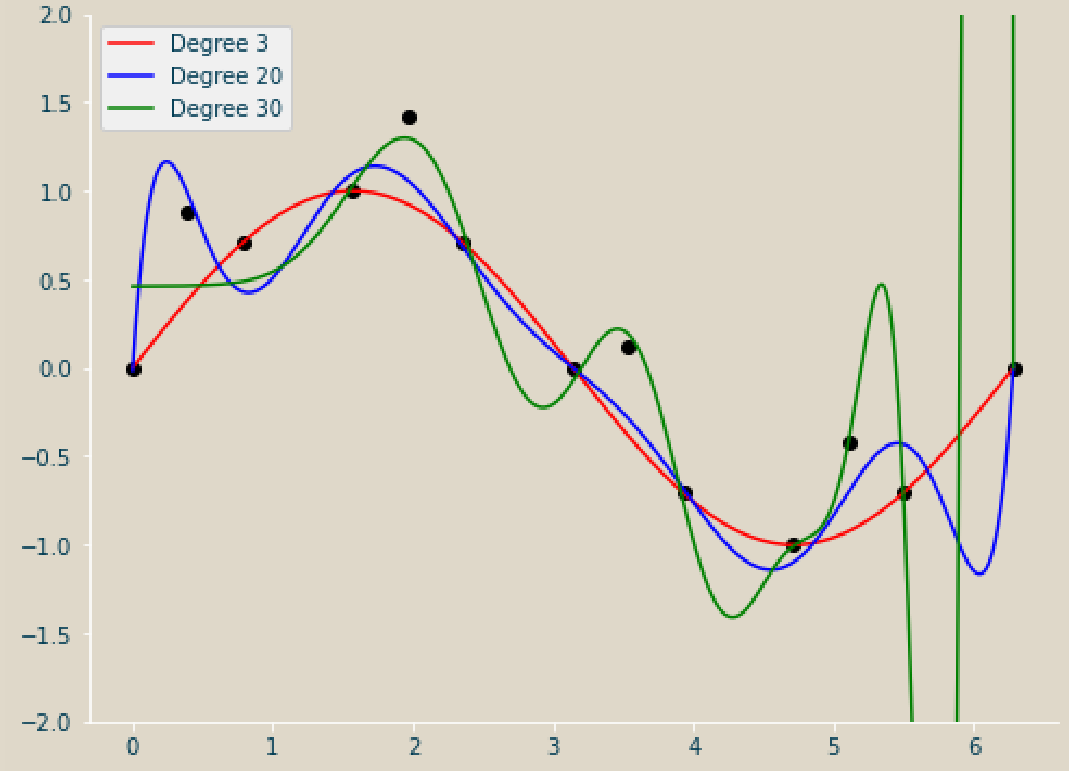
Given multiple possible solutions of the problem **we want to have the simplest**.

Occam's razor: The simplest explanation is usually the best one.

- Instead of tuning the complexity parameters in multiple training operations
  - We let the model **learn the correct complexity during the training**
- Idea: **punish complex model**

$$L(\theta, \mu) = \sum_{i=1}^{n} l(y_i, f(x_i; \theta)) + c\Omega(\theta, \mu)$$

$\theta$ hyperparameters that stay constant during one atomic training run
μ is the complexity parameter
Ω is the cost function for complexity
$c$ constant factor controls the influence of complexity on the loss
$c$ is an additional hyperparameter



# Regularization in linear regression

Given multiple possible solutions of the problem we want to have the simplest.

The **simplest solution** in the case of **linear regression** is **the one with the smallest weights**.

$$L(\theta,\mu) = \sum_{i=1}^{n} l(y_i, f(x_i;\theta)) + \Omega(\theta,\mu)$$

$$\Omega(\theta,\mu) = \Omega(\theta) = \begin{cases} |\theta_0| + \ \ldots + |\theta_d| & \textbf{\textit{Lasso}} \\ \theta_0^2 + \ \ldots + \theta_d^2 & \textbf{\textit{Ridge}} \end{cases}$$

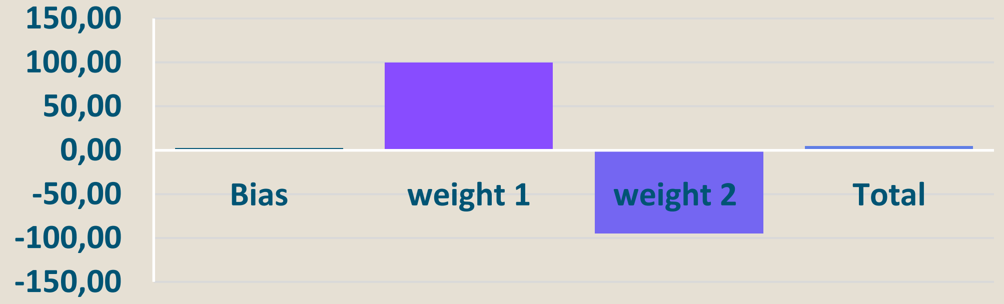
- **Lasso or L1 regularization**: Forces sparsity of the weights
- **Ridge or L2 regularization**: Forces the weights to be small

Model: $\beta_0 + \beta_1 x_1 + \beta_2 x_2$

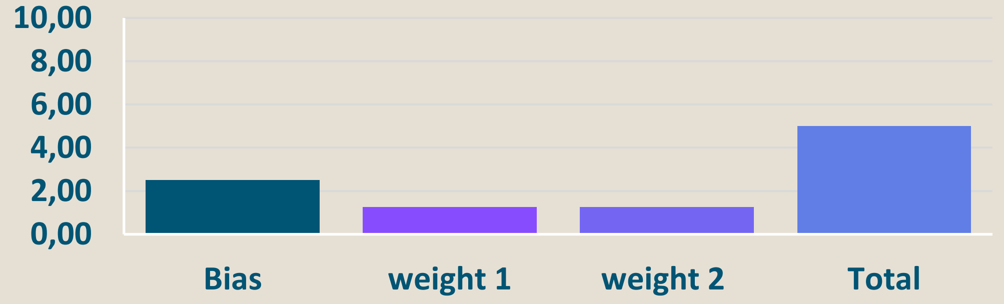
Fit the points $x = (1,1)$ with $y = 5$

There is an infinite number of possible solutions.

E.g.: $2.5 + 100x_1 - 97.5x_2$



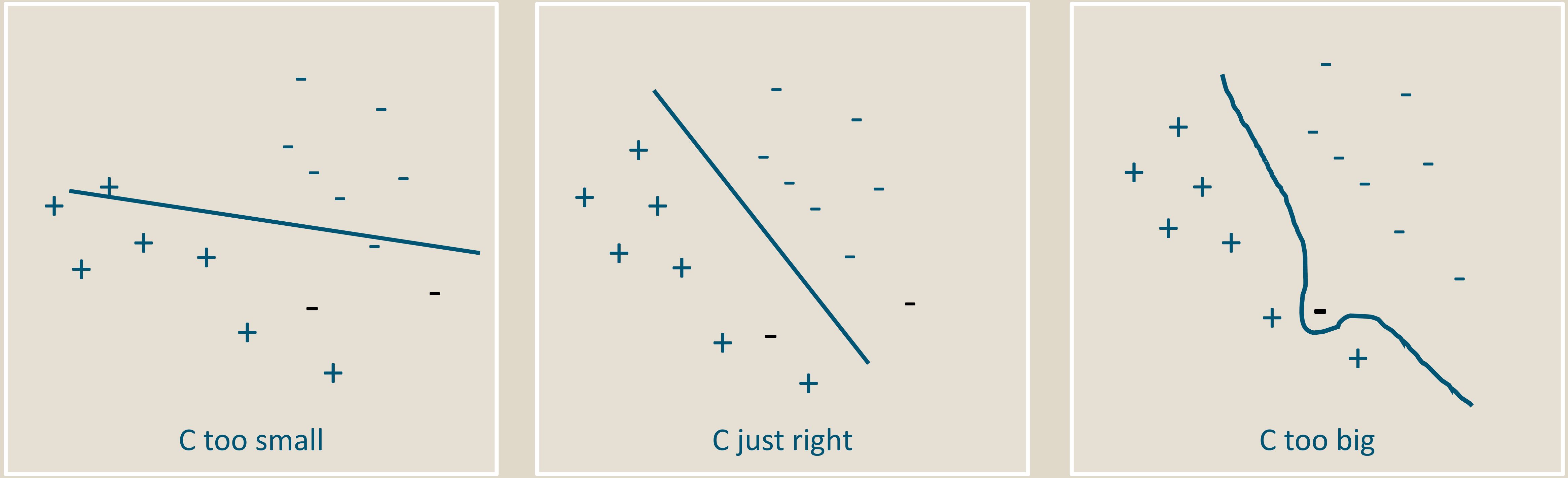
E.g.: $2.5 + 1.25x_1 + 1.25x_2$



# Regularization for SVMs

- **Allow misclassification at a cost**, depending on how large c is

$$\sum_{i=1}^{n} l(y_i, f(x_i;\theta)) + \mathrm{c}\Omega(\theta,\mu)$$



# Regularization in Neural Networks

- Prevent overfitting by **controlling the size of the NN's weights**
  - **L1 norm:** Enforce sparsity in the model weights

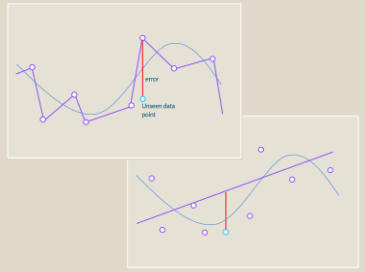
$$\Omega(\theta, w) = \sum_{\text{layer } l} \sum_{\text{input } i} |w_{i,l}|$$

  - **L2 norm:** Force the model weights to stay small

$$\Omega(\theta, w) = \sum_{\text{layer } l} \sum_{\text{input } i} (w_{i,l})^2$$

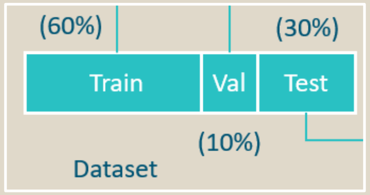
- Prevent overfitting by **controlling the NN's number of neurons**
  - **Drop Out**:
    - In a training step for each neuron with a probability of x% deactivate it
    - Dropout effectively controls the model size

# Important takeaways

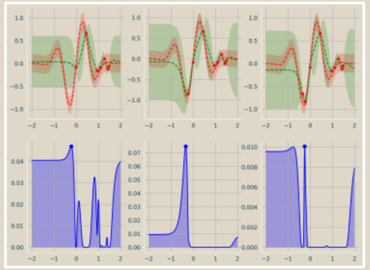


We always try to find a model with suitable complexity. The goal is to achieve a good **tradeoff between bias and variance**.

- **Overfitting** characterizes the problem of high variance, while
- **Underfitting** describes the problem of high bias.



To test **the generalizability** of your model, best practice is to split your data in a **train set** and **test set**. When further optimizing hyperparameters another split for creating a separate **validation set** is required.



Common ways to find the right combination of **hyperparameters**

- Grid search
- Random search
- Bayesian optimization

**Always tune your hyperparamters to find the best fit of your model for the data!**

## Quiz: Evaluation and hyperparameters

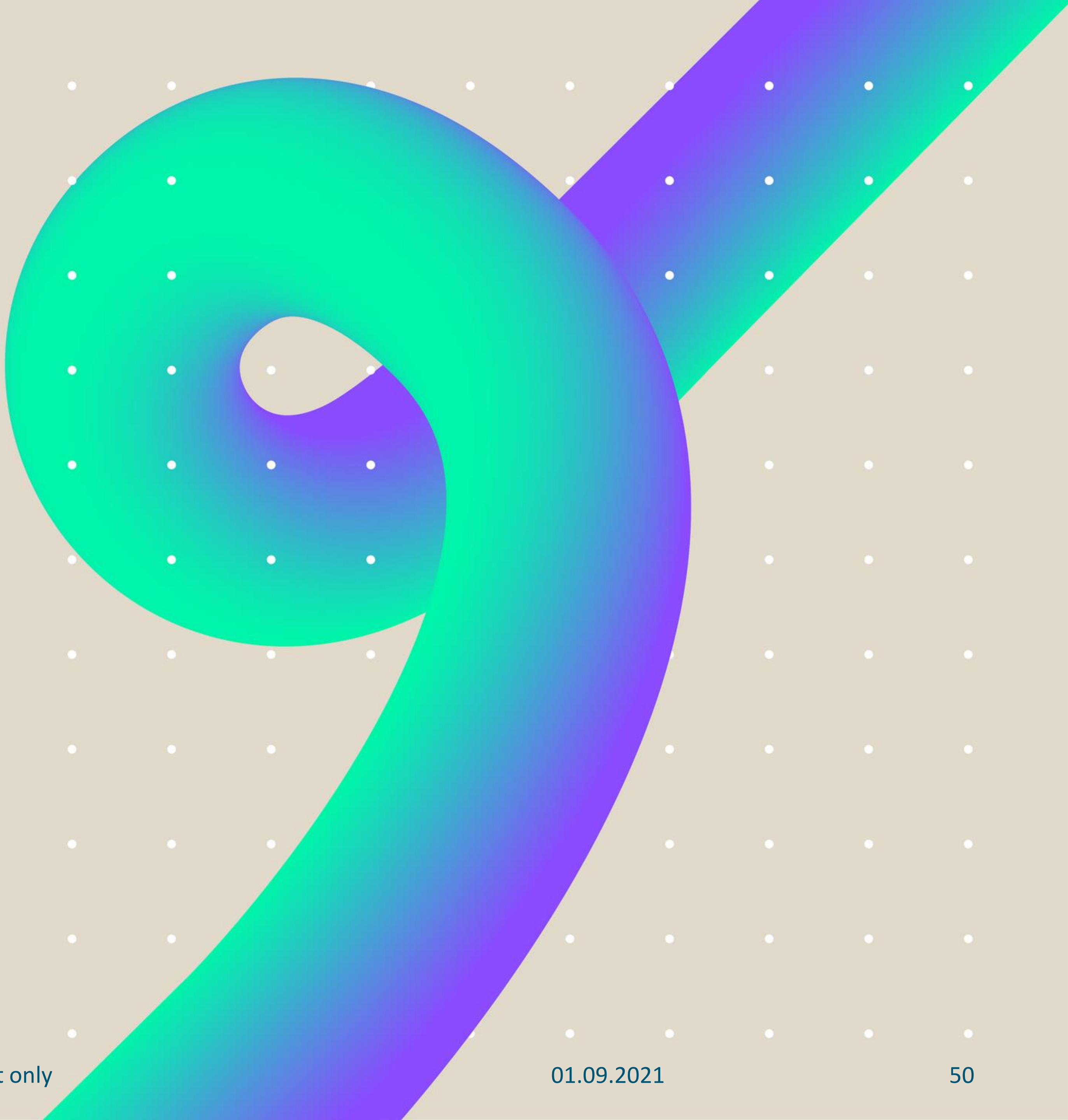
Please join at slido.com with #031 077.

Let’s go through some questions together.

Let’s see what you think. All answers will be anonymous.

## Try it yourself!

## **In the following exercises**



## Feedback and Q&A

# Thank you

If you would like any further
information please contact
Werner,Dr.,Fabian_Georg (BI X) BIX-DE-I
<fabian_georg.werner@boehringer-ingelheim.com>