Cyber-Physical Systems
in Mechanical Engineering TU Berlin

Applied Machine Learning in Engineering

Lecture 08, June 06, 2023

Prof. Merten Stender

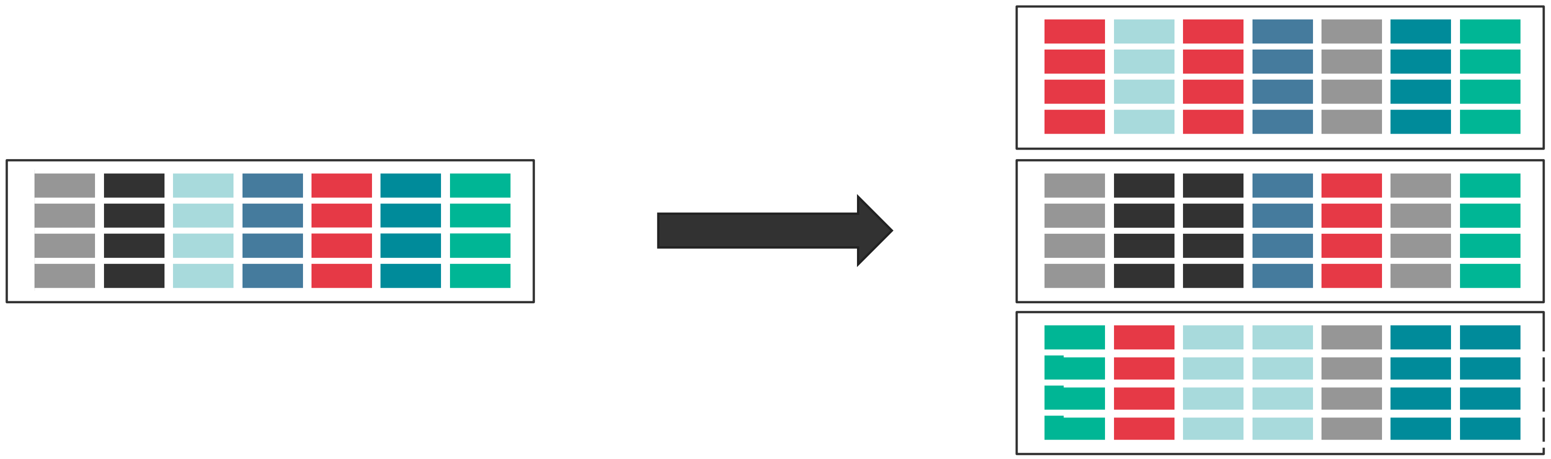
Cyber-Physical Systems in Mechanical Engineering, Technische Universität Berlin

www.tu.berlin/cpsme
merten.stender@tu-berlin.de

Recap: Lecture 07

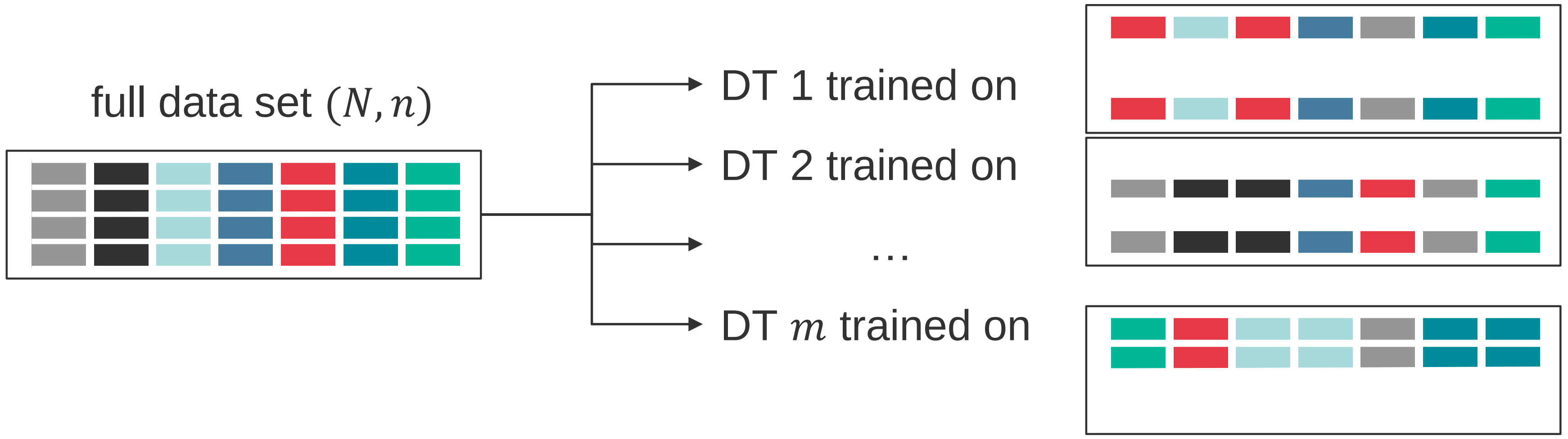
- **Ensemble methods**: combining weak learners to a stronger one
  - Bagging
  - Boosting
  - Random Forests

- **Bagging**: Training m models on m bootstrapped data subsets



Recap: Random Forests

- Combination of bagging with random feature selection



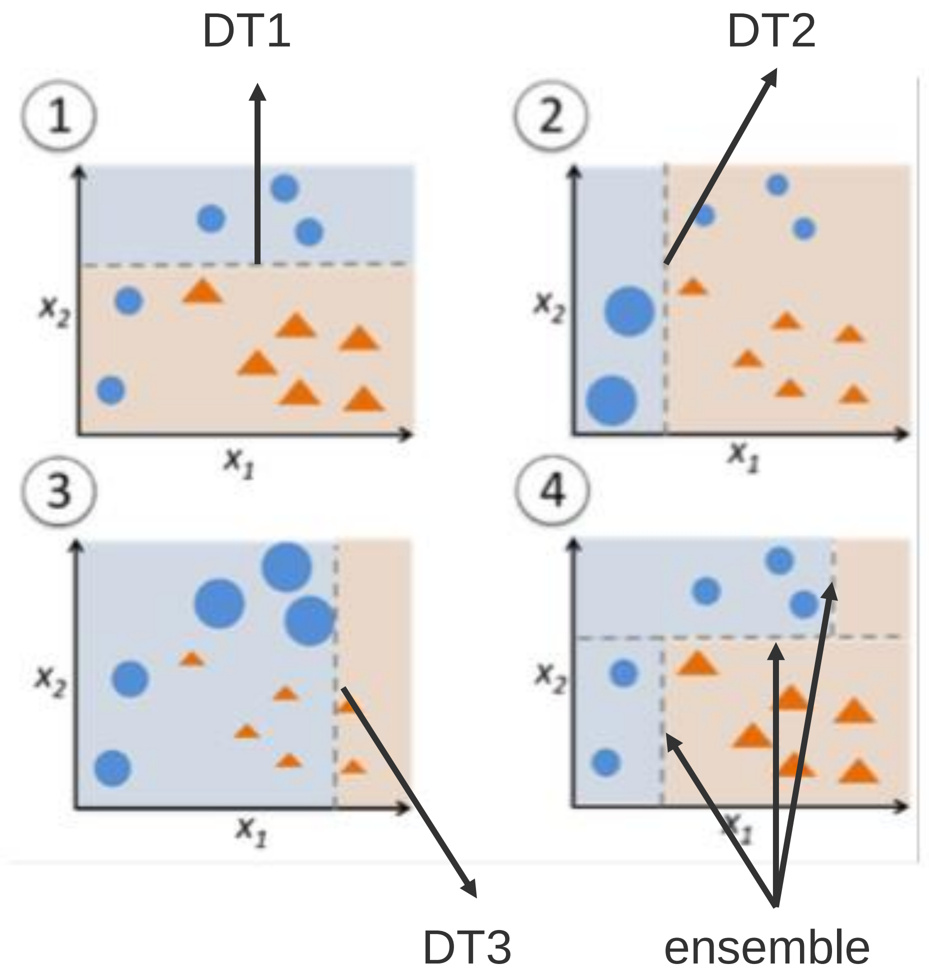
Recap: Boosting

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Weak learners: decision tree (DT) with only 1 splitting rule (two leaf nodes)
  decision tree stump model

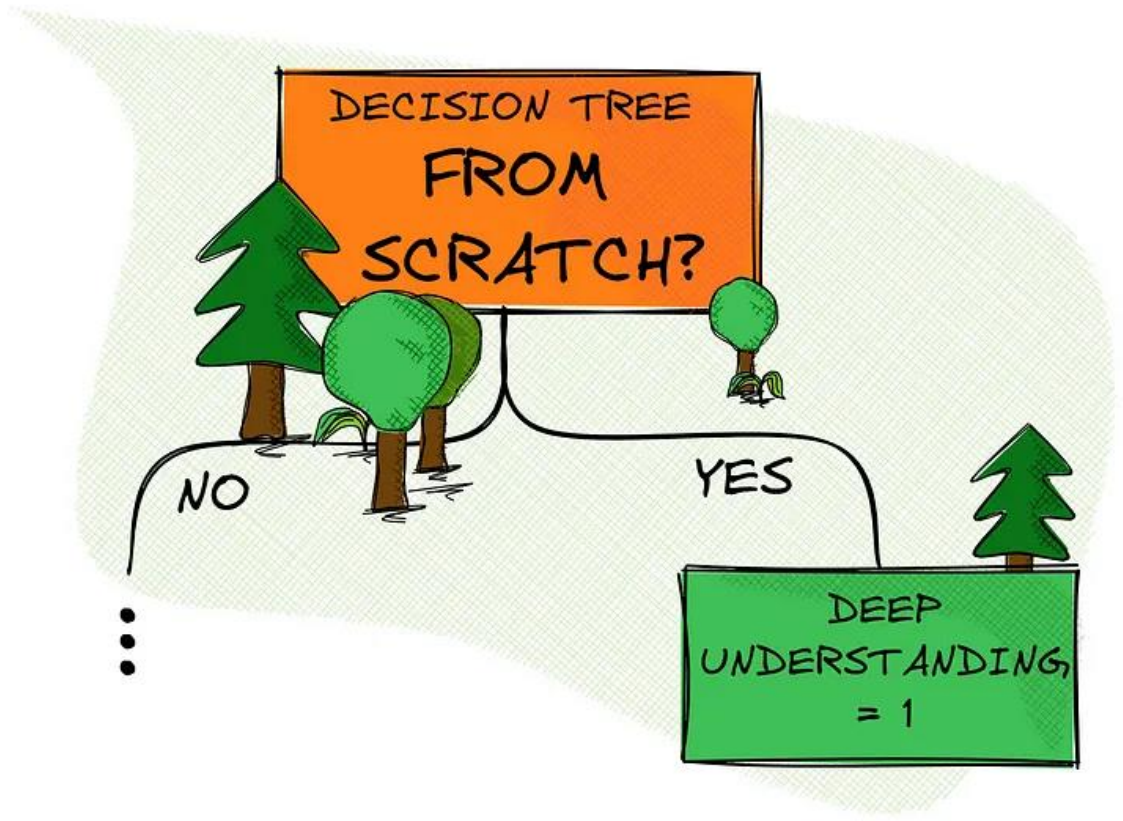
1. Build first tree, find data set regime where is performs weak
2. Put larger weight on mis-classified data points
3. Build second tree using weighted data set

- Continue (without weight reduction)



Recap: Exercise 07

- From scratch implementation of a decision tree for classification
- Recursive functions to grow the tree and traverse data through the tree



© Marvin Lanhenke, https://towardsdatascience.com/implementing-a-decision-tree-from-scratch-f5358ff9c4bb

Recap: Exercise 07

```
# 0.0 base condition (stopping criteria) for recursive call.
# return leaf node if base condition is met
if self.is_completed(depth=curr_depth):
    return Node(prediction=self._majority_vote(y))

# 1. find best split for current data (X, y)
split_dim,split_val = self._best_split(X=X,y=y)

# 2.split the current data according to best split and
# return index for data sent to left and right child node
left_idx = self._create_split(X, split_dim, split_val)
right_idx = ~ left_idx

# 3. create child nodes and assign data using recursive call
child_left = self._grow_tree(X=X[left_idx,:], y=y[left_idx], curr_depth=curr_depth+1)
child_right = self._grow_tree(X=X[right_idx,:], y=y[right_idx] ,curr_depth=curr_depth+1)

# create a node with child nodes for the given spit condition
node = Node(split_dim=split_dim, split_val=split_val, child_left=child_left, child_right=child_right)
```

Today

- Recognize supervised classification tasks
- Understand k-nearest neighbors and logistic regression models
- Evaluate classification prediction models

Agenda

Machine Learning:

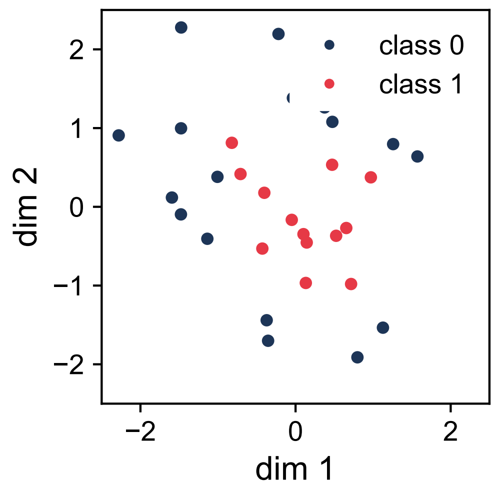
- K-nearest neighbors
- Logistic regression
- Classification metrics
- Data set splitting and k-fold cross validation

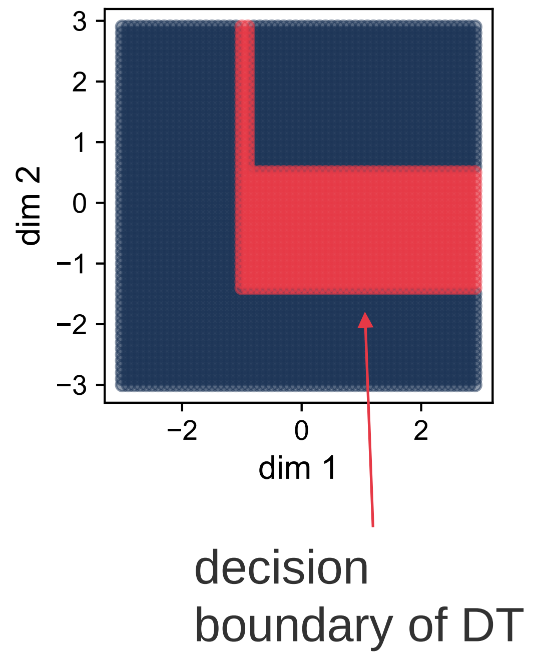
Python:

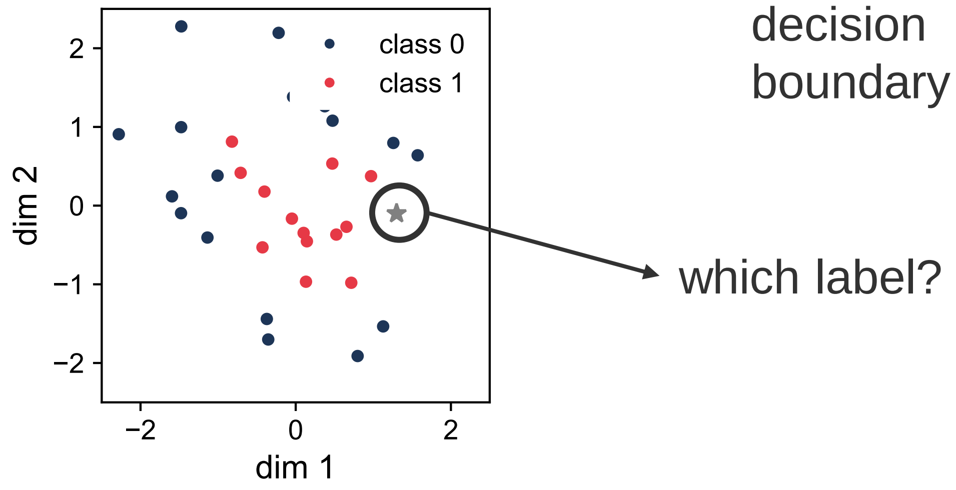
k-nearest neighbors (kNN)

Motivation

- Recap from exercise 07: binary classification in 2-dimensional space
- Decision Tree:
  - Classification of new, unseen data points based on binary feature space segmentation
  - Issue: overfitting
- Can you think of a simpler way to assign labels to new data points?





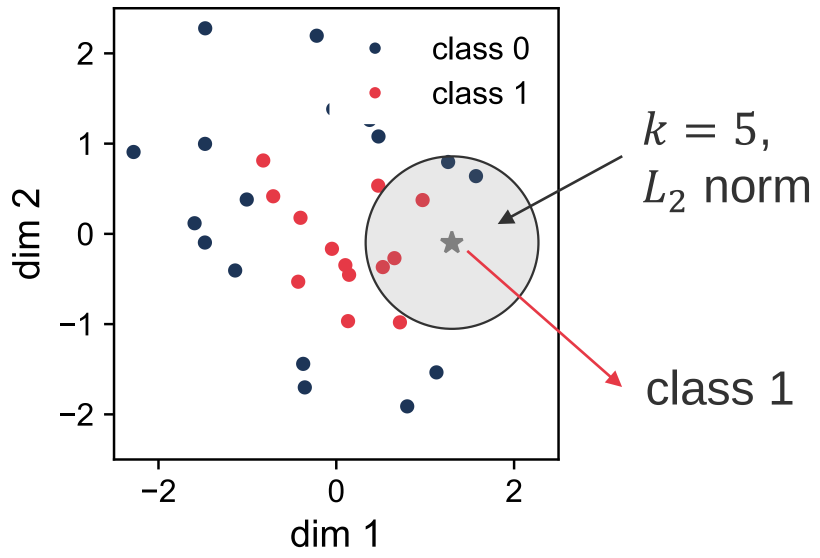


k-nearest neigbor algorithm (classification)

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

Underlying assumption: Two data points in close distance will be likely to belong to the same class

- ***k*-nearest neighbor** (kNN): classification of new data point based on the majority vote across its k nearest neighbors w.r.t. norm $||\cdot||$
- **Lazy learning**: Unparameterized model, no internal model parameters $\boldsymbol{\theta}$ are adjusted to training data samples. Training data is stored and queried for predictions on new data.
- **Hyperparameter**: number of neighbors k
  - Empirical choice of k for balancing under- and overfitting



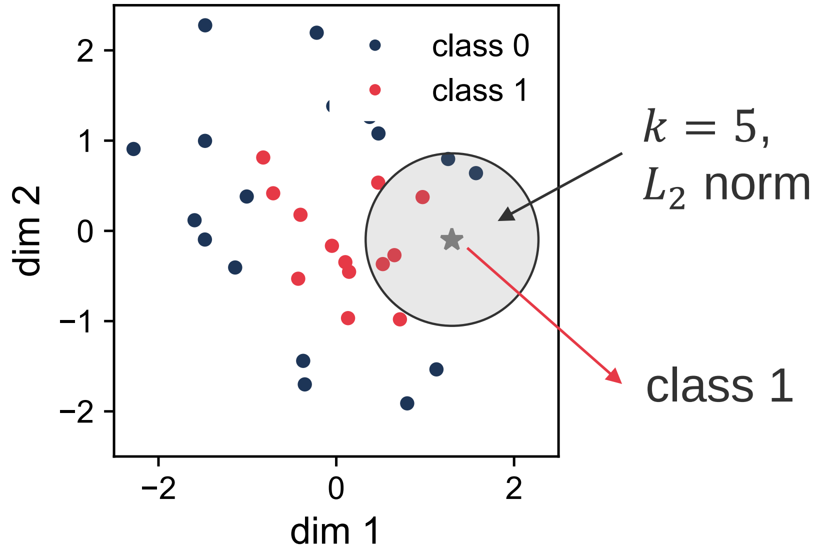
k-nearest neigbor algorithm

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- **Regression task**: return mean / median target value of k nearest neighbors

- (Time) **complexity**:
  - $O(N)$ (trivial approach)
  - $O(\log N)$ efficient kd-tree neighboorhood search

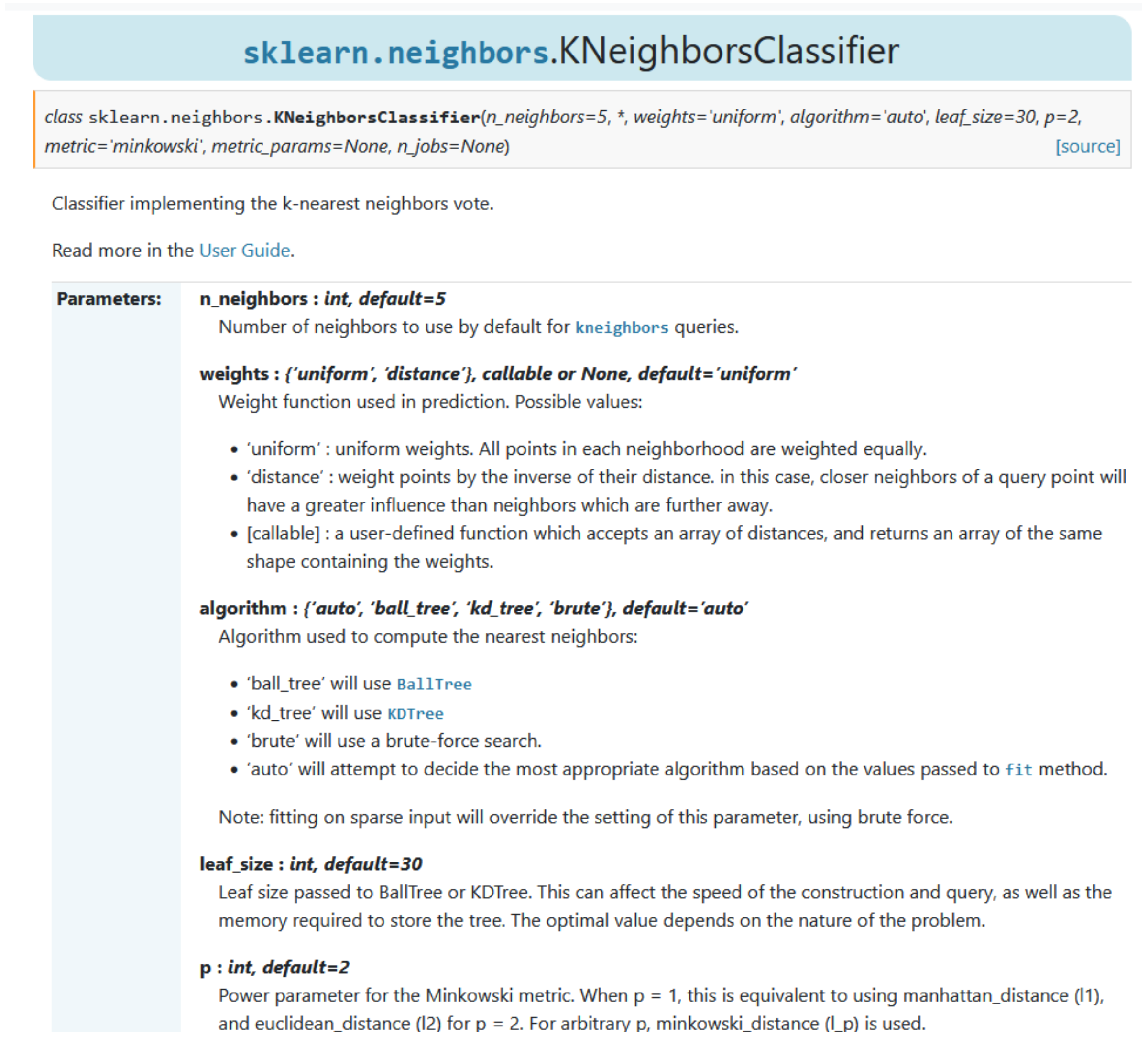
- **Limitations and caviats**
  - High-dimensional data sets: curse of dimensionality
  - Data on different scales: normalization required



- **Extensions**:
  - Probability prediction (classification): return probability instead of majority vote
  - Sample weights based on inverse of distance
  - Sample weights based on importance, e.g. based on trust in training data samples

k-nearest neigbor algorithm

Cyber-Physical Systems
in Mechanical Engineering TU Berlin



sklearn.neighbors.KNeighborsClassifier

class `sklearn.neighbors.`**`KNeighborsClassifier`**(*n_neighbors=5*, ***, *weights='uniform'*, *algorithm='auto'*, *leaf_size=30*, *p=2*, *metric='minkowski'*, *metric_params=None*, *n_jobs=None*) [source]

Classifier implementing the k-nearest neighbors vote.

Read more in the User Guide.

Parameters:

n_neighbors : *int, default=5*
Number of neighbors to use by default for `kneighbors` queries.

weights : *{'uniform', 'distance'}, callable or None, default='uniform'*
Weight function used in prediction. Possible values:

- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

algorithm : *{'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'*
Algorithm used to compute the nearest neighbors:

- 'ball_tree' will use `BallTree`
- 'kd_tree' will use `KDTree`
- 'brute' will use a brute-force search.
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to `fit` method.

Note: fitting on sparse input will override the setting of this parameter, using brute force.

leaf_size : *int, default=30*
Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

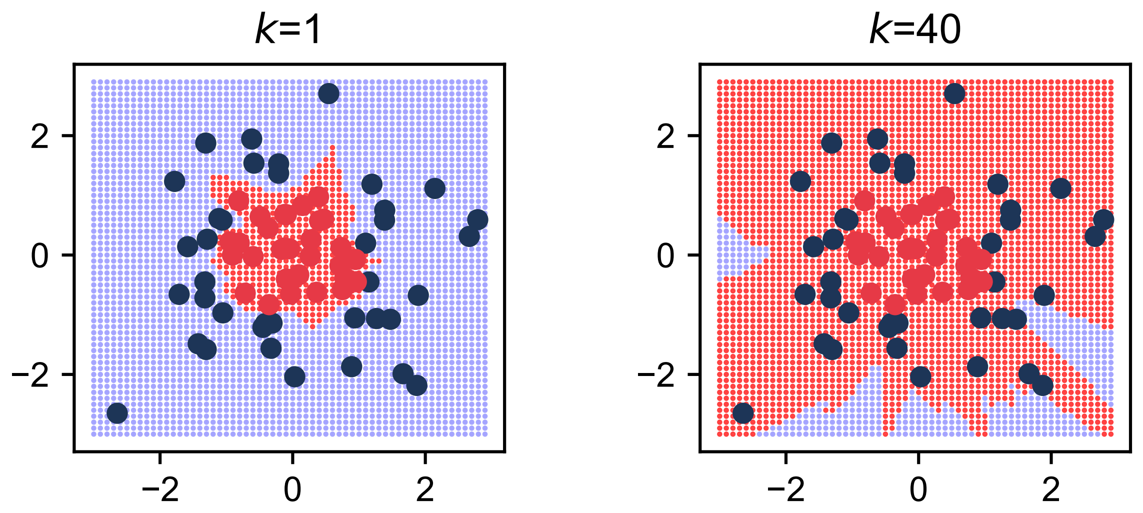
p : *int, default=2*
Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.

kNN: choosing k

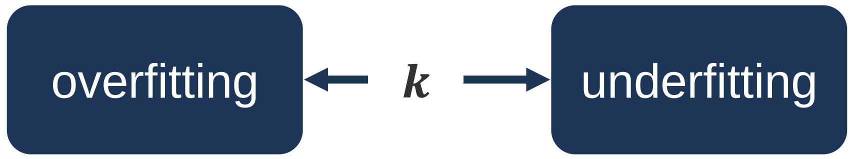
Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Hyperparameter k: number of nearest neighbor to consider

- (Too) small k:
  - High accuracy on training data
  - Low accuracy on validation data
  - Complicated decision boundary
  - Risk of overfitting



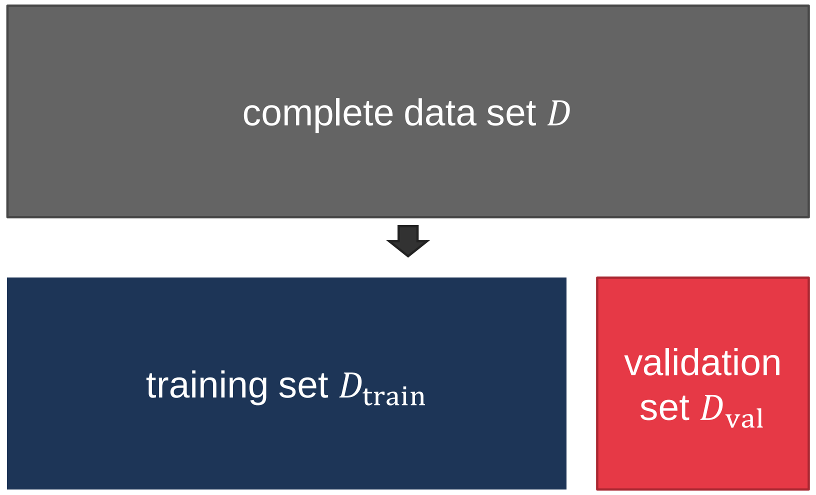
- (Too) large k:
  - Low accuracy on training and validation data set
  - Blurred decision boundaries
  - Risk of underfitting



Training and Validation Data Splitting

- **Aim**: find well-generalizing models with low bias and low variance

- **Generalization property**:
  - Generalization is the act of performing tasks of the same difficulty and nature
  - In data-driven modeling: predictive model performs well on previously unseen data


complete data set D
training set D_{train}
validation set D_{val}

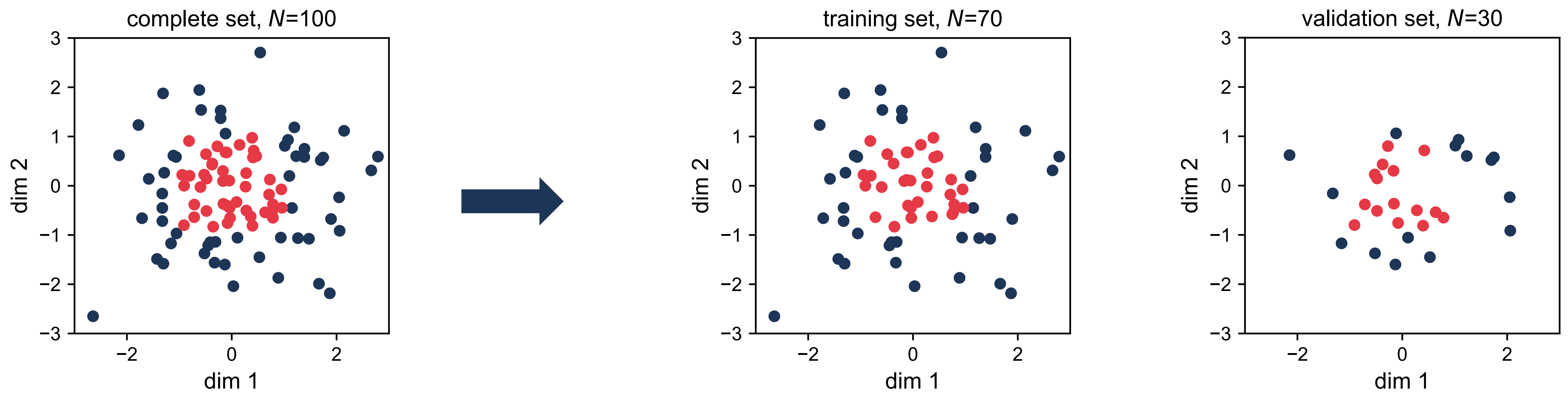
Typical procedure in machine learning:
1. Split data into training and validation set (typically at ratio of 0.7 / 0.3 or 0.8 / 0.2)

2. Find best model parameters θ on training set D_{train}

3. Evaluate performance on hold-out validation set D_{val}

Example

- Train-validation split through random sampling (without replacement)



- Potential risk: different data distributions in training and validation set
  - Check data split distributions, particularly for small data sets
  - Consider using stratified splits, particularly for imbalanced data sets

Measuring Classification Performance

- To choose an appropriate k for kNN, we need some quality metric for how many of the data points got classified correctly
- Straight-forward metric: **accuracy** $\mathrm{acc} = \frac{\mathrm{TP+TN}}{\mathrm{TP+TN+FP+FN}}$ ratio of correctly predicted samples

- **TP**: number of true positives $y = 1, \hat{y} = 1$
- **FP**: number of false positives $y = 0, \hat{y} = 1$
- **TN**: number of true negatives $y = 0, \hat{y} = 0$
- **FN**: number of false negatives $y = 1, \hat{y} = 0$

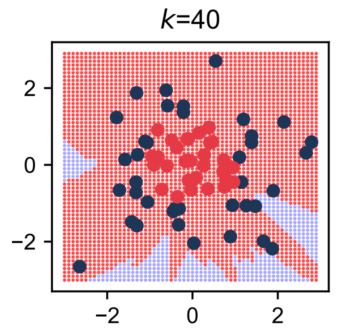
ground truth	y
prediction	$\hat{y}$

- Positives/negatives: depends on the application and the perspective. Typically: positive class is the quantity you want to predict / detect.
- **Example**: Structural health monitoring, prediction of cracks in concrete structures.
  - Positive class: crack exists
  - Negative class: no crack

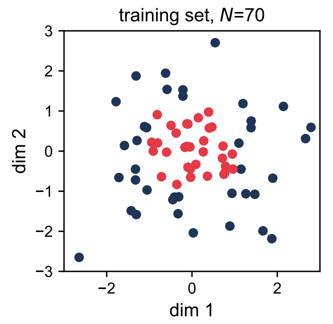
Which are the most critical false predictions?

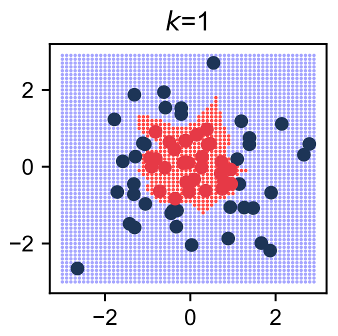
Over- and Underfitting

- **Underfitting**: prediction quality is low in training and validation data set
  - Model capability is not matching the prediction task complexity



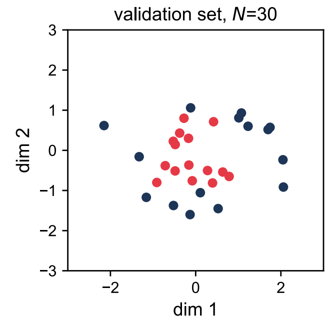
- **Overfitting**: high prediction quality on training set, weak quality on validation set
  - Model is too complex for the given data set
  - Data set is too small / has not enough variance / is strongly imbalanced
  - Model has picked up all peculiarities of the training set, but no meaningful underlying patterns that are required for new unseen data





Performance on training set: $\mathrm{acc} = 1.0$

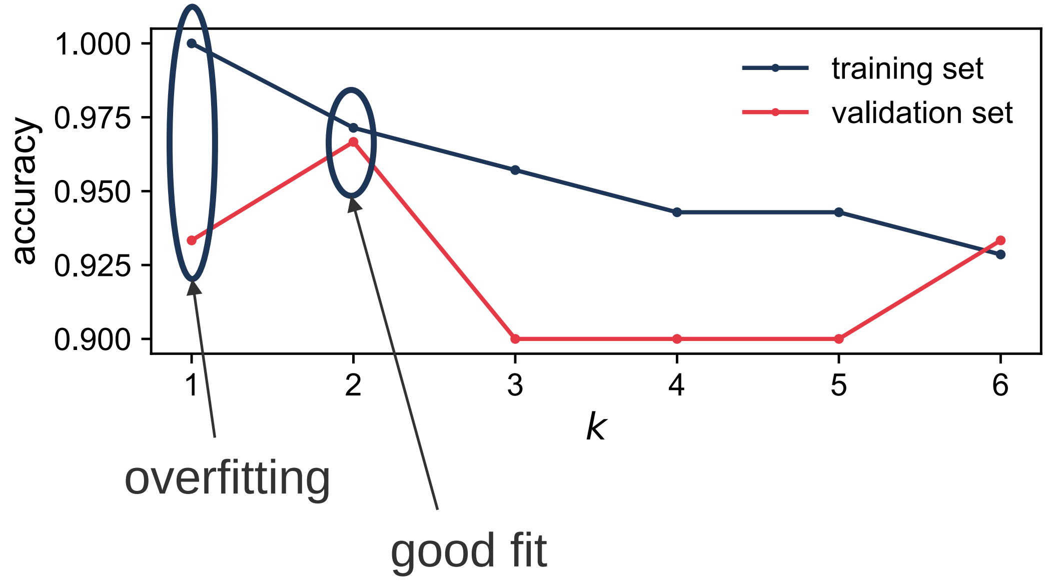
Performance on test set: $\mathrm{acc} = 0.933$

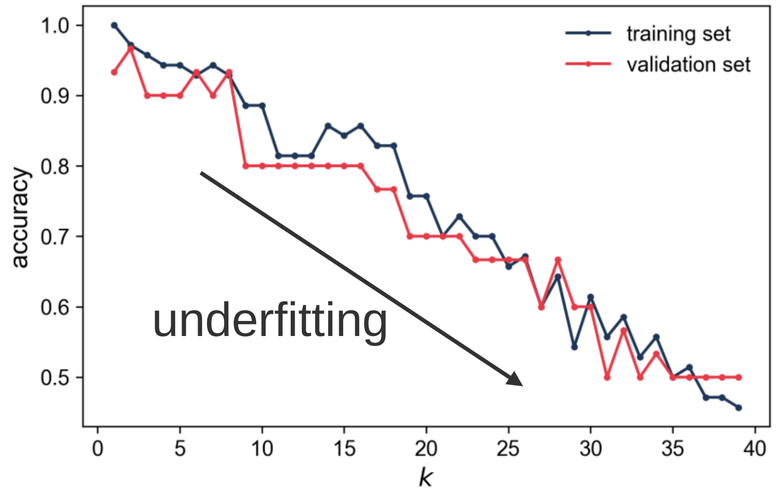


kNN: choosing k

- Edge cases:
  - $k = 1$ accuracy 1.0 on training data set
  - $k = N$ accuracy $\hat{N}/_{N}$ on training data set, $\hat{N}$ number of majority class members

- Different choices of k: $k \in \{1, \dots, 40\}$



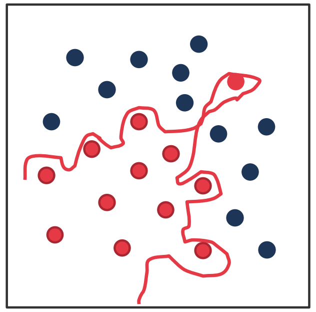


Bias Variance Tradeoff

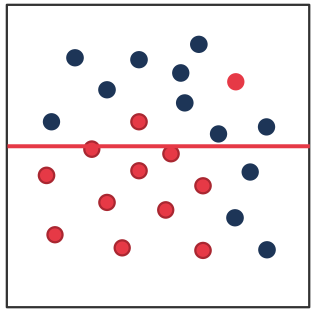
Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Setting: train different versions of a given model on different training data sets (obtained from the same original set, e.g. through bootstrapping). Every version of the model will make different predictions on a hold-out validation set

- **Variance**: a measure how far each model‘s predictions are from the average across all models
  - Large variance: model versions perform vastly different on different data sets
  - Caused by high flexibility of the model, potentially over-adjusting to the specific training data set
  - Large variance $\rightarrow$ **overfitting**



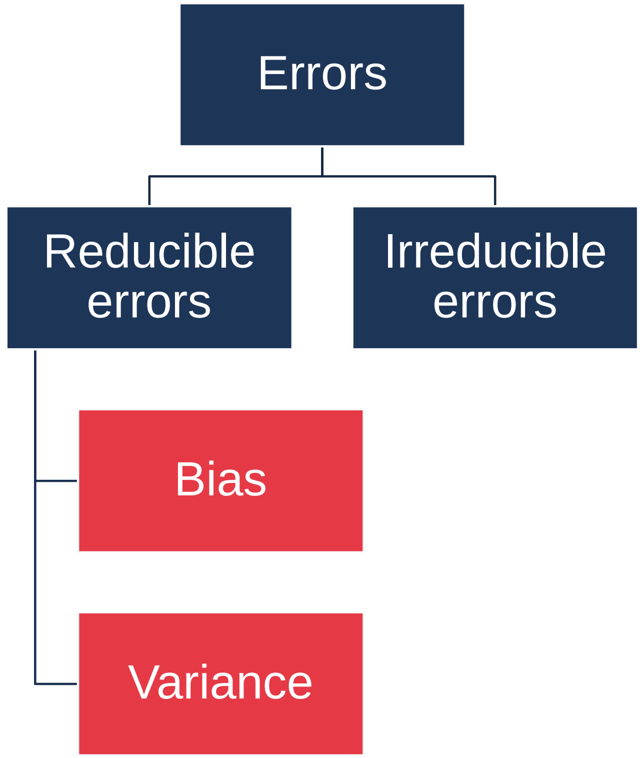
- **Bias**: **Model inable to capture the true relationship between inputs and targets**
  - A measure how far the average model prediciton is from the ground truth
  - Large bias: on average, all model versions deviate strongly from the ground truth
  - Caused by a model that is unable to capture the dominant patterns in the data
  - Large bias: $\rightarrow$**underfitting**

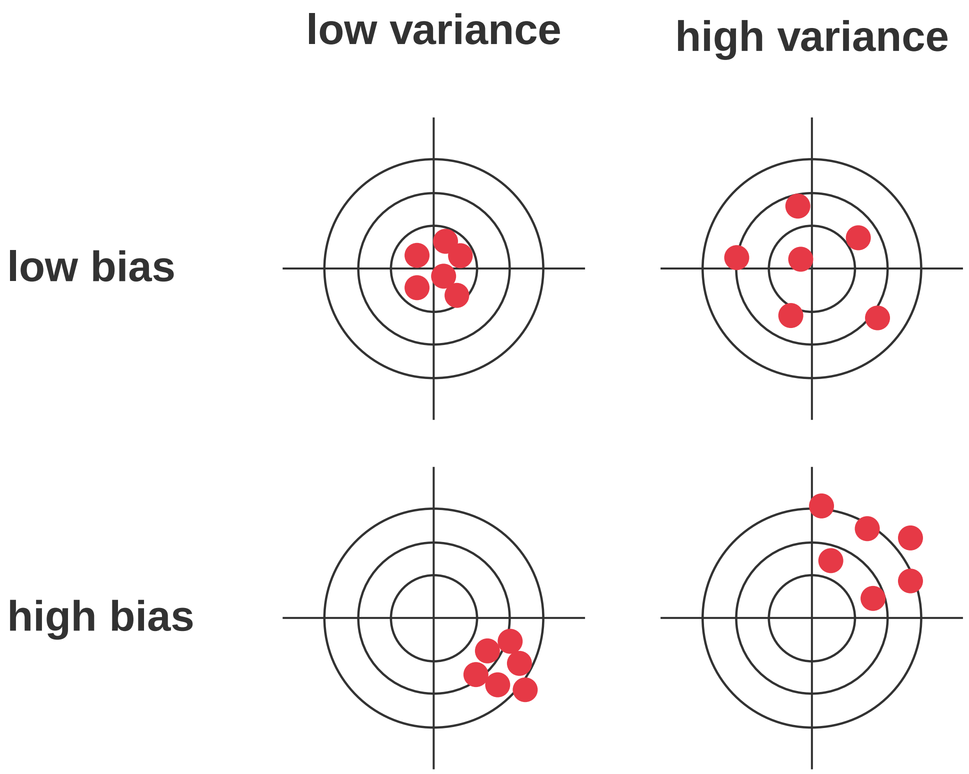


Bias Variance Tradeoff

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Bias and variance are **reducible errors** in data-driven modeling





Bias Variance Trade-Off

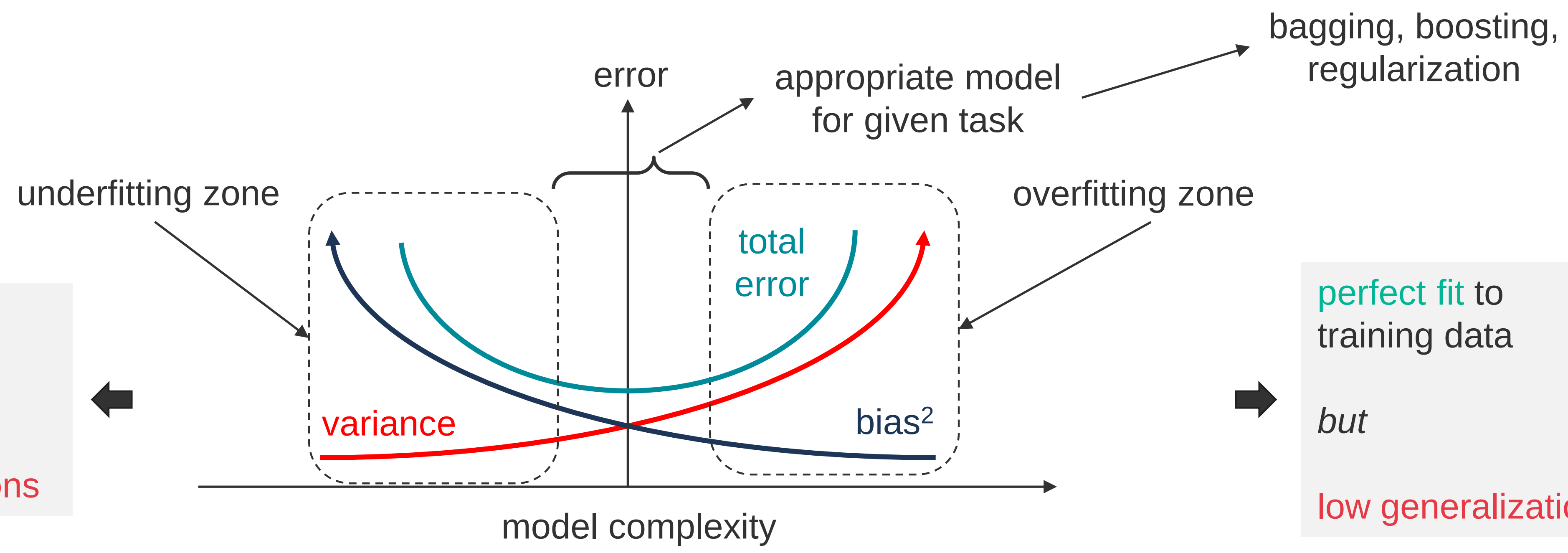
Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Reducible errors, **but** there is a **tradeoff** between variance and bias

consistent

but

weak predictions



perfect fit to training data

but

low generalization

- A biased model can perform better on the validation data than a model that was overfitted to the training data, hence finding an optimal model can be a tradeoff

Bias and Variance: How to Proceed

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Typical approach to machine learning modeling tasks

1. **Start with a very simple model** (*baseline* model) with high bias
2. **Increase model complexity until acceptable bias obtained**
3. **Validation set evaluation**. In case of high variance:
   a. Try to get more data or better data (e.g. data augmentation)
   b. Make changes to the model architecture (regularization)
   c. Try different model architecture and continue with (2)

Reduce bias

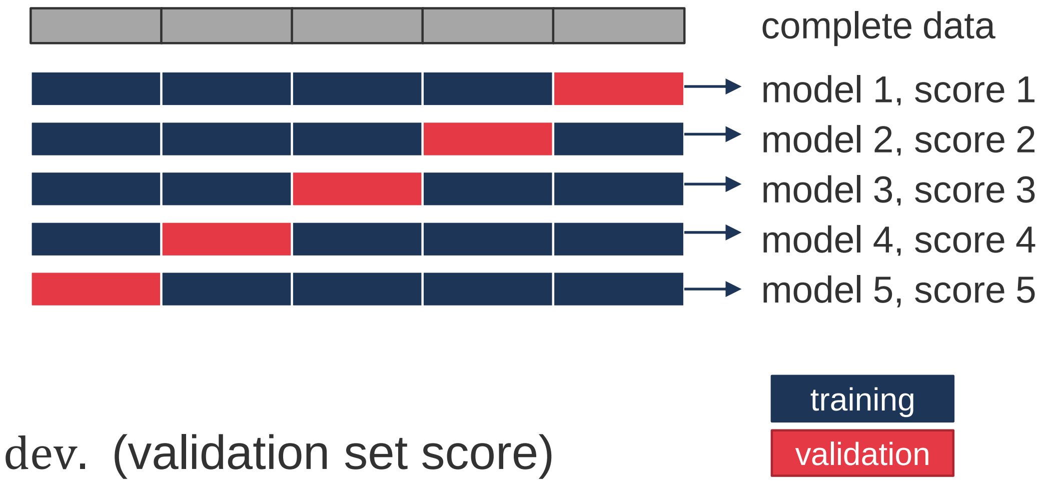
⬇

Reduce variance

→ **Diagnose the problem first, then take actions!**

k-fold Cross Validation

- **Procedure:**
  1. Split data set into k subsets
  2. Use $(k-1)$ subsets as training set, and remaining subset as validation set
  3. Repeat by iterating over all subsets

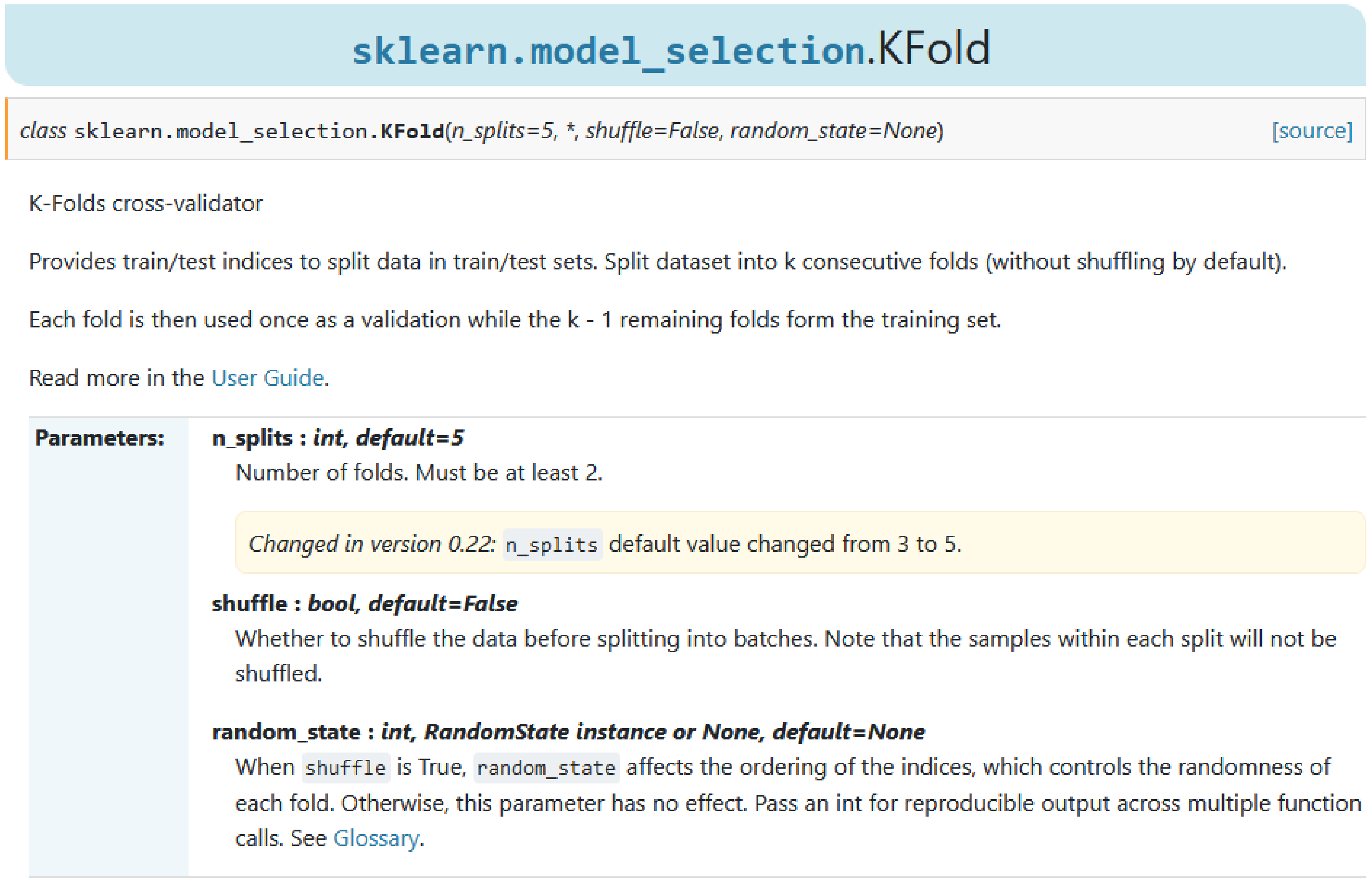


- Report of evaluation metric: $\text{score} = \text{mean} \pm \text{std. dev.}$ (validation set score)

- **When to use**?
  - Comparatively small data set with strong data spread
  - Simple train-test split yields non-repeatable results (within some tolerance)
  - Measure bias and variance statistics of your model
  - Compare different model architectures against each other

k-fold Cross Validation

Cyber-Physical Systems
in Mechanical Engineering TU Berlin



sklearn.model_selection.KFold

class `sklearn.model_selection.`**`KFold`**(*n_splits=5*, ***, *shuffle=False*, *random_state=None*) [source]

K-Folds cross-validator

Provides train/test indices to split data in train/test sets. Split dataset into k consecutive folds (without shuffling by default).

Each fold is then used once as a validation while the k - 1 remaining folds form the training set.

Read more in the User Guide.

Parameters:

n_splits : *int, default=5*
Number of folds. Must be at least 2.

Changed in version 0.22: `n_splits` default value changed from 3 to 5.

shuffle : *bool, default=False*
Whether to shuffle the data before splitting into batches. Note that the samples within each split will not be shuffled.

random_state : *int, RandomState instance or None, default=None*
When `shuffle` is True, `random_state` affects the ordering of the indices, which controls the randomness of each fold. Otherwise, this parameter has no effect. Pass an int for reproducible output across multiple function calls. See Glossary.

Cross Validated Choice of k

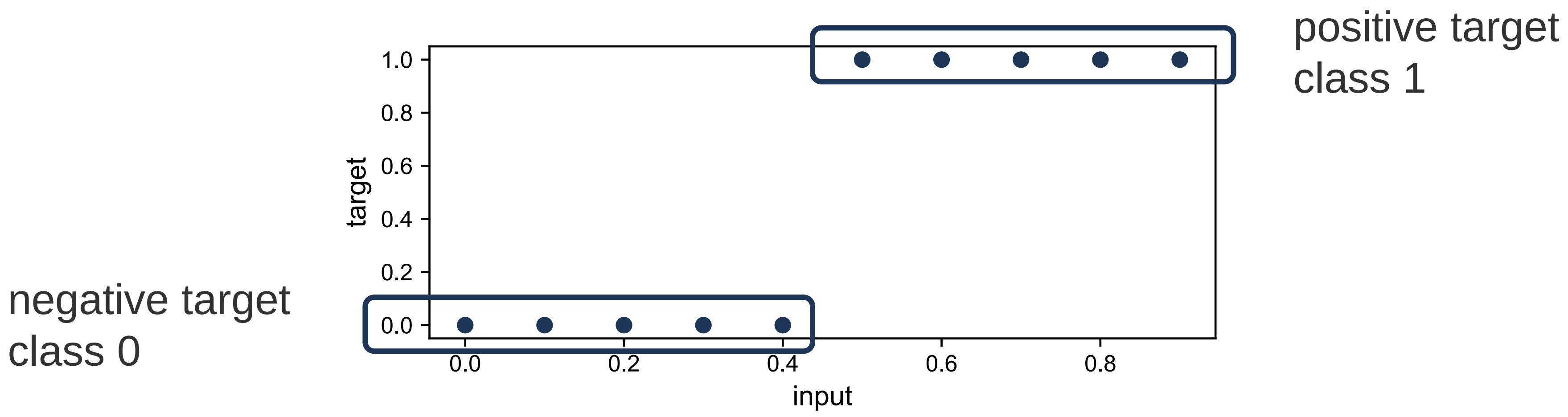
- Using 5-fold cross validation for selecting an optimal value of k. Validation scores:

k	mean accuracy	std. dev accuracy	bias / variance
1	0.94	0.058	low bias, high variance
2	**0.95**	**0.032**	**low bias, low variance**
3	0.93	0.051	low bias, high variance
4	0.94	0.049	low bias, high variance
5	0.90	0.550	high bias, high variance
10	0.91	0.037	high bias, low variance
20	0.78	0.040	high bias, high variance

Logistic Regression

Logistic Regression

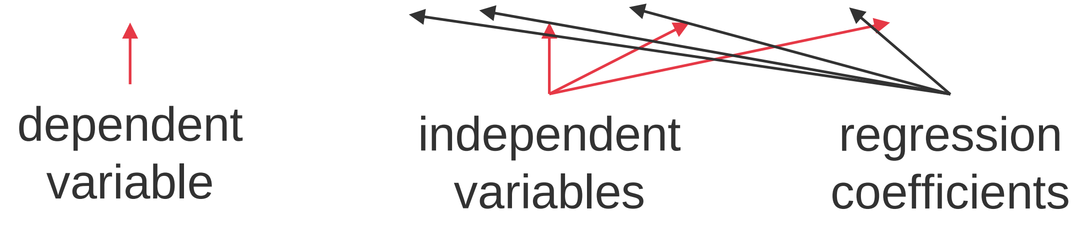
- Simple **classification method**
- Why ‚regression'? We are in fact predicting probabilities $[0, 1]$ and not categories
- Based on maximum likelyhood
- Different names: *logit regression*, *maximum-entropy classification* or *log-linear classifier*



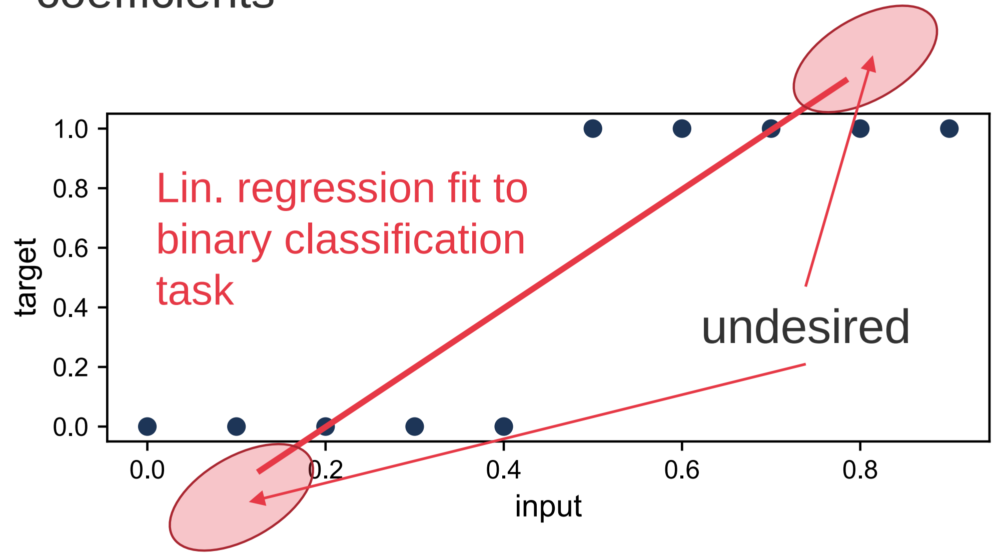
Logistic Regression

- Logistic regression: estimate the propability of observing a positive label given the current inputs

- **Linear regression:** $\hat{y}_i = \mathbf{x}_i^\top \boldsymbol{\theta} = \theta_0 + \theta_1 \boldsymbol{x}_1 + \theta_2 \boldsymbol{x}_2 + \cdots \theta_n \boldsymbol{x}_n$



- **Logistic regression:**
  - Dependent variable on nominal scale
  - Linear regression would estimate the value itself, not the probability of observing the positive class
  - Aim: prediction of $P(y = 1| \boldsymbol{x})$, i.e. the probability of the positive (1) target



Logistic Regression

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Logistic regression: estimate the propability of observing a positive label given the current inputs $\boldsymbol{x}$

- **Linear regression:** $\hat{y}_i = \mathbf{x}_i^\top \boldsymbol{\theta} = \theta_0 + \theta_1 \boldsymbol{x}_1 + \theta_2 \boldsymbol{x}_2 + \cdots \theta_n \boldsymbol{x}_n$

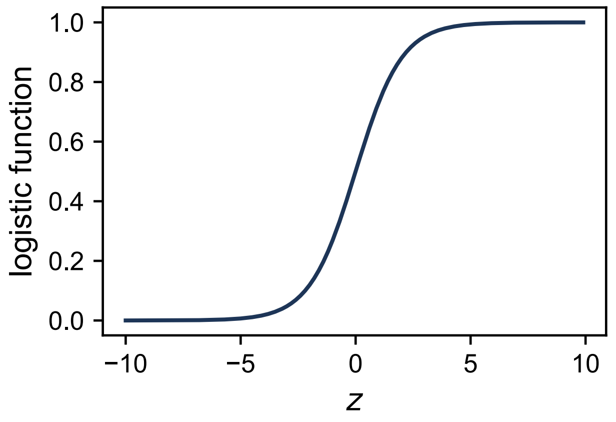
- **Logistic regression:**
  - Restrict output range to $[0, 1]$ using sigmoid

- Idea: take the logistic of the linear regression:

$$\hat{y} = \frac{1}{1+e^{-(\theta_0+\theta_1 x_1+\theta_2 x_2+\cdots\theta_n x_n)}}$$

Logistic function (sigmoid function)

$$f(z) = \frac{1}{1+e^{-z}} \in [0,1]\ \forall z$$



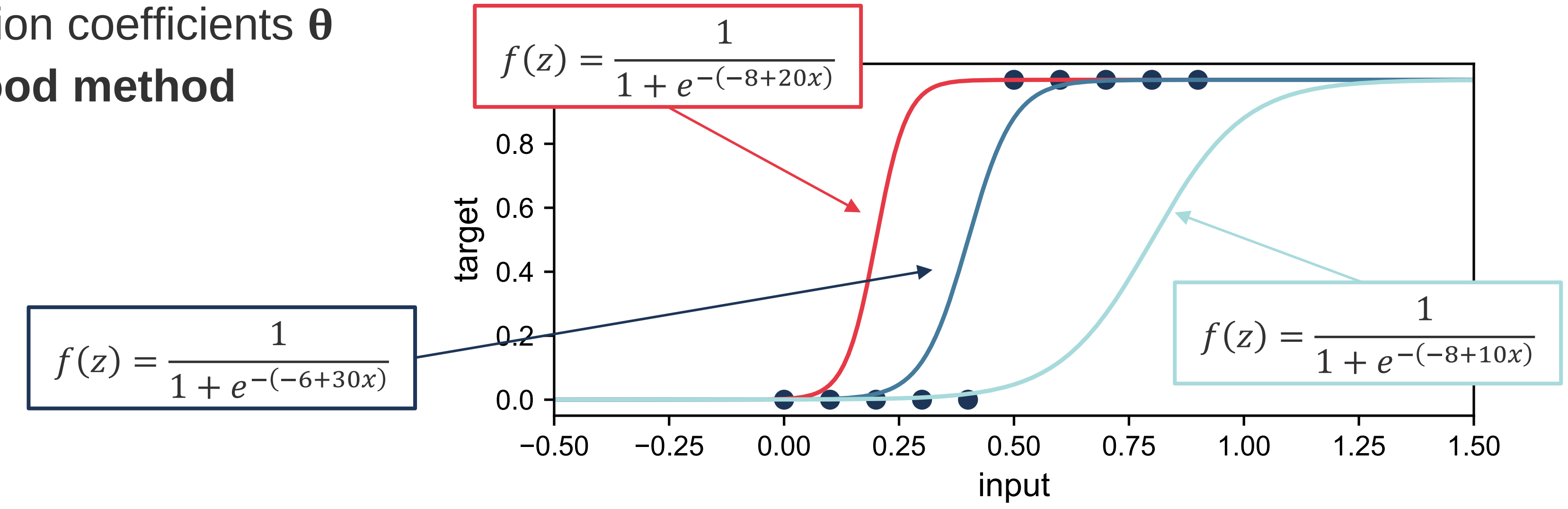
Logistic Regression

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- **Logistic regression:** $\hat{y} = \frac{1}{1+e^{-(\theta_0+\theta_1 x_1+\theta_2 x_2+\cdots\theta_n x_n)}}$

- Probability of observing the positive class: $P(y=1|\, x_1, x_2, \dots x_n) = \frac{1}{1+e^{-(\theta_0+\theta_1 x_1+\theta_2 x_2+\cdots\theta_n x_n)}}$
- Probability of observing the negative class: $P(y=0|\, x_1, x_2, \dots x_n) = 1 - \frac{1}{1+e^{-(\theta_0+\theta_1 x_1+\theta_2 x_2+\cdots\theta_n x_n)}}$

- Computation of regression coefficients $\boldsymbol{\theta}$
  - **Maximum likelyhood method**



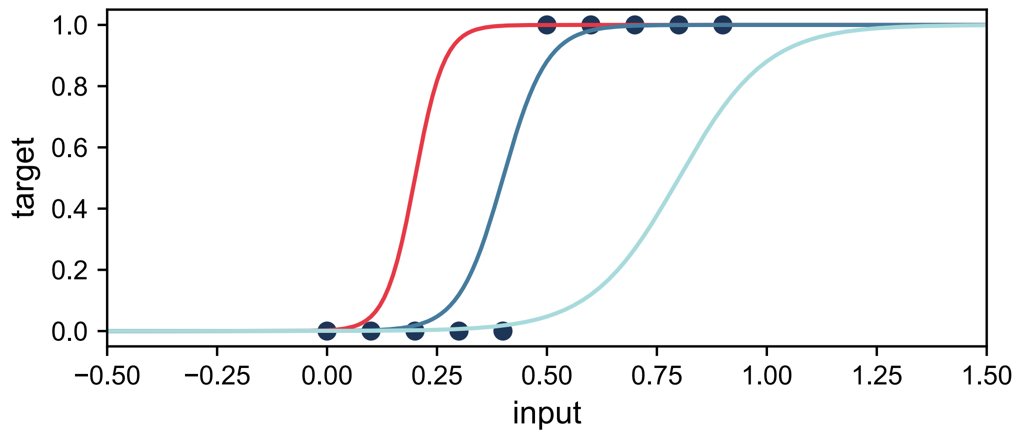
Maximum Likelyhood

- Likelyhood function $L(\boldsymbol{\theta})$ denoting the probability of observing the positive class for a given input $\boldsymbol{x}$

- **Maximum likelyhood estimator:** maximize log likelyhood $LL(\boldsymbol{\theta}) = \log(L(\boldsymbol{\theta}))$

→ Minimize $LL(\boldsymbol{\theta}) = \sum_{i=1}^{N}(-y_i \log(\hat{p}(\boldsymbol{x}_i)) - (1 - y_i)\log(1 - \hat{p}(\boldsymbol{x}_i)))$

- **Methods for finding optimal $\boldsymbol{\theta}^*$:** nonlinear optimizers
  - Stochastic gradient descend (SGD)
  - Newton-Cholesky
  - Newton-Conjugate Gradient
  - …



Exercise 08

May 31, 2023

Exercise 08

- Implement k-nearest neighbors from scratch
  - Object-oriented implementation
  - .fit(X, y), .predict(X) methods

- Implement classification scores
  - Accuracy
  - Recall
  - Precision

- Evaluate tendency to under- and overfitting for
  - kNN,
  - decisionTrees
  - logistic regression

Questions?