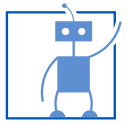


Robotics, Artificial Intelligence and Embedded Systems

Cognitive Systems: Neural Networks and Applied AI

Exercise 2

Basic Concepts in Machine Learning

Prof. Dr.-Ing. habil. Alois Knoll
Florian Walter, M.Sc.

Summer Semester 2019

Topics in this Chapter

What is machine learning…
… and how can a system learn from a teacher?

- What is (machine) learning?
- What different types of datasets do exist?
- Which learning paradigms do result from these types?
- How does the Perceptron learning rule work?
- What is a common method for supervised learning?

What is (Machine) Learning?

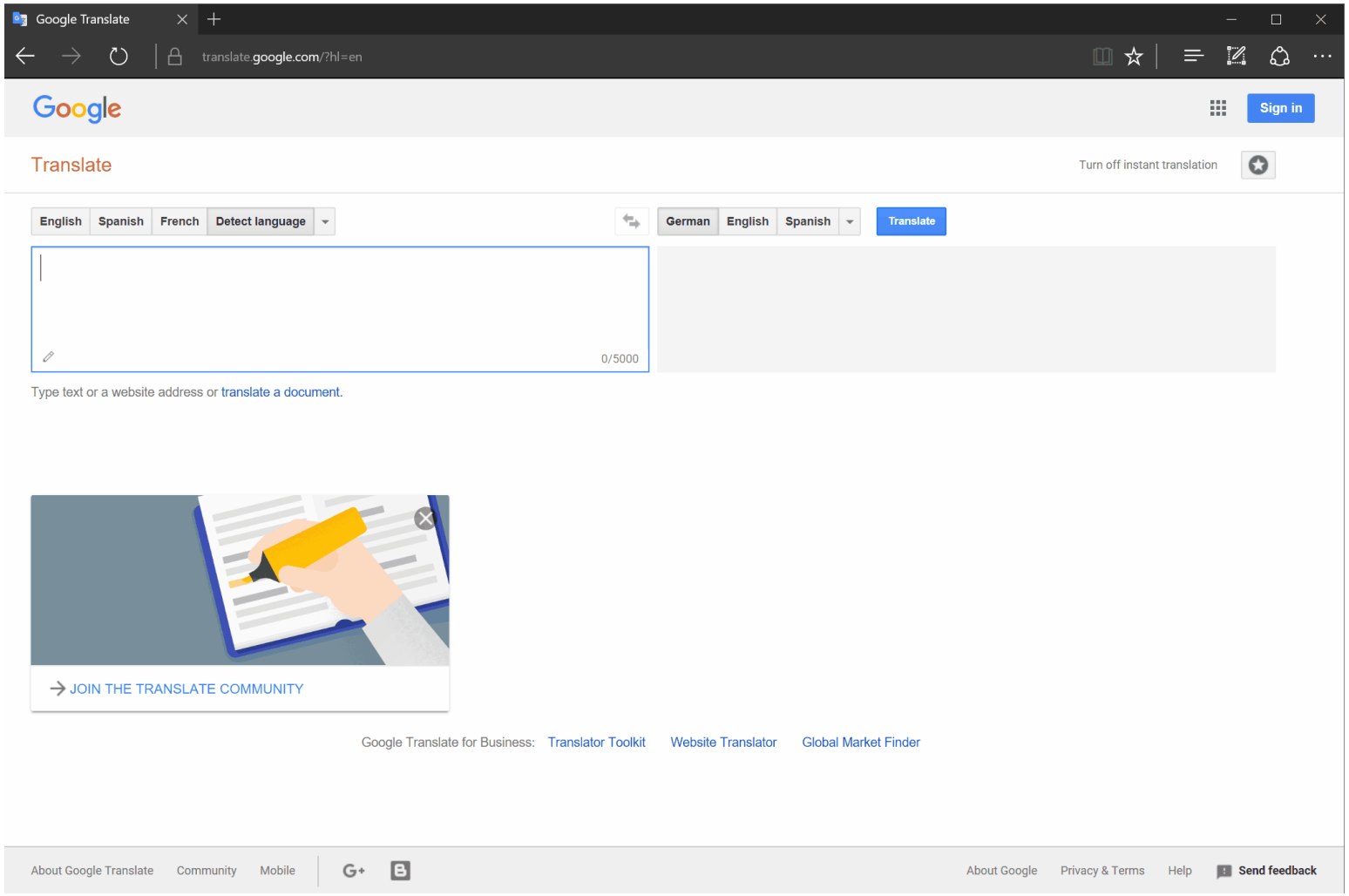
Learning – Definition

Definition E4.1: Learning

Learning is the acquisition of new information or knowledge or the process to acquire knowledge or skill by systematic study or by trial and error.

Adapted from the Encyclopedia of Neuroscience

Is this System Based on Machine Learning?



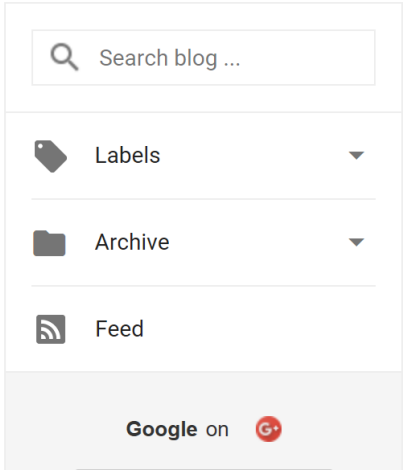
Is this System Based on Machine Learning? Yes!

Zero-Shot Translation with Google's Multilingual Neural Machine Translation System

Tuesday, November 22, 2016

Posted by Mike Schuster (Google Brain Team), Melvin Johnson (Google Translate) and Nikhil Thorat (Google Brain Team)

In the last 10 years, Google Translate has grown from supporting just a few languages to 103, translating over 140 billion words every day. To make this possible, we needed to build and maintain many different systems in order to translate between any two languages, incurring significant computational cost. With neural networks reforming many fields, we were convinced we could raise the translation quality further, but doing so would mean rethinking the technology behind Google Translate.



https://research.googleblog.com/2016/11/zero-shot-translation-with-googles.html

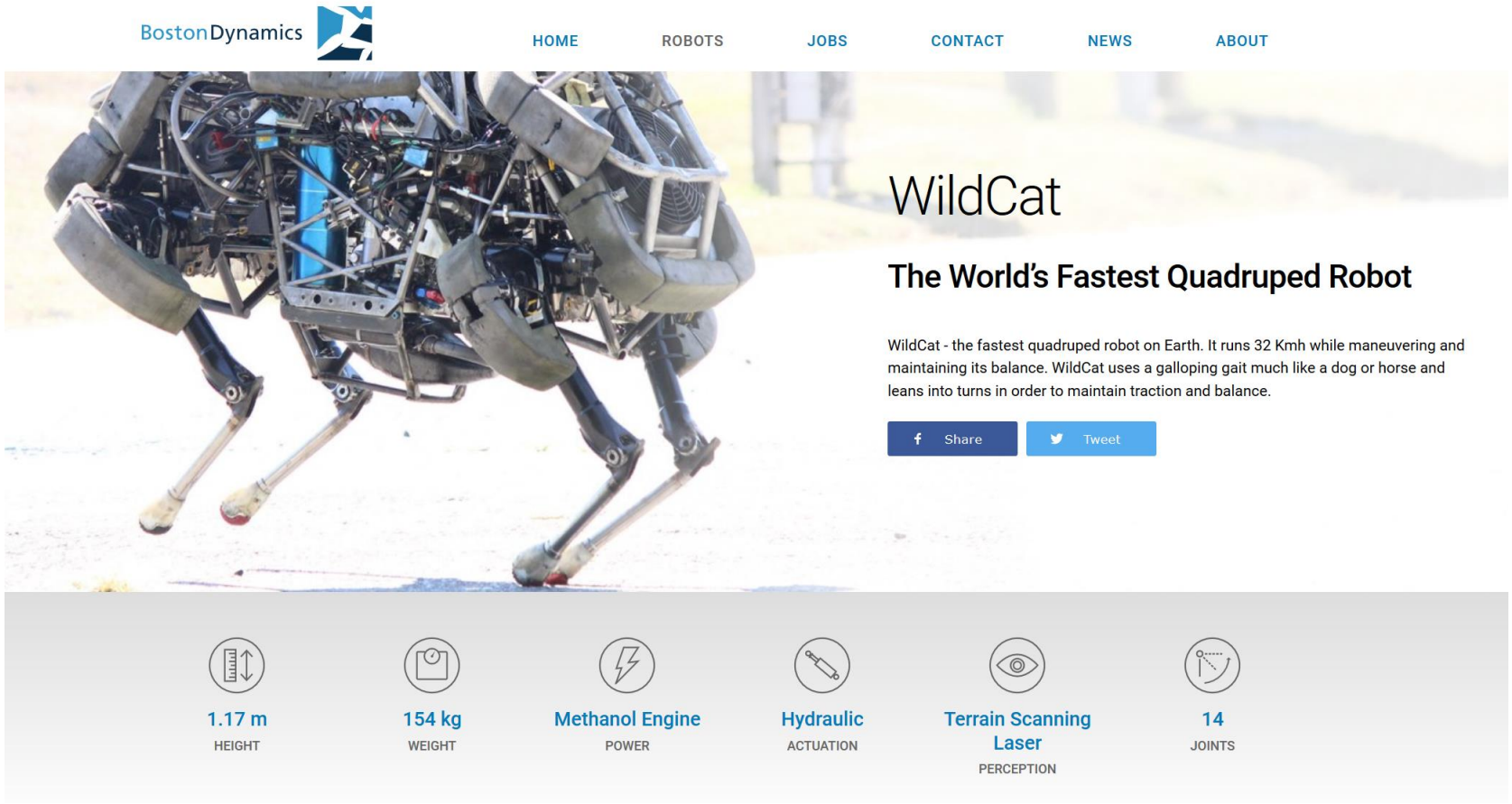
More Recently…

Is this System Based on Machine Learning?

From https://www.youtube.com/watch?v=wE3fmFTtP9g

Is this System Based on Machine Learning? No!

"The robot uses a variety of gaits, including trotting, bounding, and galloping to maintain its balance while running and manoeuvring over relatively flat terrain. The on-board computer uses dynamic control algorithms and a variety of sensors (IMU, ground contract, proprioception, visual odometry) to control and stabilize the running motion."



From https://www.bostondynamics.com/wildcat

What is Machine Learning?

Machine learning is “the field of study that gives computers the ability to learn without being explicitly programmed” (Arthur Samuel, 1959). A machine learning system is comprised of the following four components:

- Dataset $\mathcal{S}$: a set of samples generated by some system or process; the samples can be single data points or pairs of input and output values
- Model $\mathcal{M}$: an adjustable and compact representation of a certain class of input/output relationships that is hypothesized to be capable of modeling the system or process which generates $\mathcal{S}$
- Objective Function $\mathcal{L}$: a function that encodes the current performance of $\mathcal{M}$ (e.g. loss or reward)
- Algorithm $\mathcal{A}$: the learning algorithm that adjusts $\mathcal{M}$ based on $\mathcal{S}$ and $\mathcal{L}$

Machine Learning in Artificial Cognitive Systems

Machine learning is an important prerequisite for the implementation of a broad range of cognitive functions in artificial cognitive systems:

- Learning and Development: modeling and implementation of biological learning mechanisms (operant conditioning, implicit learning, explicit learning, perception etc.)
- Memory, Knowledge, and Internal Simulation: modeling and implementation of the encoding, storage and retrieval of facts, experiences, and actions (e.g. associative memory)
- Perception: learning basic features to detect and categorize perceptual stimuli (e.g. unsupervised learning of visual features)
- Autonomy: dynamic adaption to changes in the environment (e.g. continuous online learning from a live data stream)

Programming vs. (Machine) Learning

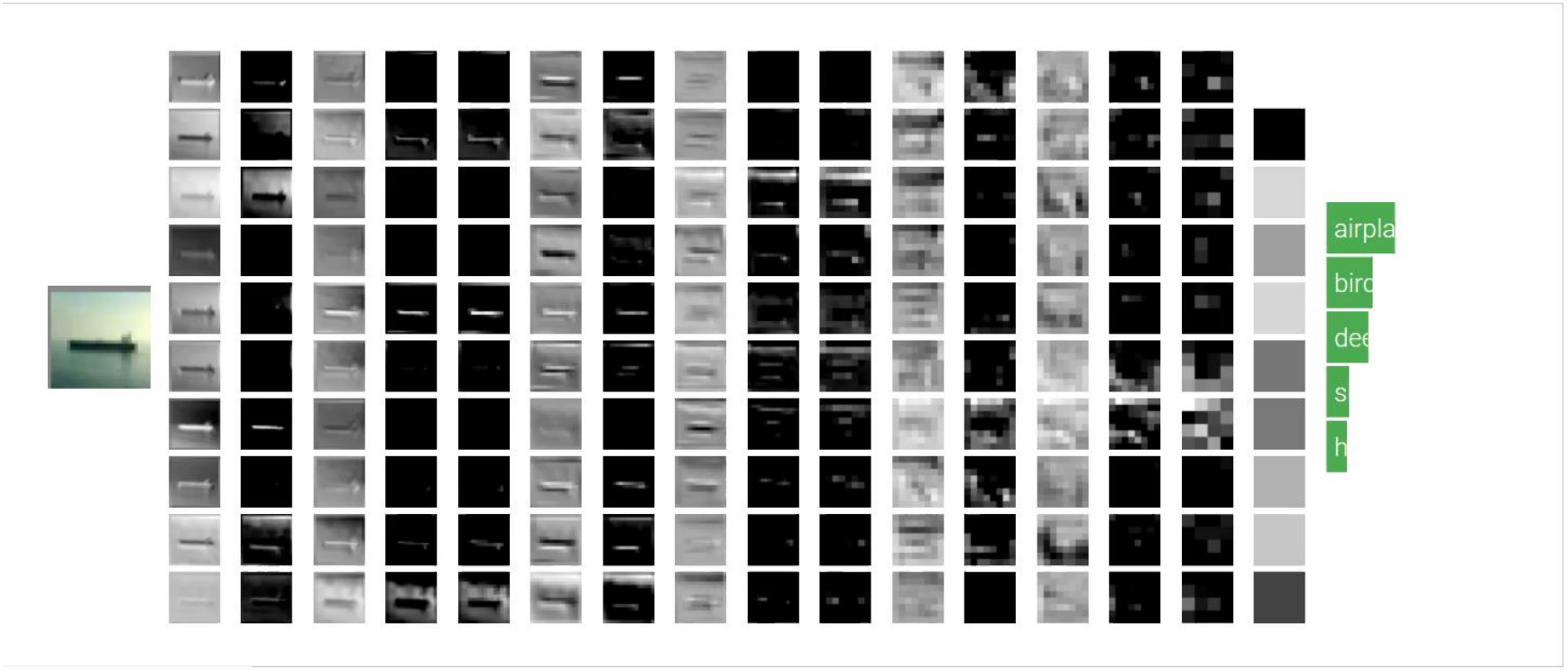
Machine learning is an important feature of artificial cognitive systems that are situated in complex dynamic environments. In these environments, it is not possible to program everything in advance because…

- … the environment is changing continuously
  (e.g. through actions of other cognitive agents or through physical processes)
- … dynamics and objects are too complex to be modeled explicitly
  (e.g. faces, animals, plants)
- … the system itself is subject to change
  (e.g. through growth, aging, injury, exercise, tool use)

Practical Applications of Machine Learning

- Image classification
- Speech recognition
- Autonomous driving
- Recommendation systems
- Threat protection
- Control systems
- …

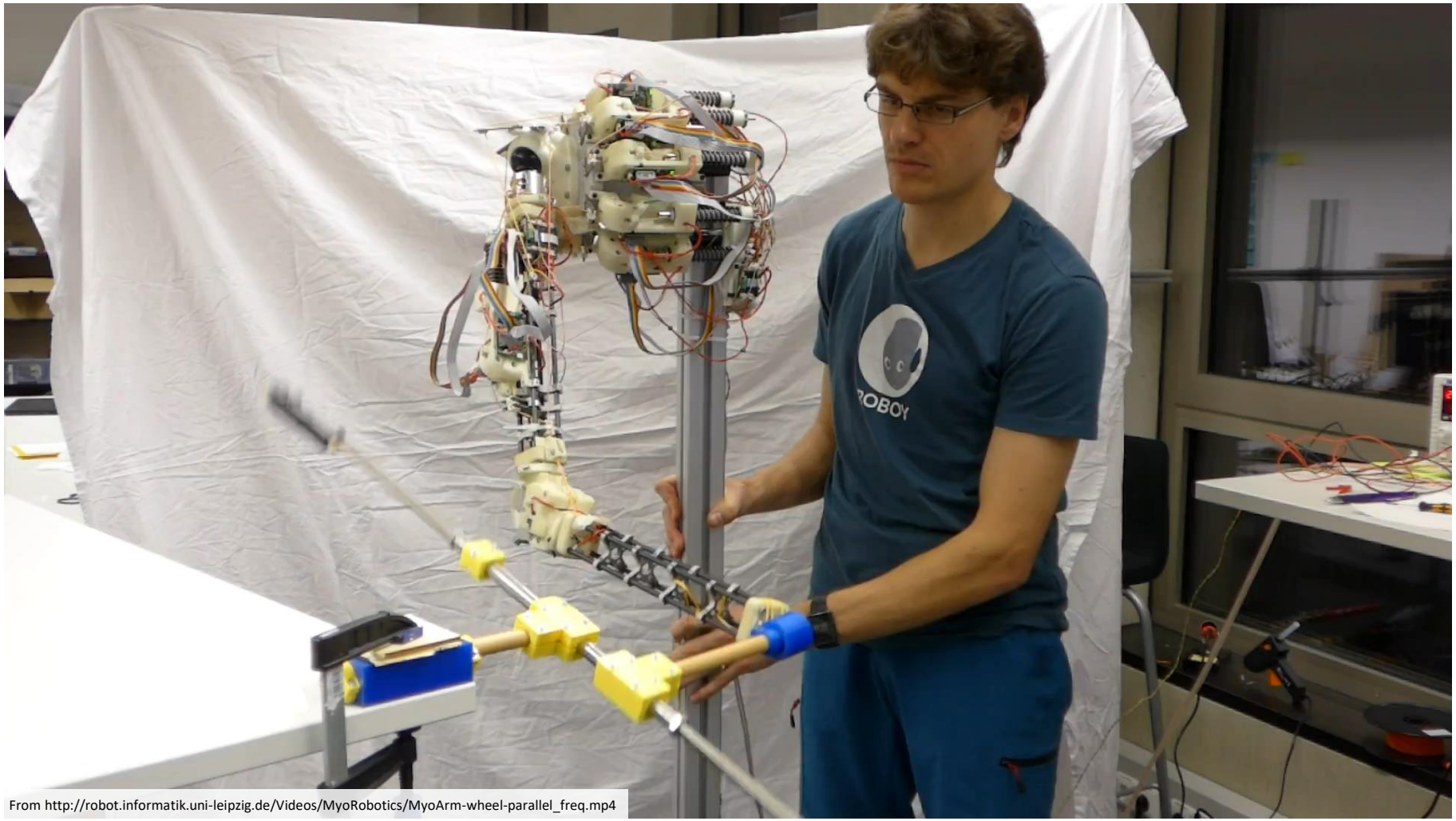
Example: Image Classification



From http://cs231n.stanford.edu/

Image classification of the CIFAR-10 dataset with a convolutional neural network with 17 layers and 7000 parameters.

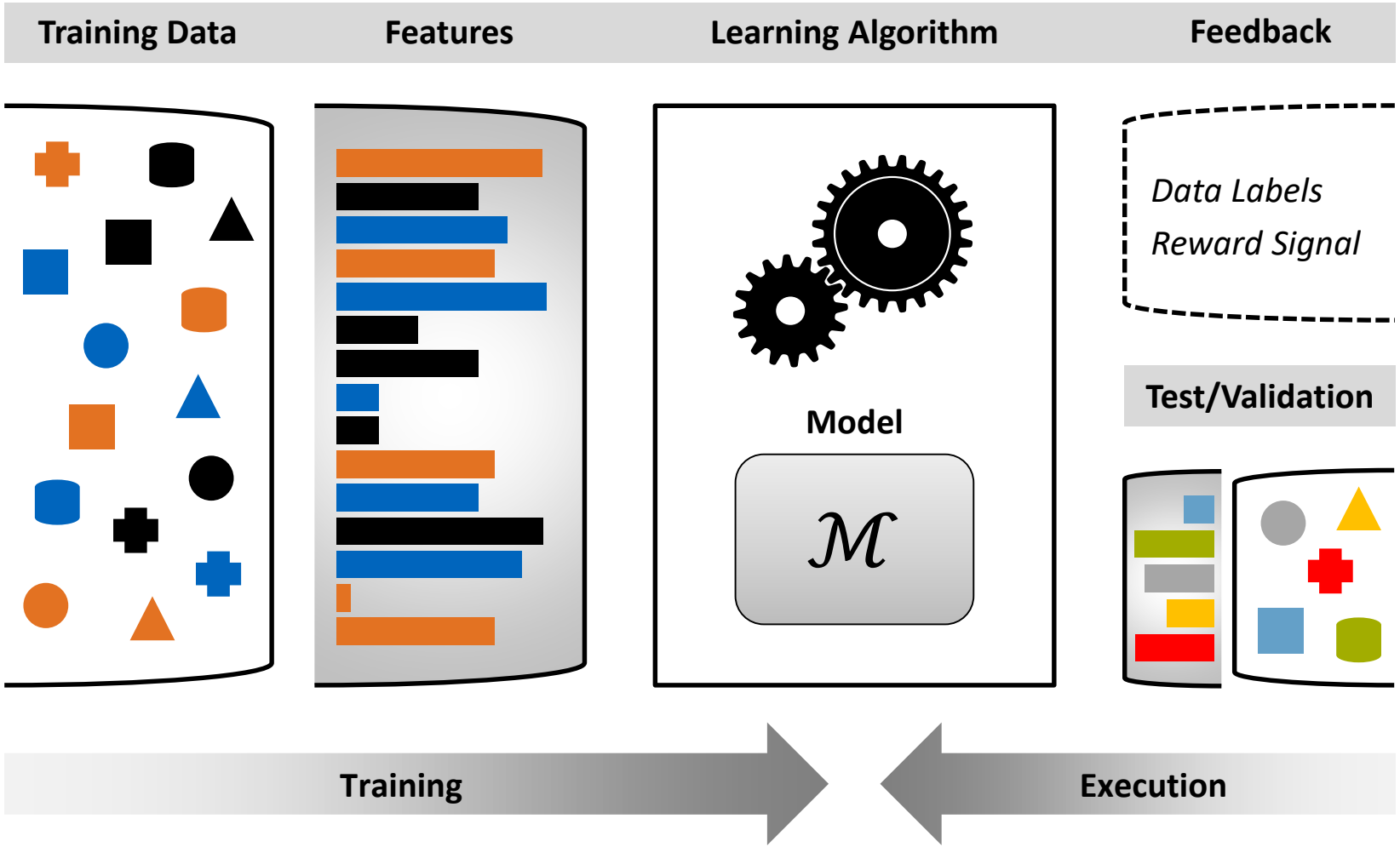
Example: Unsupervised Learning in Robotics



From http://robot.informatik.uni-leipzig.de/Videos/MyoRobotics/MyoArm-wheel-parallel_freq.mp4

Datasets and Learning Paradigms

Machine Learning: The General Workflow



Feature Selection and Feature Vectors

- There are many different types of features that can be used to describe the data at hand (shapes, colors, histograms, filters etc.)
- The selection of the right features (feature engineering) is critical for the performance of the machine learning model – the features must contain the information required for predictions
- All selected features are grouped into a feature vector:

$$\begin{pmatrix} colors \\ edges \\ location \\ \vdots \\ dimensions \end{pmatrix}$$

Definition of the Machine Learning Task

Overall Goal

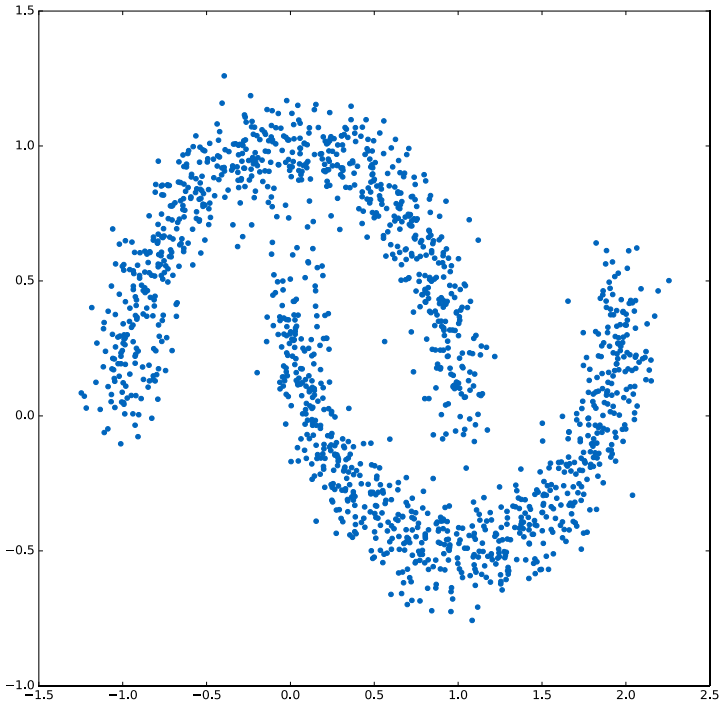
Train a model $\mathcal{M}$ in a hypothesis space $\mathcal{H}$ using a learning algorithm $\mathcal{A}$ so that $\mathcal{M}$ minimizes loss $\mathcal{L}$ for dataset $\mathcal{S}$. This type of learning is referred to as inductive learning.

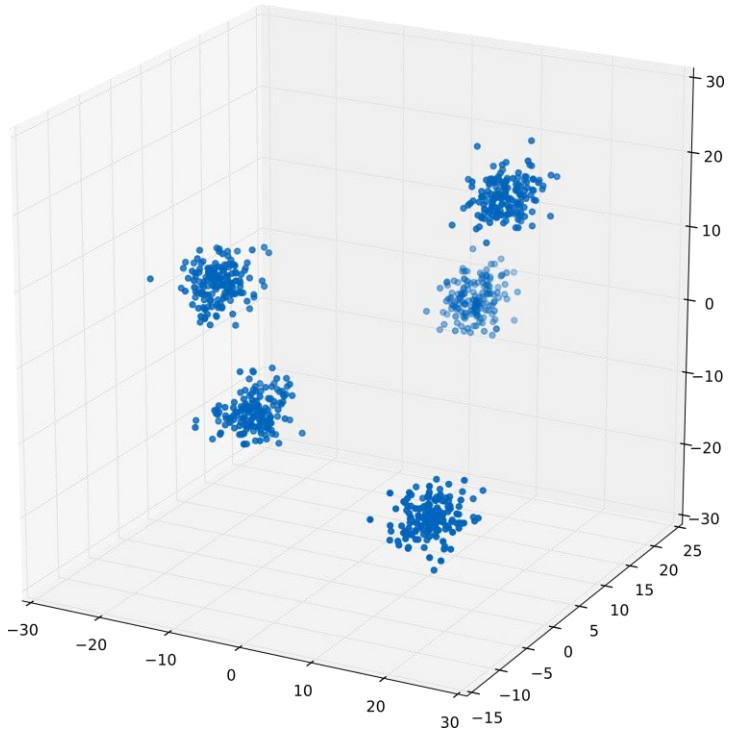
→ The choice of $\mathcal{H}$ and $\mathcal{L}$ depends heavily on the properties of $\mathcal{S}$

Types of Datasets

- Unlabeled data: $\mathcal{S} = \{x_1, x_2, \dots, x_n\}$
- Labeled data: $\mathcal{S} = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$
- Mixed data: $\mathcal{S} = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\} \cup \{x_1, x_2, \dots, x_n\}$
- "Dynamic data": $S = \{(x_t, s_t) | \, s_{t+1} = \mathcal{D}(x_t, s_t) \wedge 0 \leq t \leq T\}$ where $\mathcal{D}$ is a dynamical system and s_t denotes the system state

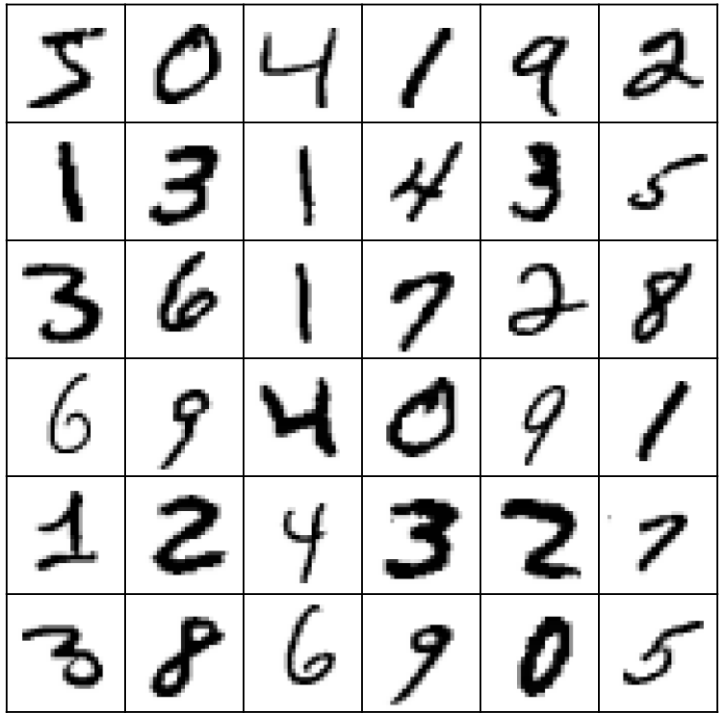
Example: Unlabeled Datasets





Most real-world data is unlabeled (e-mails, images, video recordings, measurement data) and assigning labels is usually a tedious manual task. However, natural datasets usually have specific structural features (→ Big Data, Datamining).

Example: Labeled Handwritten Digits



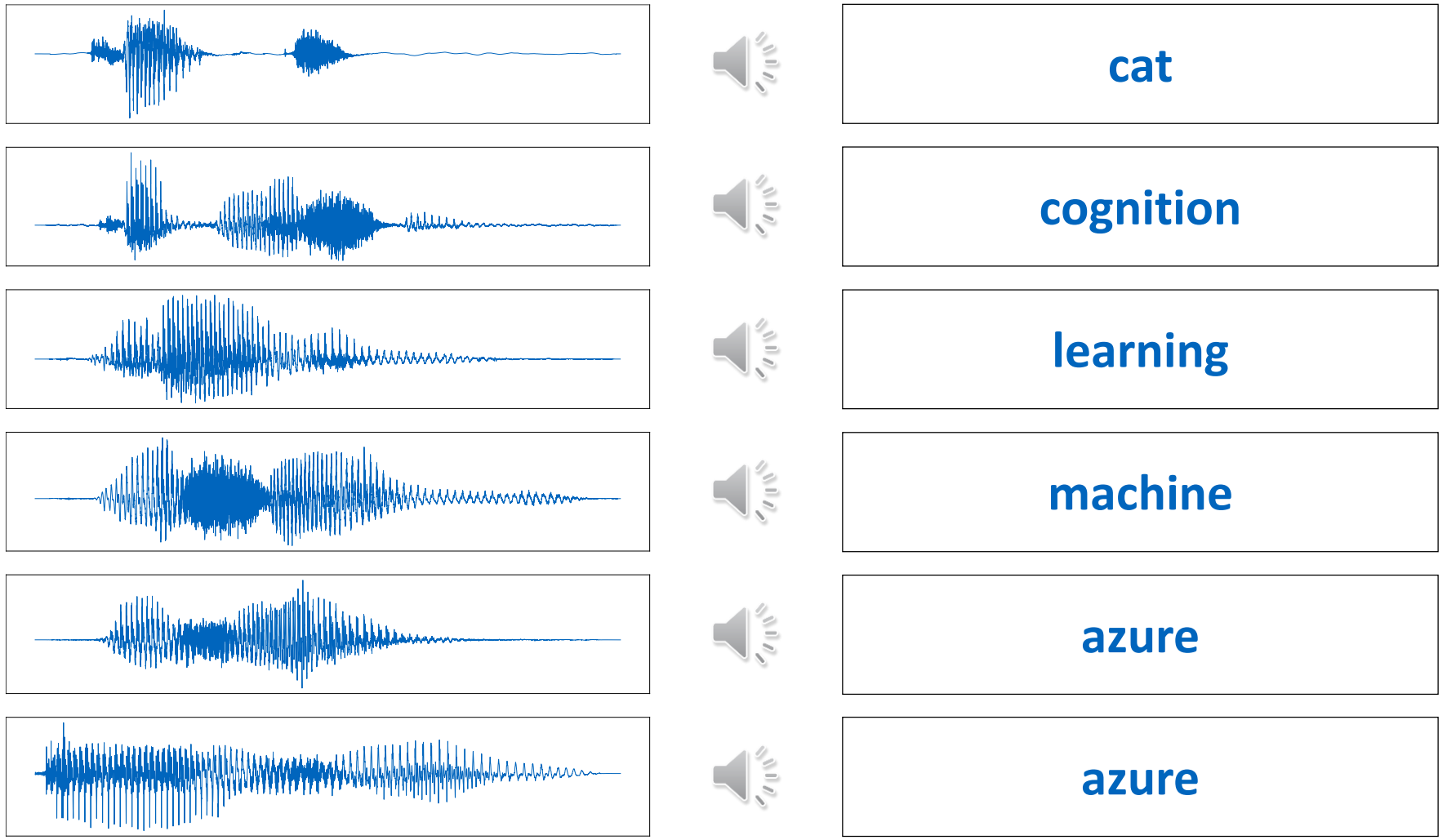
Raw Data

5	0	4	1	9	2
1	3	1	4	3	5
3	6	1	7	2	8
6	9	4	0	9	1
1	2	4	3	2	7
3	8	6	9	0	5

Labels

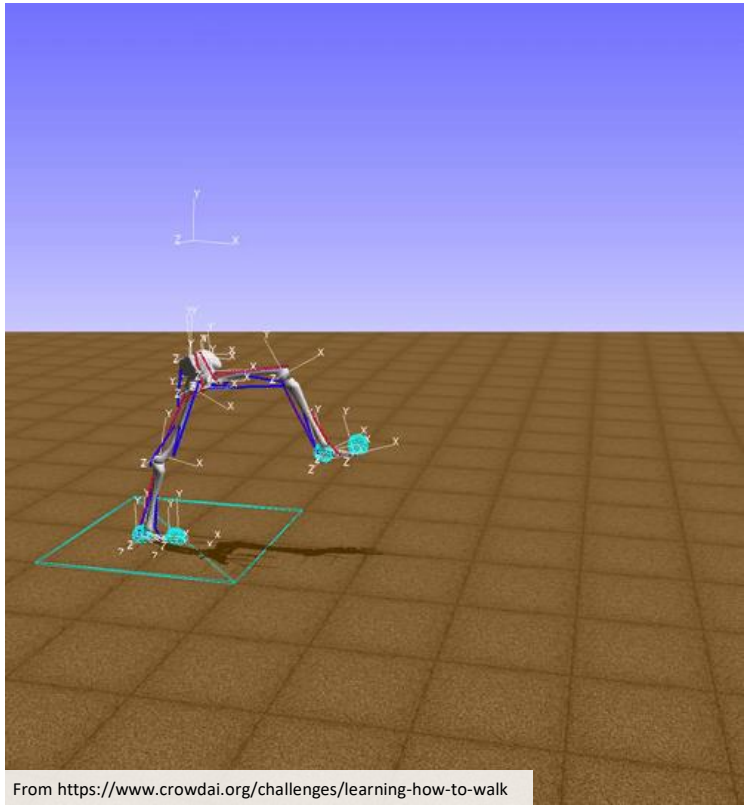
The MNIST dataset contains 70,000 size-normalized, centered, and labeled handwritten 28x28 pixel digits (http://yann.lecun.com/exdb/mnist/).

Example: Labeled Audio Files



Audio from http://www.leo.org/

Example: Dynamic Tasks and Environments



From https://www.crowdai.org/challenges/learning-how-to-walk

Levine et al. (2016), from https://www.youtube.com/watch?v=CE6fBDHPbP8

Many real-world tasks have complex dynamics and an unknown goal representation and therefore cannot be modeled as pairs of input and desired output.

Types of Machine Learning

Unsupervised Learning
- Solely unlabeled data
- Discovery of structural features in the data set

Semi-Supervised Learning
- Labeled and unlabeled training samples
- A priori assumptions on input data required

Amount of information provided during training

Reinforcement Learning
- Interaction with the environment
- Reward signal encodes feedback for the policy

Supervised Learning
- All training samples are labeled
- Desired output is specified exactly

Datasets and Learning Paradigms

The choice of a learning paradigm is in most cases directly motivated by the type of data available:

Unlabeled Data	→	Unsupervised Learning
Labeled Data	→	Supervised Learning
Mixed Data	→	Semi-Supervised Learning
Dynamic Environment	→	Reinforcement Learning

- In practice, especially supervised learning requires assigning labels manually
- Unsupervised learning is often applied to labeled datasets as a preprocessing tool for initial data analysis

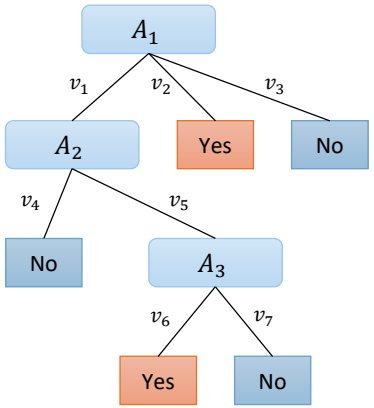
Model Selection

The model $\mathcal{M}$ of a machine learning system encodes, stores and retrieves the outcome of the learning process. The adaption of $\mathcal{M}$ by a learning algorithm $\mathcal{A}$ to find an optimal model $\mathcal{M}^*$ corresponds to a search in a hypothesis space $\mathcal{H}$:

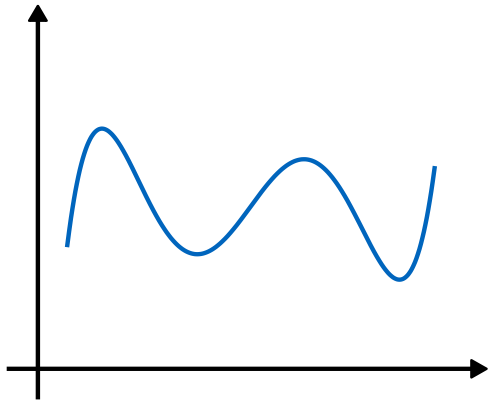
$$\mathcal{M}^* = \underset{\mathcal{M} \in \mathcal{H}}{\operatorname{argmin}} \mathcal{L}(\mathcal{M})$$

- In most practical applications, it is not possible to find a global minimum $\mathcal{M}^*$; most learning algorithms therefore search for local optima
- $\mathcal{H}$ determines which aspects of the data are captured (e.g. simple statistics or probability distributions) and how they are represented (e.g. implicit or human-readable)
- Different hypothesis spaces support the incorporation of a priori knowledge about the learning task to different degrees (e.g. neural networks or logic)

Examples of Common Hypothesis Spaces



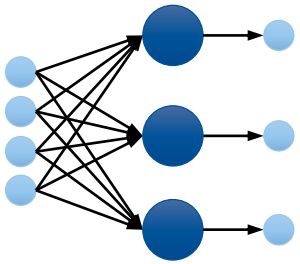
Decision Trees



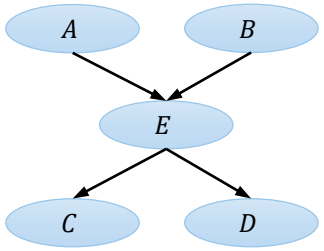
Polynomials

$$A \wedge B \Rightarrow C$$
$$\forall x: P(x)$$

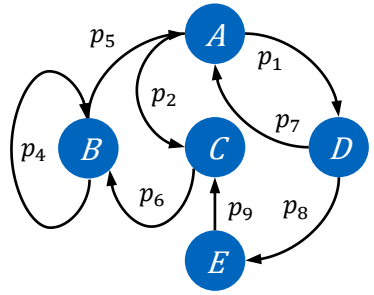
Logic



Neural Networks



Bayesian Networks



Markov Models

Ensemble Methods and Boosting

Combining Hypotheses to Ensembles

Ensemble methods in machine learning are a simple way to extend hypothesis spaces by combining a set of hypotheses $h_1, h_2, \ldots, h_n \in \mathcal{H}$ to a new hypothesis $h^* \in \mathcal{H}^n$.

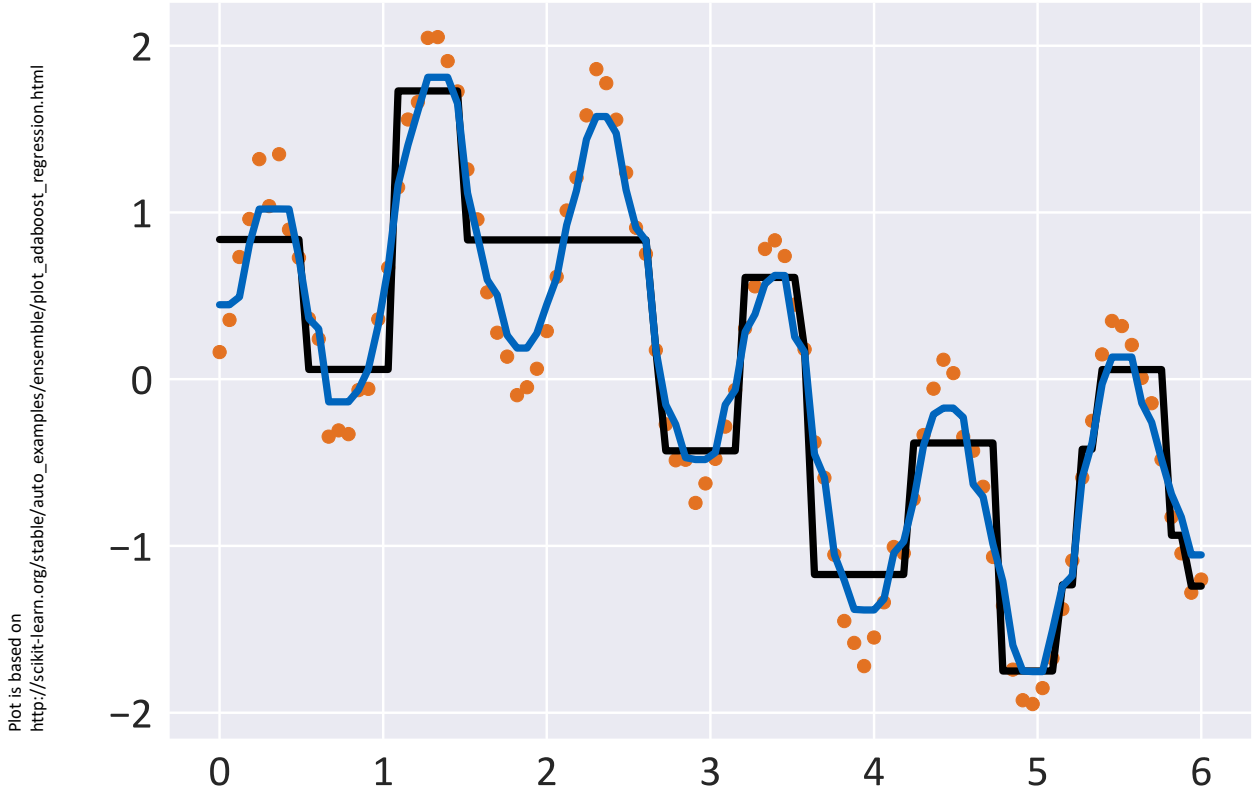
→ Is it possible to build a strong learner by combing several weak learners with accuracies that are just slightly above chance?

Boosting

Boosting algorithms compute a strong learner by incrementally constructing an ensemble of hypotheses:

- Every training sample $s_i \in \mathcal{S}$ is assigned a weight w_i; initially, all weights are set to the same value
- Weights of incorrectly learned samples are increased
- The training of new hypotheses focusses on samples with high weights

Example: Boosted Decision Tree Regression



Decision tree regression with a single regression tree (**black**) and with 300 boosted decision trees (**blue**). Boosting was computed with the AdaBoost algorithm.

Underfitting, Overfitting, and Generalization

The performance of a hypothesis $h \in \mathcal{H}$ can be characterized by how well it predicts seen data from the training set and unseen data:

- "Fit": the performance of h the training data (i.e. the value of the of the objective function $\mathcal{L}$ for h and $\mathcal{S}$)
- Generalization: predictive performance of h on data that were not considered during the training phase

Depending on the expressiveness of the hypothesis space $\mathcal{H}$ and the properties of the objective function $\mathcal{L}$, two different issues can occur during learning:

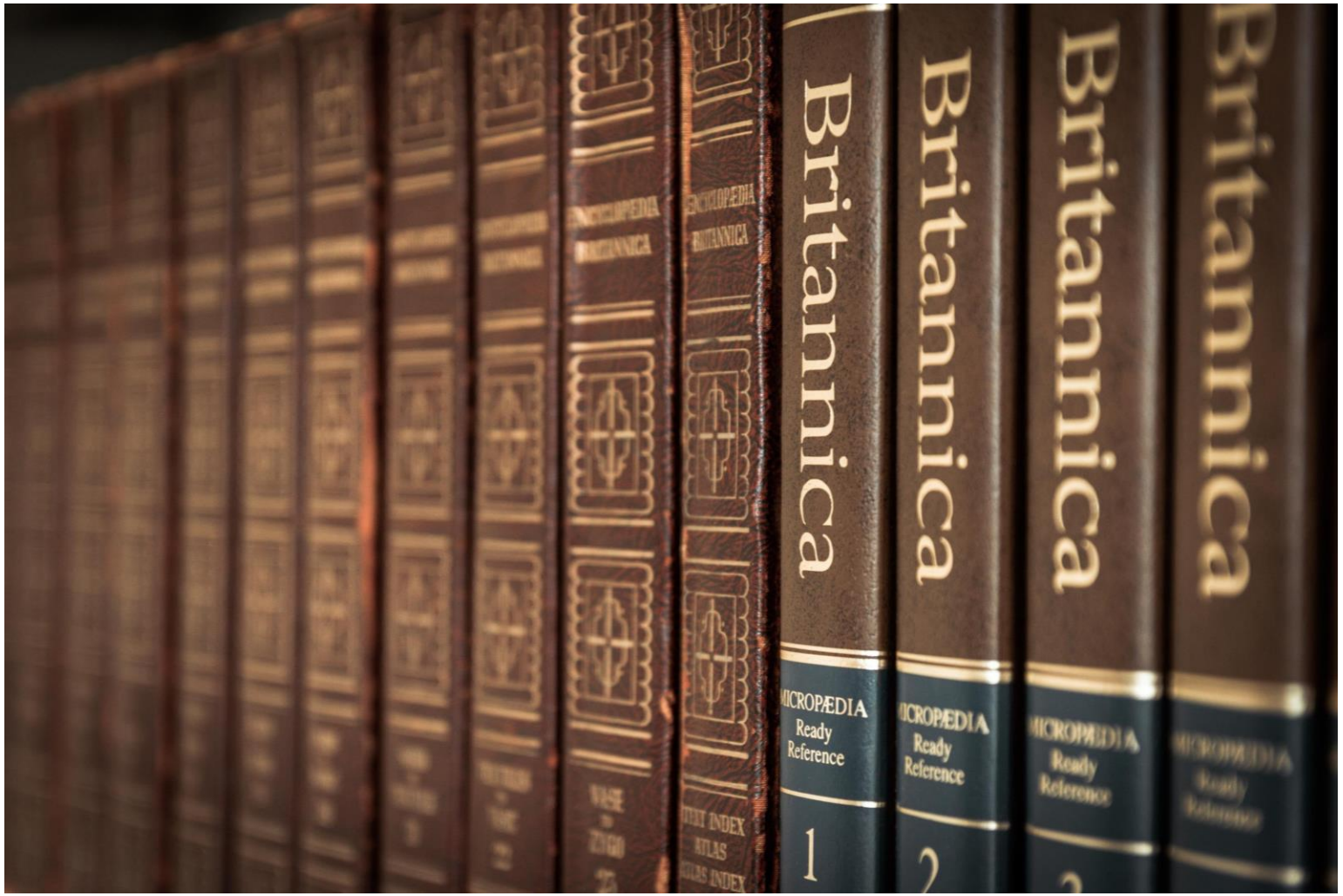
- Underfitting: h fits the training data poorly and does not model the underlying process because $\mathcal{H}$ is not expressive enough
- Overfitting: h fits the training data very well but does not model the underlying process because it does not generalize well

An Analogy to Cognition

An overfitted model “learns the data by heart” without understanding the underlying rules.

This is like a student who learns all exercises in a math book by heart without having understood the underlying theorems and formulas. She will only be able to answer questions that are identical to those in the book.

Adapted and extended from Wikipedia

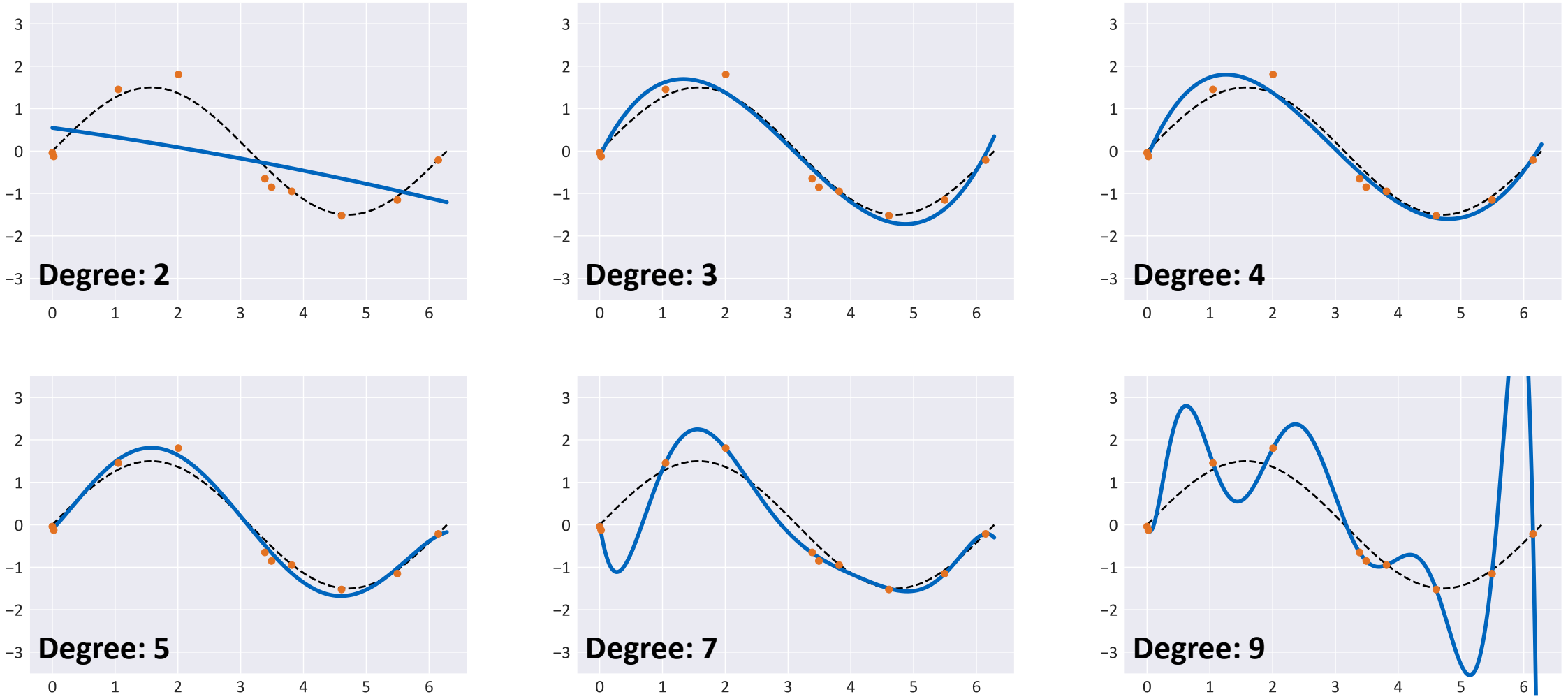


Occam's Razor

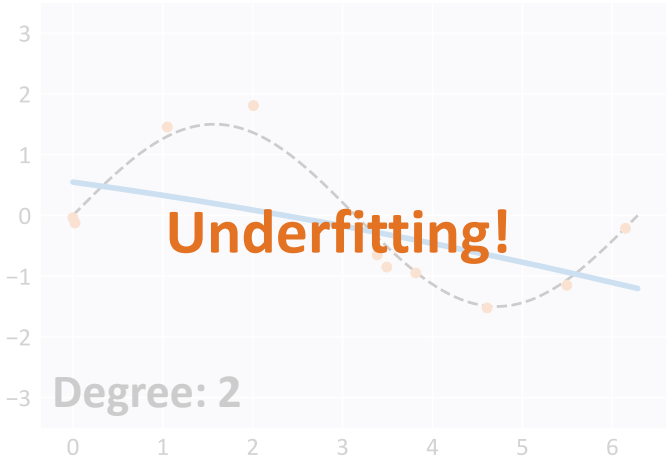
Models h that overfit a given dataset S have more parameters than required to capture the properties of the process that generates S. In general, it is desirable to choose a model that is as simple as possible. This law of parsimony is stated in the principle of Occam's razor :

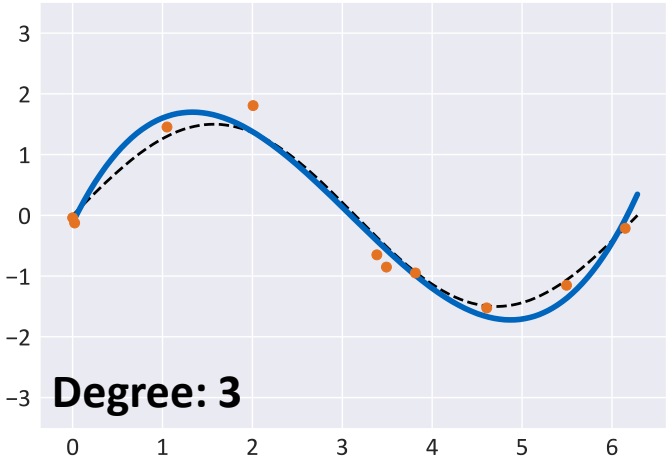
> "Of two competing theories, the simpler explanation of an entity is to be preferred."
>
> *Encyclopedia Britannica*

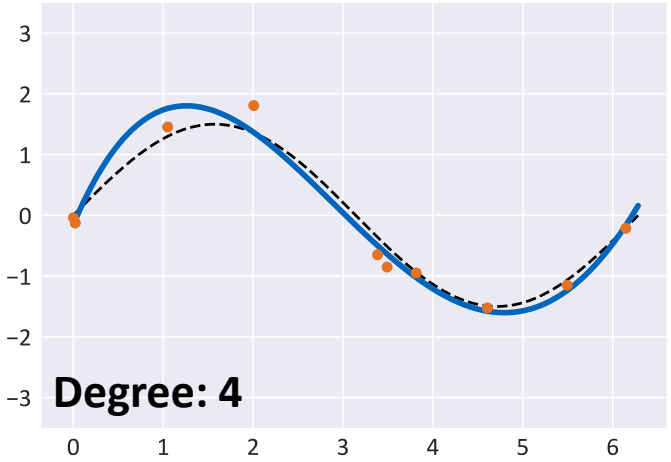
Example: Fitting Polynomials to a Dataset

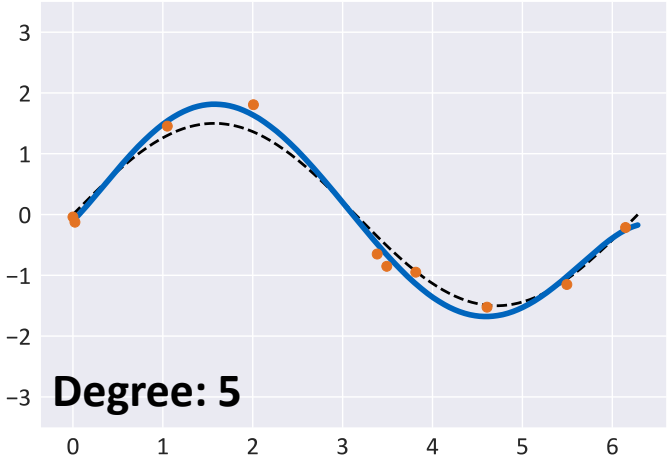


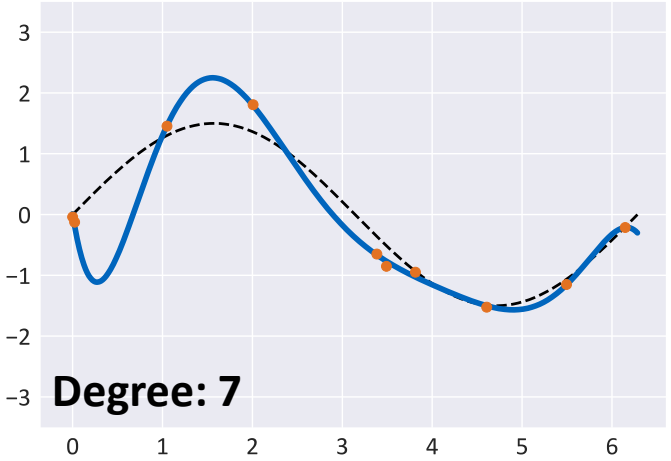
Example: Fitting Polynomials to a Dataset

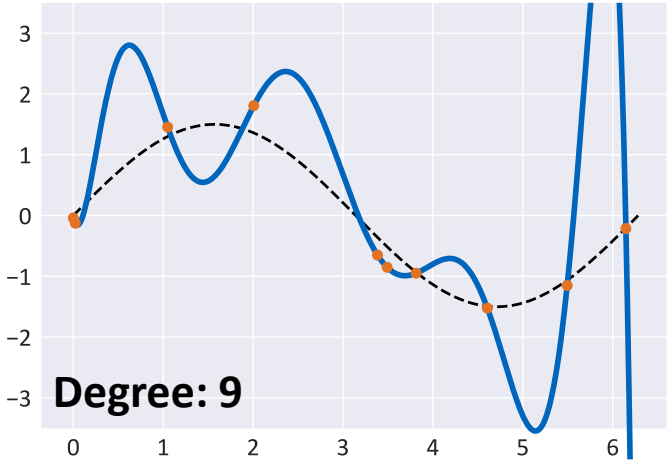




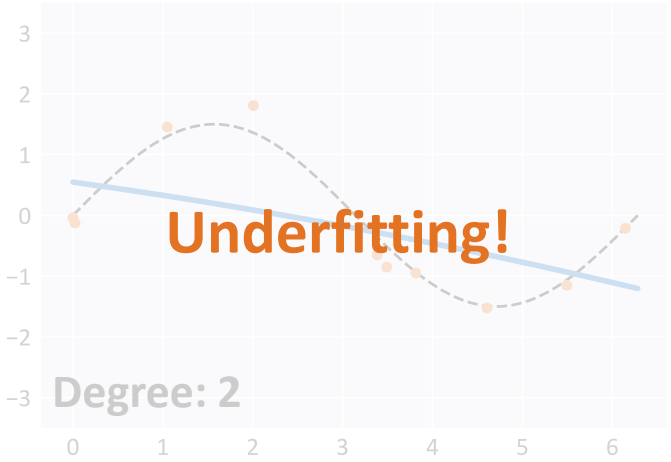


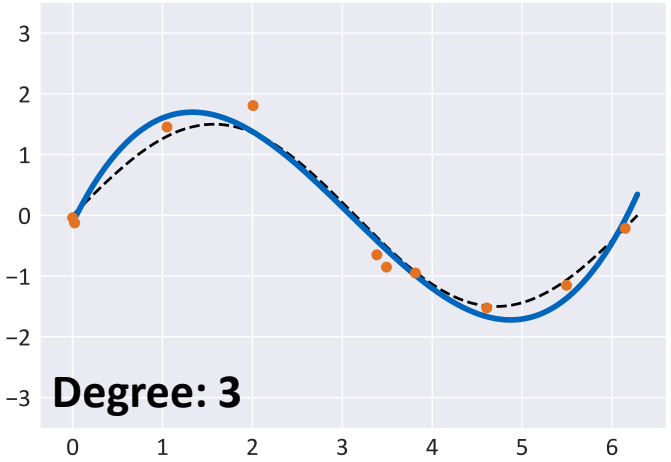


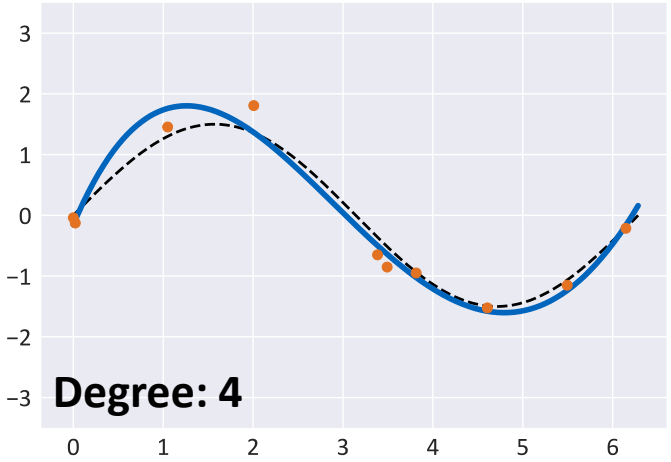


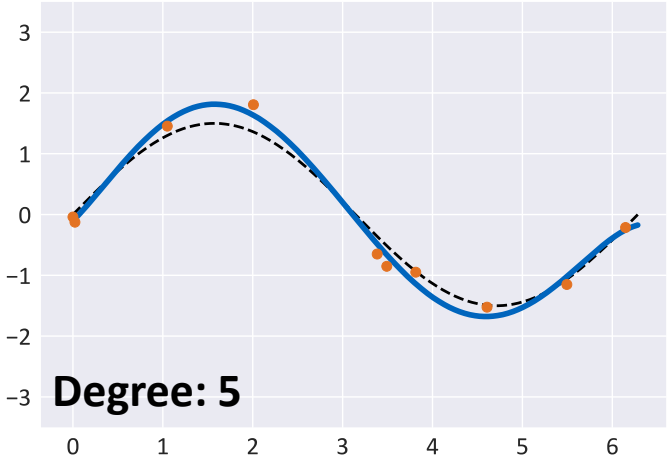


Example: Fitting Polynomials to a Dataset

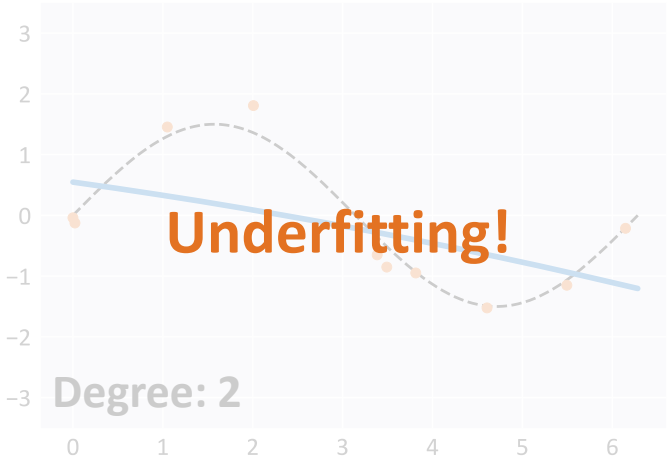


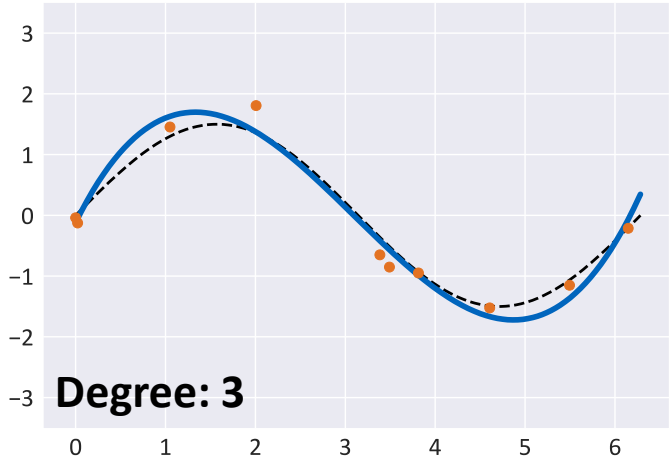


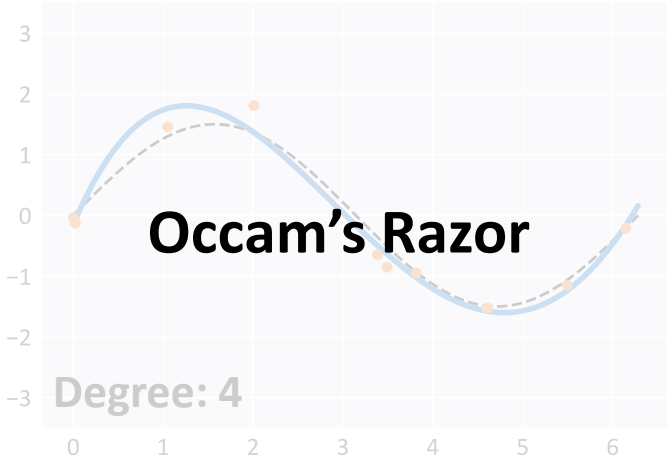


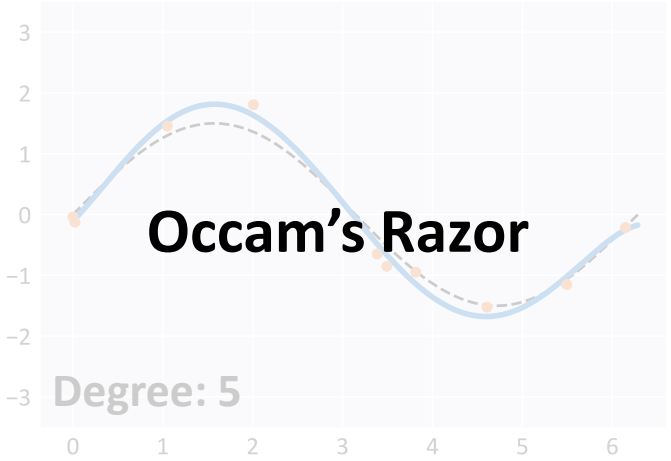


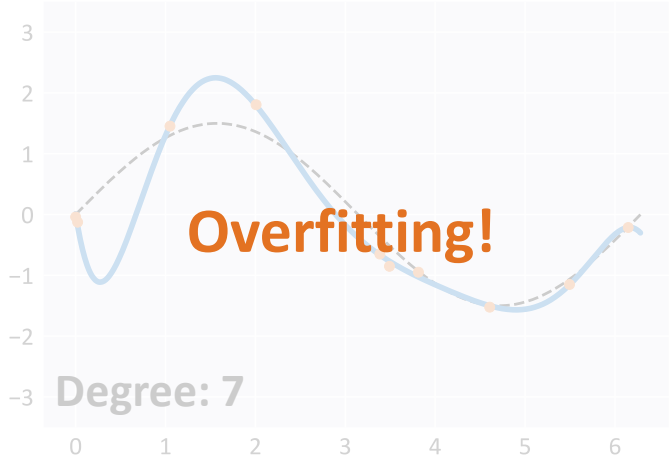
Example: Fitting Polynomials to a Dataset











Generative and Discriminative Models

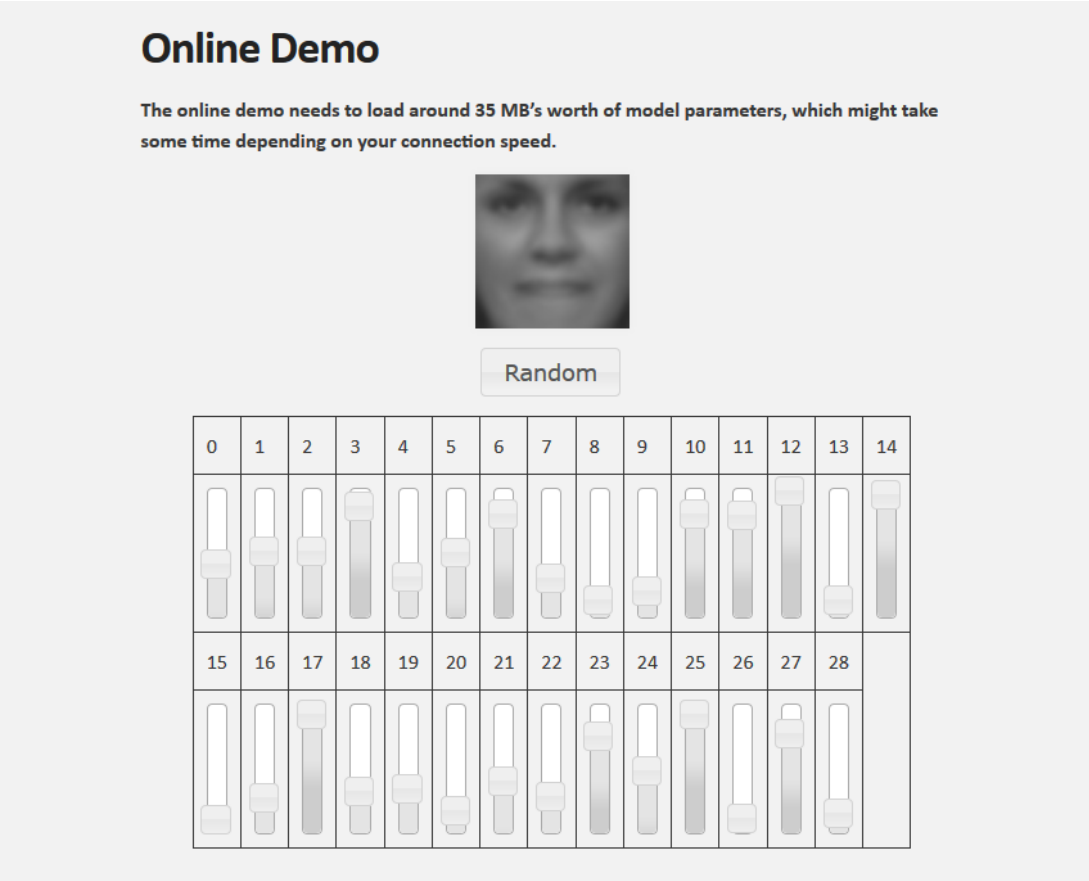
Probabilistic models cannot only be trained to discriminate data (compute the prediction y for an input x) from the distribution $\mathcal{D}$ that generates $\mathcal{S}$ but also to model $\mathcal{D}$ directly. In general, samples (x, y) drawn from $\mathcal{D}$ are distributed according to $\mathrm{P}(x, y)$:

- Discriminative models are based on the posterior probabilities $\mathrm{P}(y|x)$
- Generative models are based on the prior probabilities $\mathrm{P}(x|y)$; predictions can be computed by applying Bayes' theorem:

$$\mathrm{P}(y|x) = \frac{\mathrm{P}(x|y)\mathrm{P}(y)}{\mathrm{P}(x)}$$

Generative models are compact representations of the training data that have considerably less parameters than the dataset $\mathcal{S}$.

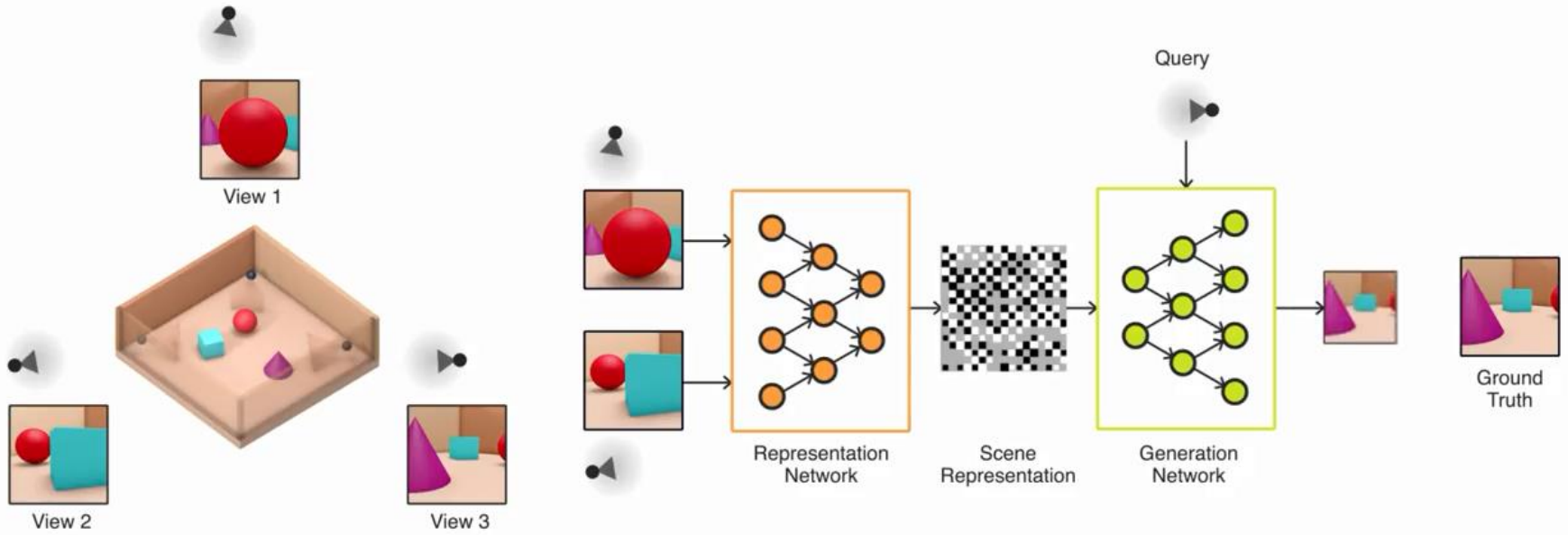
Demo: Generative Modeling of Faces



http://vdumoulin.github.io/morphing_faces

Generation of Camera Views in 3D Scenes

Generative Query Networks learn internal scene representations in a representation network and generate predict the visual appearance of the scene through a generation network.



From https://www.youtube.com/watch?v=wE3fmFTtP9g

Detection of Overfitting

Overfitting can be detected by applying the model h to unseen data samples and computing the objective function $\mathcal{L}$. Especially in supervised learning, the data are therefore split into three different and disjoint subsets:

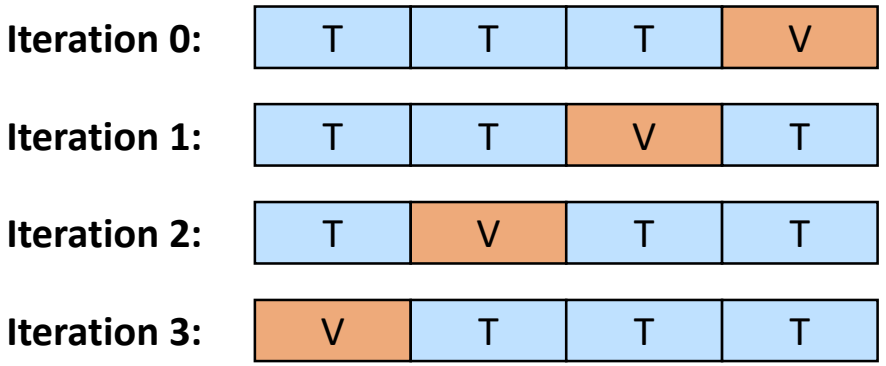
- Training set: the samples used in the training phase by the learning algorithm to search for a hypothesis h in the hypothesis space $\mathcal{H}$
- Validation set: a set of samples that are used to assess the performance of a hypothesis h that was computed in the training phase; based on the performance of h, the parameters of the training phase can be adjusted
- Test set: a set of samples (or real-world data) that is used to assess the performance of the final model

If h is modified based on the performance of the test set, the test set is invalidated!

Cross-Validation

In typical real-world applications, data is generated by a physical processes in the real world (e.g. a robot interacting with an environment) and therefore limited. k-fold cross-validation enables an efficient use of data at the cost of computational complexity:

- The dataset is partitioned into k subsets and learned in k iterations
- In every iteration, a different subset is selected as validation set
- The overall performance corresponds to the averaged performances of the k iterations



Avoidance of Overfitting

Regularization

Overfitting is only possible with hypotheses h that are complex enough to capture statistical features that do not explain the data (e.g. noise). A regularization term in the objective function $\mathcal{L}$ guides the learning process towards simpler solutions by punishing complexity.

More Training Data

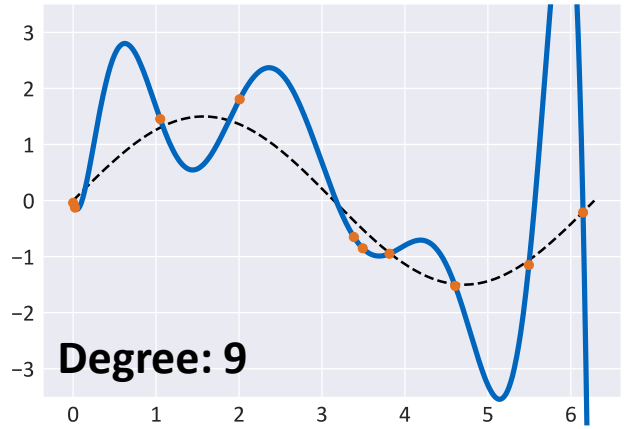
Overfitting can be reduced by increasing the size (i.e. the complexity) of the dataset.

Dataset Augmentation

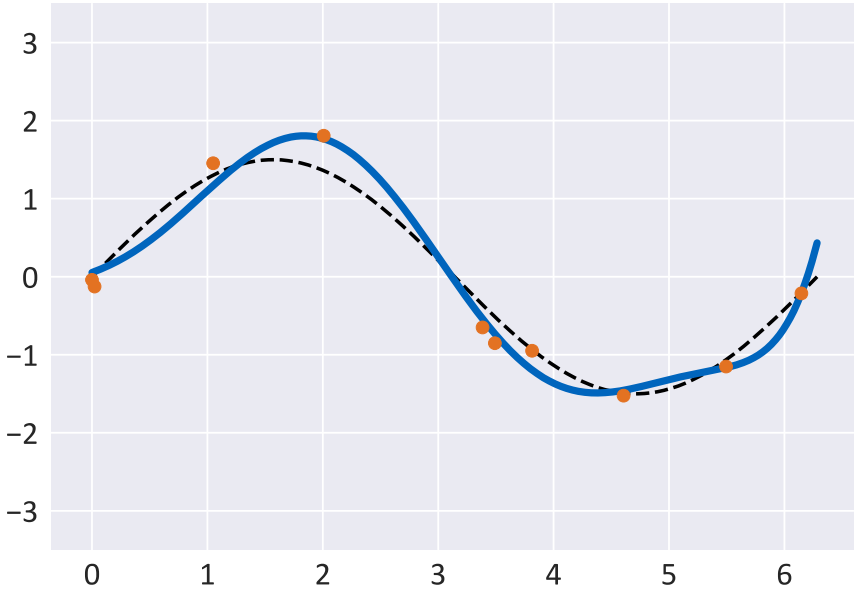
If not enough training data is available, the size of the dataset can be increased by applying transformations to the training samples (i.e. add noise, apply shifts, translations, and rotations, etc.).

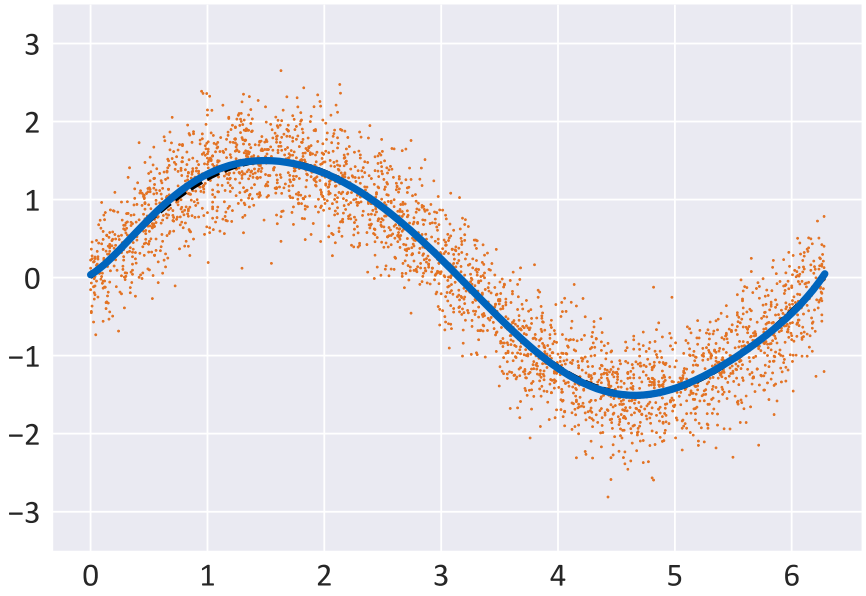
Example: Fitting Polynomials

Regularization



More Data





Classification Criteria of Learning Methods

Type of Learning
- Supervised learning
- Semi-supervised learning
- Reinforcement learning
- Unsupervised learning

Model
- Deterministic – Stochastic
- Parametric – Nonparametric
- Generative – Discriminative

Value Domain
- Discrete (classification)
- Continuous (regression)

Reasoning
- Inductive
- Deductive
- Transductive

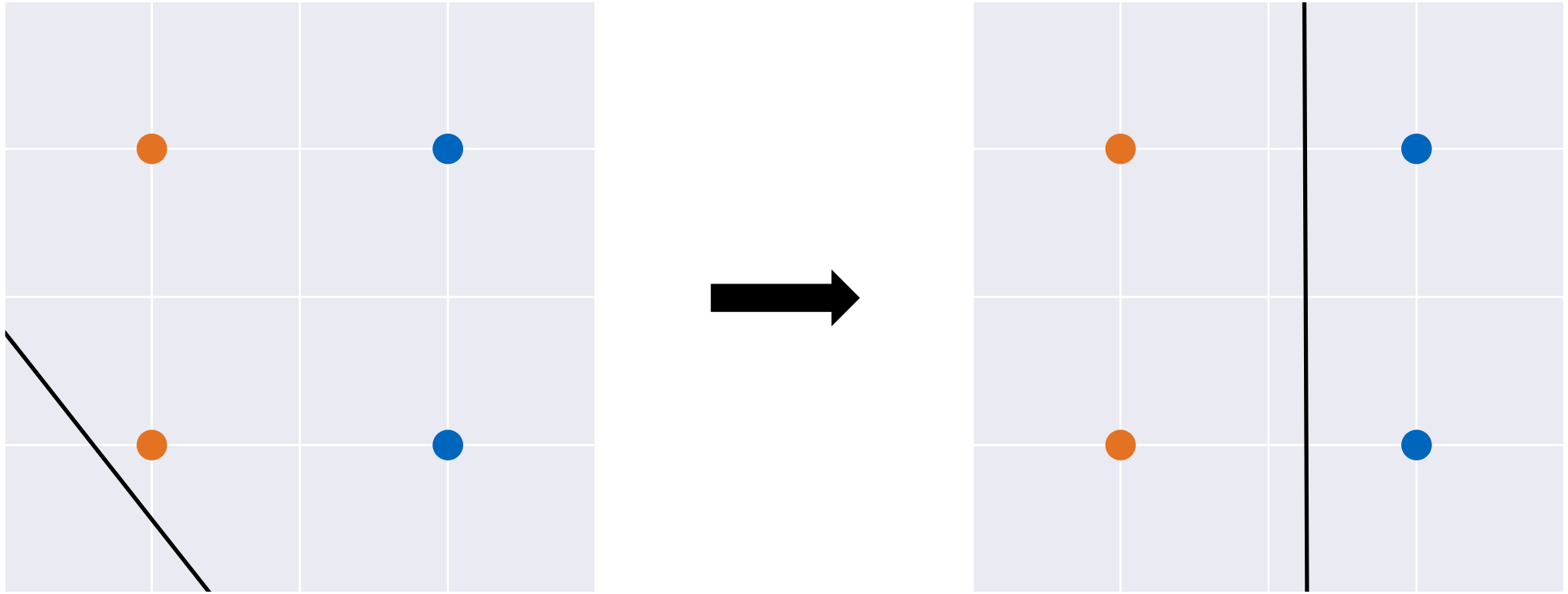
Online Learning vs. Offline Learning

Support for A Priori Knowledge

“Passive” Learning vs. “Active Learning”

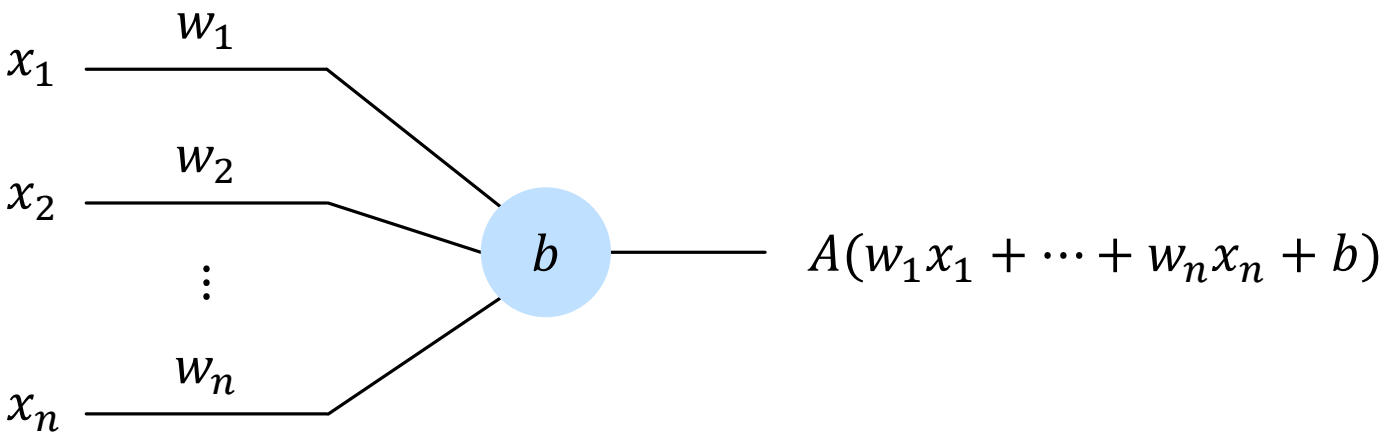
A First Example: The Perceptron Learning Rule

Goal: Separate Points through a Line



Artificial Neurons

An analog artificial neuron is defined by its synaptic weights and an activation function:



The output $n(x)$ of a neuron n with activation function A, synaptic weights w, bias b, and input x is computed as follows:

$$n(x) = A(w_1x_1 + \cdots + w_nx_n + b) = A(w^Tx + b)$$

The Perceptron

The perceptron is a linear classifier that is based on a single neuron with a digital threshold function:

$$y(x) = f(w^T x) \text{ with } f(x) = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

With $y_n \in \{-1,1\}$, the following inequality holds for all correctly classified samples:

$$(w^T x_n) y_n > 0$$

The perceptron criterion punishes only incorrectly classified samples $n \in \mathcal{M}$:

$$E_P(w) = -\sum_{n \in \mathcal{M}} (w^T x_n) y_n$$

The Perceptron Learning Rule

Computing the gradient of $E_P(w)$ yields the following expression:

$$\nabla E_P(w) = -\sum_{n \in \mathcal{M}} \nabla(w^T x_n) y_n = -\sum_{n \in \mathcal{M}} x_n y_n$$

The online weight update rule can be derived by applying stochastic gradient descent and adding learning rate η:

$$\Delta w = \eta x_n y_n$$

Finally, the perceptron learning rule can be derived by applying the weight update:

$$w^{t+1} = w^t + \Delta w = w^t + \eta x_n y_n \text{ for all } n \in \mathcal{M}$$

$$w^{t+1} = w^t + \eta'\left(y_n - f^t(x_n)\right) x_n \text{ for all } n$$

Convergence of the Perceptron Learning Rule

A single update step of the perceptron learning algorithm reduces the error of the corresponding misclassified sample:

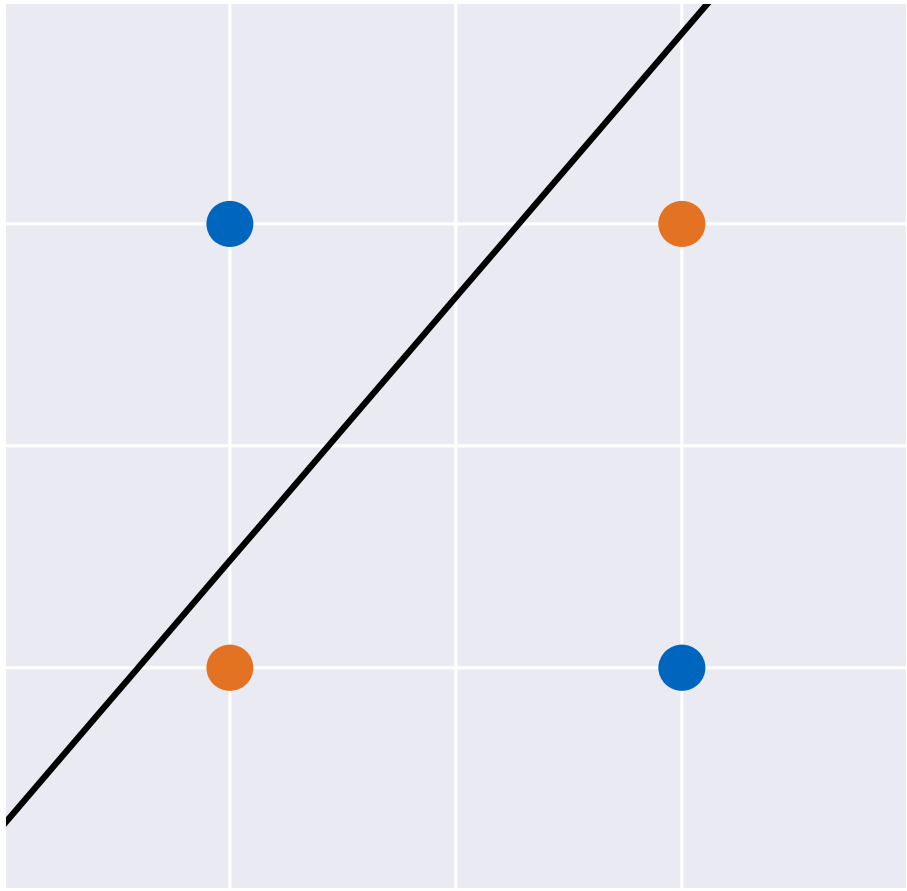
$$-w^{t+1^T} x_n y_n = -w^{t^T} x_n y_n - \eta (x_n y_n)^T (x_n y_n) < -w^{t^T} x_n y_n$$

In general, it can be proven that the perceptron learning rule has the following properties:

- If a solution exists, i.e. if the data set is linearly separable, then the perceptron learning algorithm finds a solution within a finite number of steps (perceptron convergence theorem)
- The solution computed by the algorithm depends on the initialization of the parameters and the order of presentation of the training samples
- The algorithm does not converge for not linearly separable data sets

Task 1: The XOR Problem

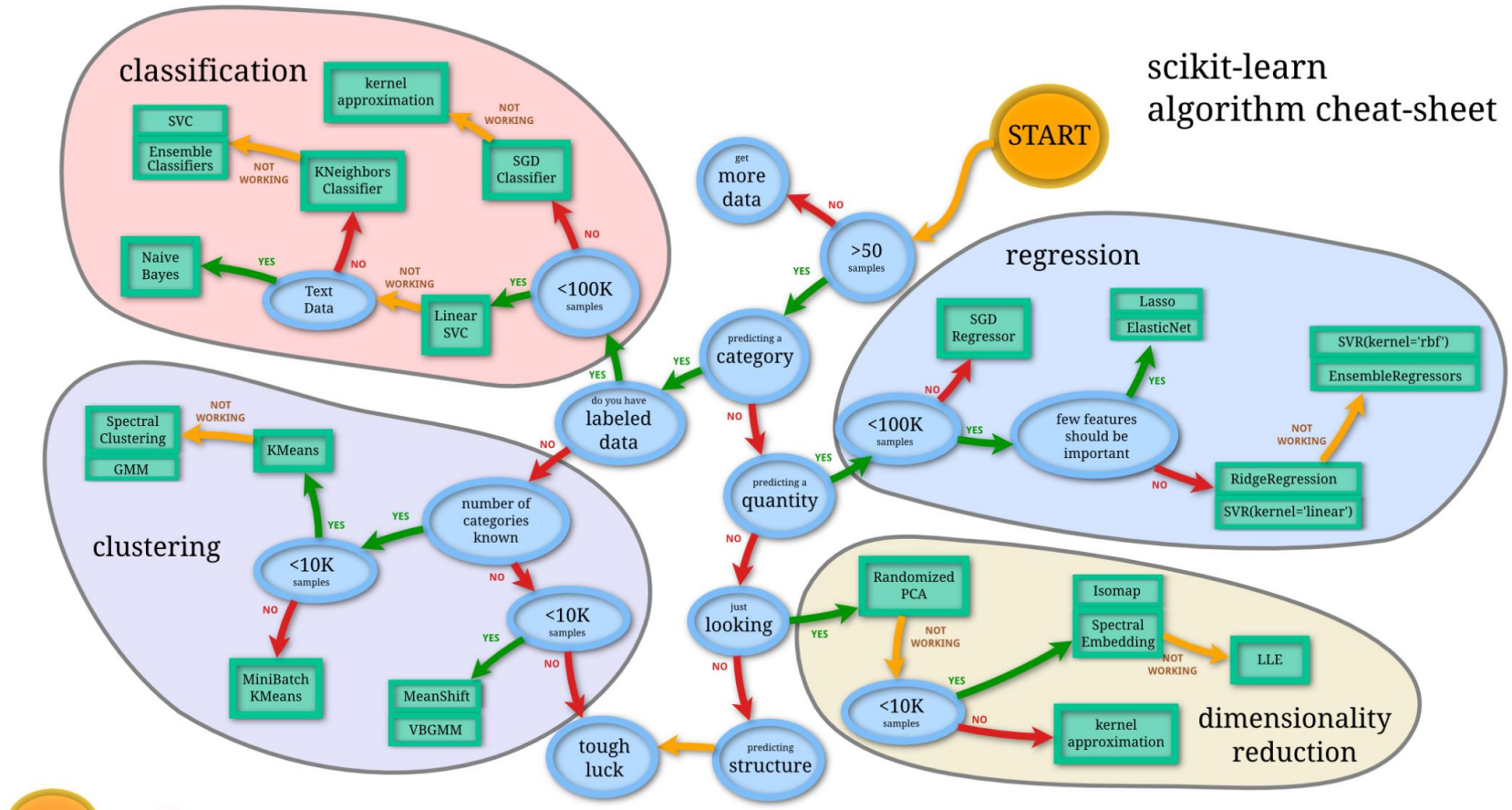
- The Perceptron learning rule only converges for linearly separable data sets
- It therefore cannot classify the XOR dataset correctly
- This finding led to an AI winter and the widespread abandonment of connectionism for almost two decades



Supervised Learning

Basic Regression and Classification

Machine Learning with scikit-learn



Regression: Task Definition

Input

- A dataset $\mathcal{S} = \{(x_0, y_0), (x_1, y_1), \dots, (x_N, y_N)\}$ with $x_i \in \mathbb{X}^K$ and $y_i \in \mathbb{Y}^L$
- $\mathbb{Y}^L$ is continuous space (e.g. $\mathbb{R}^L$)
- A loss function $\mathcal{L}: \mathbb{Y}^L \times \mathbb{Y}^L \longrightarrow \mathbb{R}$

Desired Output

A regressor $h: \mathbb{X}^K \longrightarrow \mathbb{Y}^L$ that predicts y_n for a given x_n with minimum loss $\mathcal{L}$.

Linear Basis Function Models

Linear basis function models are a class of hypothesis spaces that are linear combinations of functions on the samples x_n:

$$h_w(x_n) = w_0 + \sum_{j=1}^{M} w_j \phi_j(x_n)$$

- The functions $\phi_j(\cdot)$ are the basis functions of the model
- When setting $\phi_0(\cdot) = 1$ we can write $y(x_n, w) = \sum_{j=0}^{M} w_j \phi_j(x_n) = w^T x_i$
- The values $\phi_j(x_n)$ are called features of x_n

Examples of Basis Functions

Linear Functions

$$\phi_j(x_n) = x_n \text{ (linear regression)}$$

Powers of x_i

$$\phi_j(x_n) = x_n^j \text{ (polynomial regression for } x_i \in \mathbb{R})$$

Radial Basis Functions

$$\phi_j(x_n) = \exp\left(-\frac{(x_n-\mu_j)^2}{2s^2}\right)$$

Sigmoid Functions

$$\phi_j(x_n) = \sigma\left\{\frac{x_n-\mu_j}{s}\right\} \text{ with } \sigma(z) = \frac{1}{1+\exp(-z)}$$

Computing the Error for a Weight Vector w

The Sum-of-Squares Error Function

The sum-of-squares error of a hypothesis h_w with weights w_j on a dataset $\mathcal{S}$ is defined as

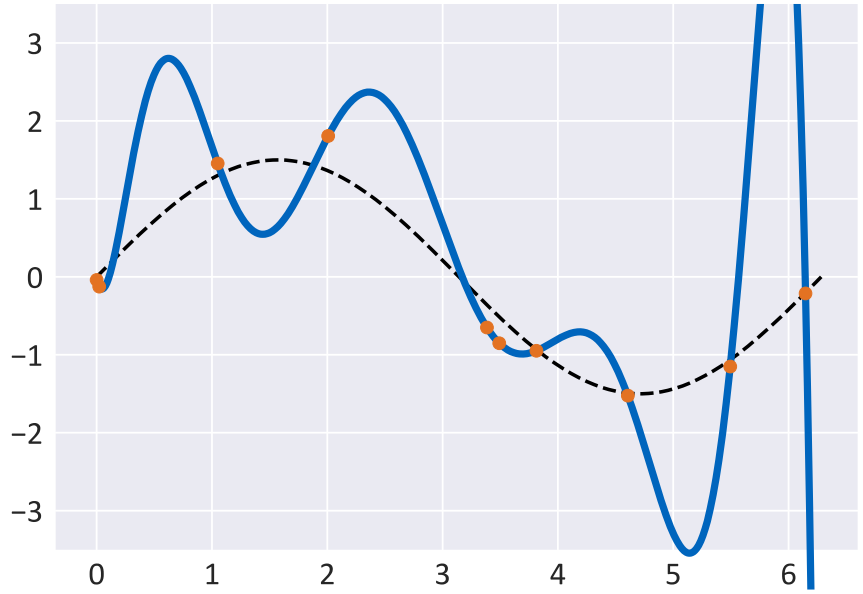
$$\mathcal{L}_{\mathcal{S}}(h_w) = \frac{1}{2}\sum_{n=1}^{N}(h_w(x_n) - y_n)^2$$

Optionally, it is possible to add a regularization term that punishes complex solutions:

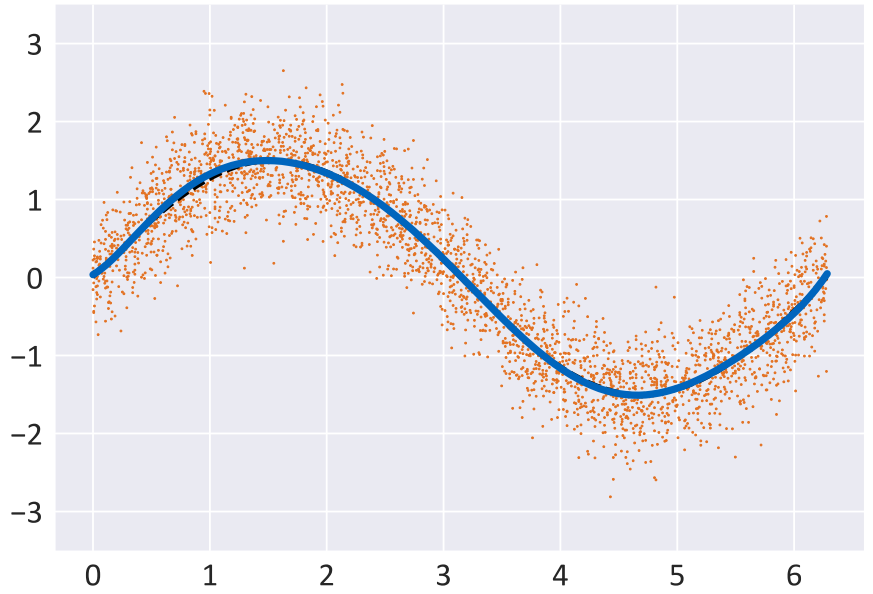
$$\mathcal{L}^*_{\mathcal{S}}(h_w) = \frac{1}{2}\sum_{n=1}^{N}(h_w(x_n) - y_n)^2 + \frac{\lambda}{2}\|w\|^2$$

→ Learning a locally optimal weight vector w corresponds to minimizing $\mathcal{L}$ or $\mathcal{L}^*$ with respect to w

Interpolation vs. Regression



The interpolation function $f(\cdot)$ must be consistent with $\mathcal{S}$: $f(x_i) = y_i$.



The hypothesis $h(\cdot)$ should minimize $\mathcal{L}$ and generalize well to new samples.

Learning w with Least Squares

Input

- A dataset $\mathcal{S} = \{(x_0, y_0), (x_1, y_1), \dots, (x_N, y_N)\}$
- A hypothesis space $\mathcal{H} = \{w_0\phi_0(x) + \cdots + w_M\phi_M(x) | w_i \in \mathbb{R}\}$

Goal

Minimize

$$\mathcal{L}_S(h_w) = \frac{1}{2}\sum_{n=1}^{N}(h_w(x_n) - y_n)^2$$

→ Find a set of weights w^* with $\nabla\mathcal{L}_S(\mathrm{h_w}) = 0$:

$$\frac{\partial}{\partial w_l}\mathcal{L}_S(h_w) = 0 \text{ for } w_0, w_1, \dots, w_M$$

Learning w with Linear Least Squares

The Least Squares Solution

A hypothesis h_{w^*} that minimizes $\mathcal{L}_{\mathcal{S}}(\cdot)$ can be computed by solving the equation below for w:

$$\phi^T \phi \, w^* = \phi^T y \longrightarrow w^* = (\phi^T \phi)^{-1} \phi^T y$$

In the above equation, ϕ and y are defined as follows:

$$\phi = \begin{pmatrix} \phi_0(x_0) & \cdots & \phi_M(x_0) \\ \vdots & \ddots & \vdots \\ \phi_0(x_N) & \cdots & \phi_M(x_N) \end{pmatrix} \qquad y = \begin{pmatrix} y_0 \\ \vdots \\ y_N \end{pmatrix}$$

A Geometrical Interpretation

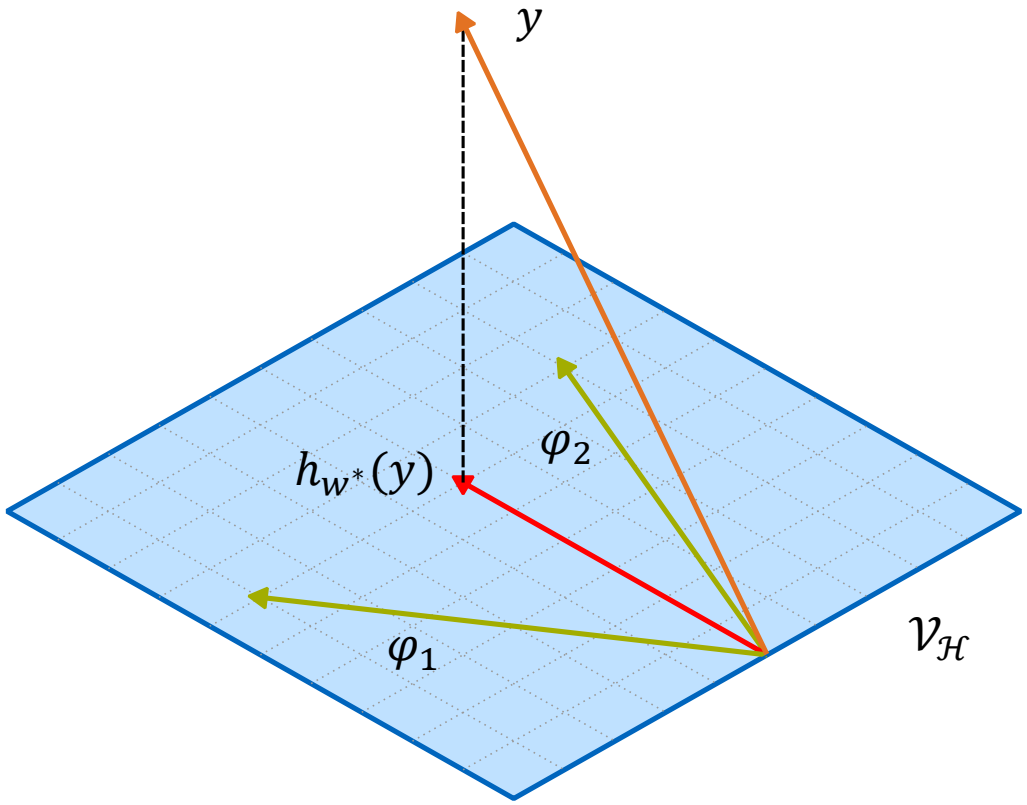
The minimization with least squares can be interpreted as a orthogonal projection of the vector

$$y = (y_0, y_1, \dots, y_n) \in \mathbb{R}^{N+1}$$

to the vector

$$h_{w^*}(y) = \left(h_{w^*}(x_0), \dots, h_{w^*}(x_n)\right)$$

in the subspace $\mathcal{V}_{\mathcal{H}} \in \mathbb{R}^{M+1}$ that is spanned by the column vectors φ_i of the matrix ϕ (if M is smaller than N).



Learning w with Maximum Likelihood

Probabilistic Modeling

The prediction error of a hypothesis h_w can be interpreted as a noise term ϵ:

$$y_n = h_w(x_n) + \epsilon$$

Usually, the noise is modeled with a Gaussian distribution with mean $h_w(x_n)$ and precision (inverse variance) β:

$$\mathrm{P}(y_n|x_n, w, \beta^{-1}) = \mathcal{N}(y_n|h_w(x_n), \beta^{-1})$$

Computing the optimal prediction for a given set of parameters is then equal to computing the expectation value $\mathrm{E}[y_n|x_n]$:

$$\mathrm{E}[y_n|x_n] = \int y_n \mathrm{P}(y_n|x_n) dy_n = h_w(x_n)$$

Learning w with Maximum Likelihood

Maximum Likelihood

For a given dataset $\mathcal{S}$, the weights w should be computed so that they maximize the likelihood of the target values y_n for the input x_n. This amounts to maximizing the following probability:

$$\begin{aligned} \mathrm{P}(y|x,w,\beta) &= \prod_{n=1}^{N} \mathcal{N}(y_n|h_w(x_n),\beta^{-1}) \\ &= \prod_{n=1}^{N} (2\pi\beta^{-1})^{-\frac{1}{2}} \cdot \exp\left(-\frac{\beta}{2}\left(y_n - h_w(x_n)\right)^2\right) \end{aligned}$$

Maximizing the log-Likelihood

Instead of maximizing $\mathrm{P}(y|x,w,\beta)$, it is often easier to maximize $\ln \mathrm{P}(y|x,w,\beta)$:

$$\ln \mathrm{P}(y|x,w,\beta) = -\frac{\beta}{2}\sum_{n=1}^{N}(h_w(x_n) - y_n)^2 + \frac{N}{2}\ln\beta - \frac{N}{2}\ln 2\pi$$

Learning w with Maximum Likelihood

Maximizing the log-Likelihood

Instead of maximizing $\mathrm{P}(y|x, w, \beta)$, it is often easier to maximize $\ln \mathrm{P}(y|x, w, \beta)$:

$$\ln \mathrm{P}(y|x, w, \beta) = -\frac{\beta}{2}\sum_{n=1}^{N}(h_w(x_n) - y_n)^2 + \frac{N}{2}\ln\beta - \frac{N}{2}\ln 2\pi$$

$\rightarrow$ Minimize $\sum_{n=1}^{N}(h_w(x_n) - y_n)^2 = 2 \cdot \mathcal{L}_S(h_w)$

For linear basis function models, when assuming a Gaussian distribution of the prediction error, maximizing the log-likelihood is equivalent to computing the least squares solution!

Classification: Task Definition

Input

- A dataset $\mathcal{S} = \{(x_0, y_0), (x_1, y_1), \dots, (x_N, y_N)\}$ with $x_i \in \mathbb{X}^K$ and $y_i \in \mathbb{C}$
- $\mathbb{C} = \{C_1, \dots, C_L\}$ is discrete space of classes $C_1, \dots, C_L$
- We will only consider problems where each x_i can be assigned to exactly to class C_l; the classes $C_1 \dots C_L$ therefore divide the input space $\mathbb{X}^K$ into decision regions
- Classification tasks with two classes $\mathbb{C} = \{C_1, C_2\} = \{0,1\}$ are also referred to as binary classification
- In general K classes are represented using a 1-of-K coding scheme where $C_l = (0,0, \dots, 1, \dots, 0) \in \{0,1\}^K$

Desired Output

A classifier $h: \mathbb{X}^K \longrightarrow \mathbb{C}$ that predicts y_n for a given x_n with minimum error

Generalized Linear Models for Classification

The output of linear regression is a weight vector w that defines a linear function of the form

$$y_{reg}(x_n, w) = \sum_{j=0}^{M} w_j \phi_j(x_n)$$

$y(x_n, w)$ maps input values to a continuous space $\mathbb{Y}^L$. Applying this type of model to classification requires an additional nonlinear activation function f that discretizes the output space:

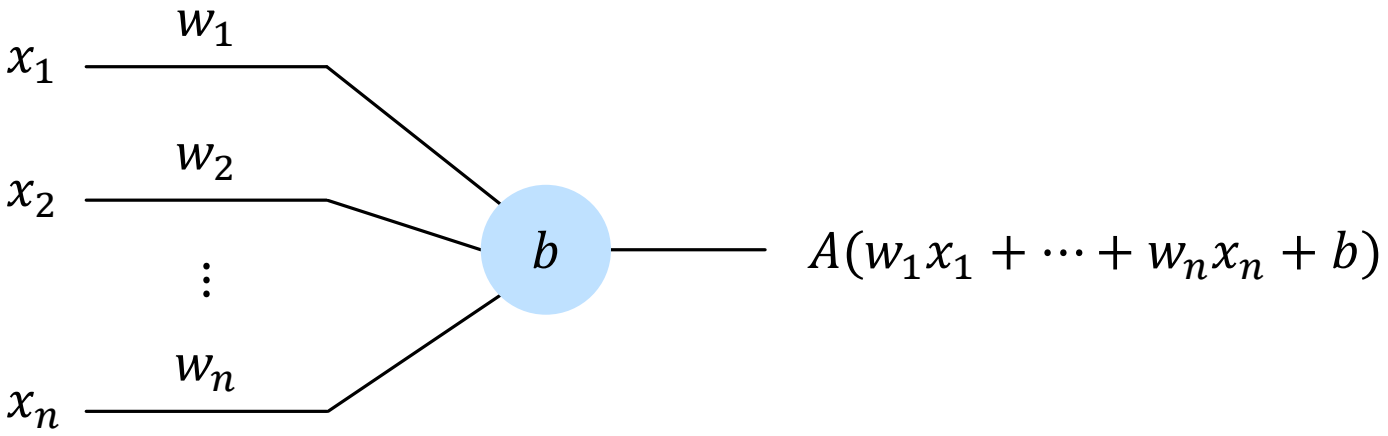
$$y_{cls}(x_n, w) = f\left(\sum_{j=0}^{M} w_j \phi_j(x_n)\right)$$

y_{cls} is called a generalized linear model. Subspaces $\mathrm{X}^* \subseteq \mathbb{X}^K$ with $y_{cls}(x_n, w) = c$ for a constant $c \in \mathbb{C}$ are called decision surfaces. The decision surfaces are linear functions of x:

$$y_{cls}(x_n, w) = c \leftrightarrow \sum_{j=0}^{M} w_j \phi_j(x_n) = c^*$$

Recap: Artificial Neurons

An analog artificial neuron is defined by its synaptic weights and an activation function:



The output $n(x)$ of a neuron n with activation function A, synaptic weights w, bias b, and input x is computed as follows:

$$n(x) = A(w_1x_1 + \cdots + w_nx_n + b) = A(w^Tx + b)$$

→ **Analog neuron models are generalized linear models!**

Linear Separation for Binary Classification

Basic Idea

Two classes C_1 and C_2 can be linearly separated based on a linear discriminant function

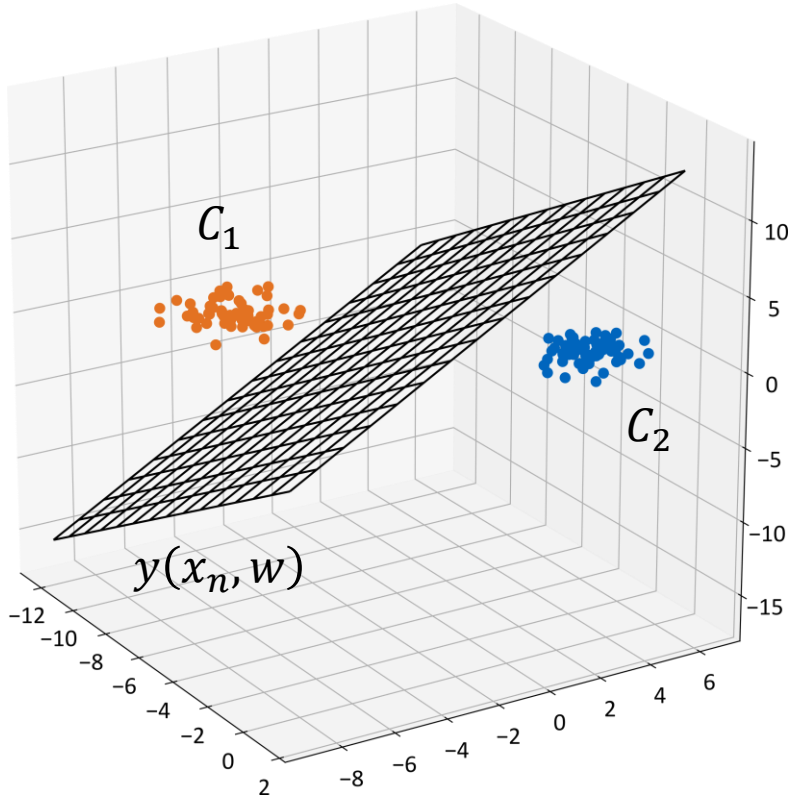
$$y(x_n, w) = w^T x_n = c^*$$

Mapping to C_l

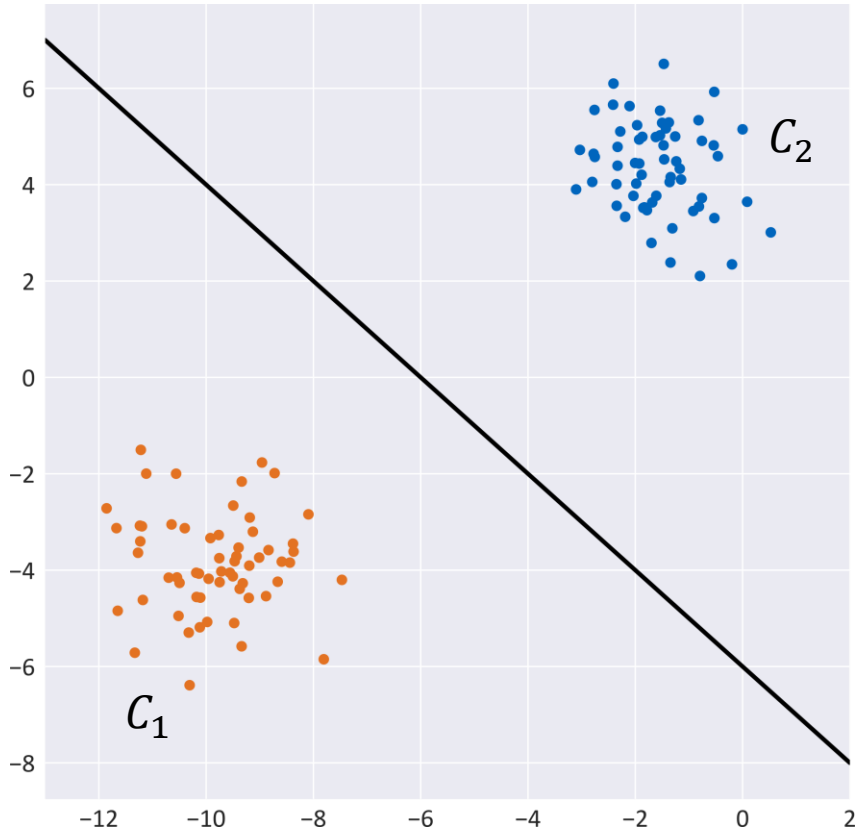
The class of a data point x_n is determined based on the value of $y(x_n, w)$. The most common activation function is the Heaviside step function f_H:

$$f_H(x) = \begin{cases} 0 & \text{if} \quad n < 0 \\ 1 & \text{if} \quad n \geq 0 \end{cases}$$

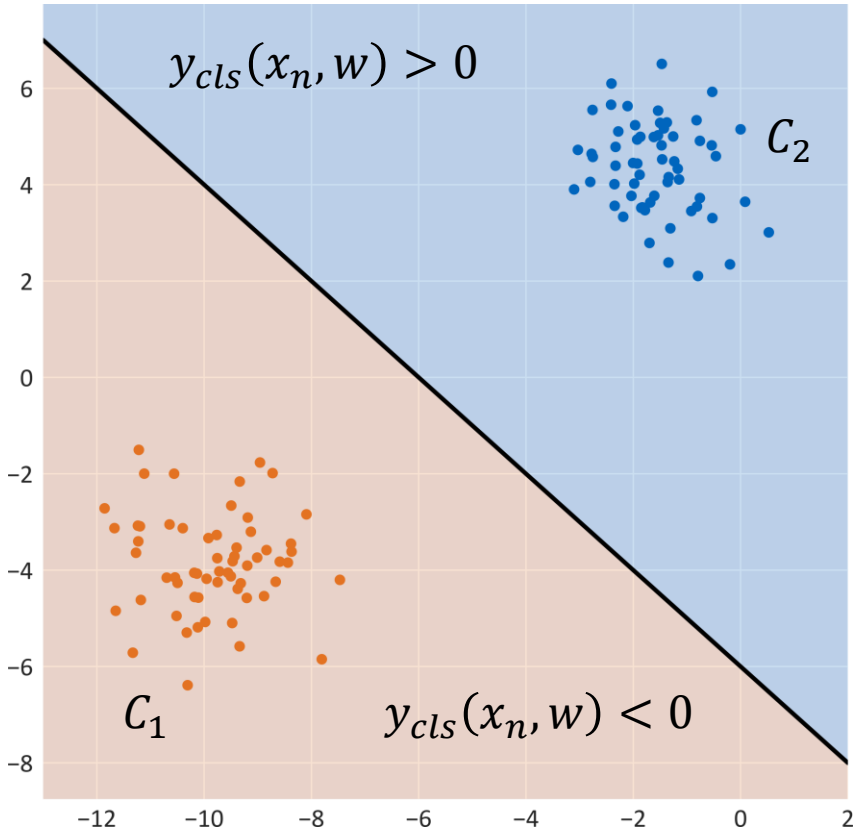
The resulting decision surface is $w^T x_n = 0$



Linear Separation for Binary Classification



The two classes C_1 and C_2 be separated by the line $w_0 + w_1 x_1 + w_2 x_2 = 0$



The decision surface divides the definition space of the samples into two half-spaces

Classifying Multiple Classes

In general, classifiers for more than two classes can be constructed by combing discriminant functions that classify subsets of $\mathbb{C} = \{C_1, \ldots, C_K\}$. However, the two most evident approaches suffer from a serious issue:

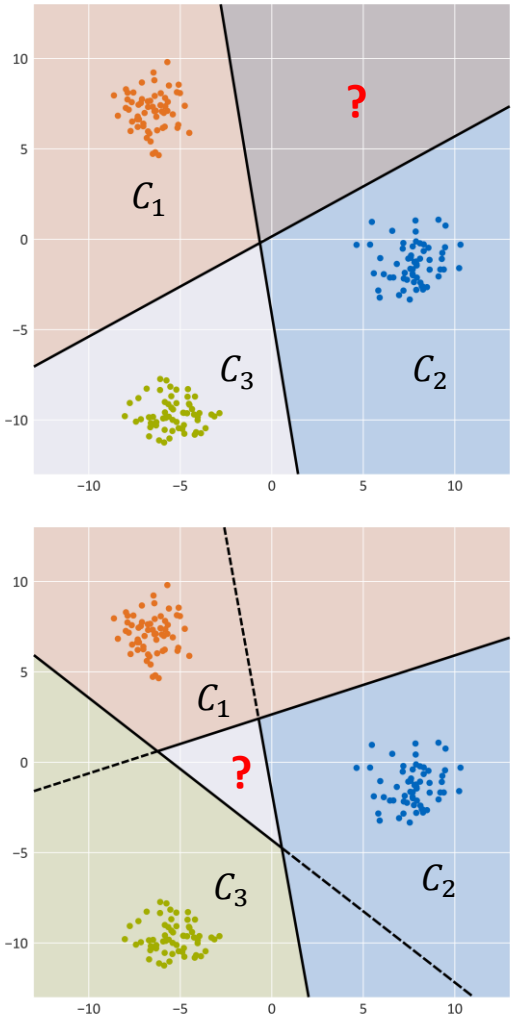
One-Versus-the-Rest Classifier (Top)

- Separation of K classes with $K - 1$ binary discriminant functions
- Every discriminant function separates one class from all others

One-Versus-One-Classifier (Bottom)

- Pairwise separation of K classes with $\mathrm{K}(K-1)/2$ binary discriminant functions
- The class of a data sample is assigned by a majority vote

In both cases, some regions are classified ambiguously!



Classifying Multiple Classes

Problem Analysis

Both one-versus-the-rest and one-versus-one classification fail because the input space regions defined by the discriminant functions overlap or do not cover the whole input space.

→ Determine a partition of the input space

Solution

Every class C_k is assigned a linear function of the form

$$y_{C_k}(x_n, w_k) = w_k^T x_n$$

A sample x_n is assigned to class C_k if the $y_{C_k}(x_n, w)$ of that class is larger than for all other classes:

$$C^* = \underset{C_k \in \mathbb{C}}{\operatorname{argmax}}\, y_{C_k}(x_n, w_k)$$

Classifying Multiple Classes

The decision regions defined by the classifier

$$C^* = \underset{C_k \in \mathbb{C}}{\operatorname{argmax}}\, y_{C_k}(x_n, w)$$

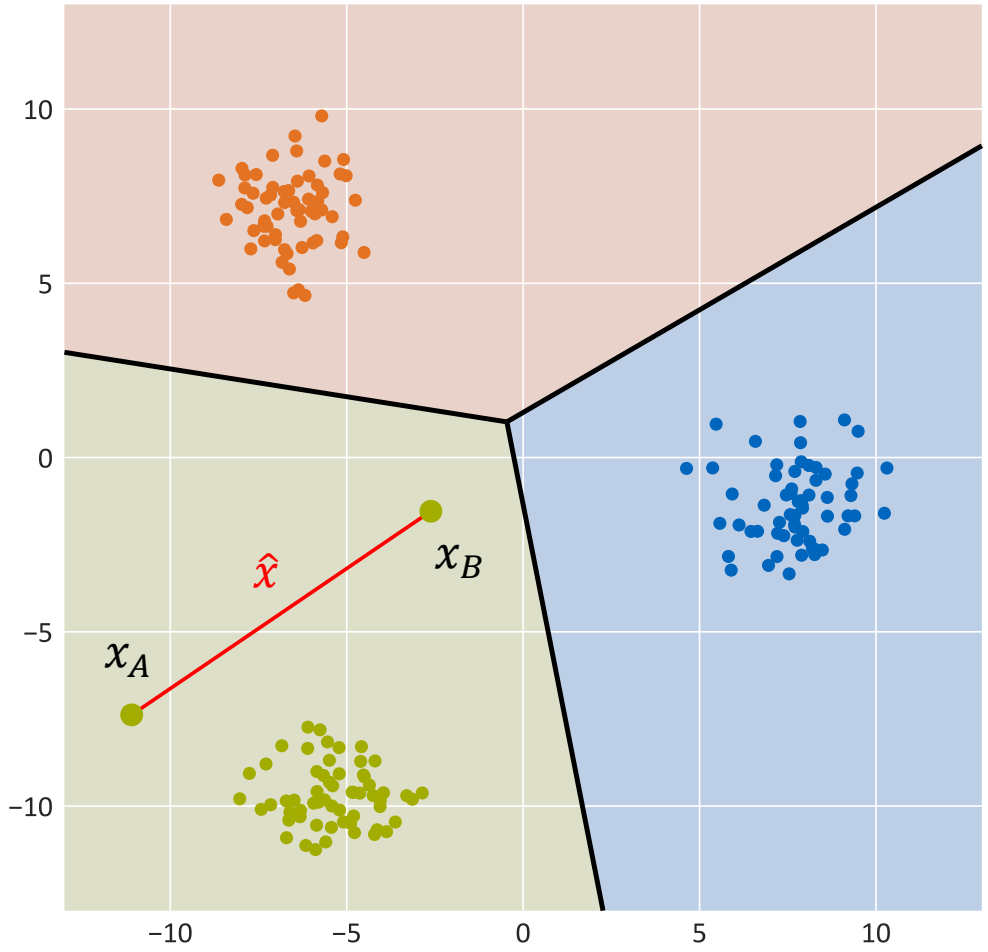
are always convex. Consider two points x_A and x_B in a decision region C_i. Any point $\hat{x}$ on the line connecting them can be expressed as

$$\hat{x} = \lambda x_A + (1 - \lambda) x_B \text{ with } 0 \leq \lambda \leq 1$$

Since the functions y_{C_k} are linear, it follows that

$$y_{C_k}(\hat{x}) = \lambda y_{C_k}(x_A) + (1 - \lambda) y_{C_k}(x_B)$$

With both x_A and x_B lying in the same decision region, it finally follows that $y_{C_i}(\hat{x}) > y_{C_k}(\hat{x})$ for all $i \neq k$, i.e. $C_i = \underset{C_k \in \mathbb{C}}{\operatorname{argmax}}\, y_{C_k}(x_n, w)$.



Representing Classes

A Naïve Approach

The classes C_k are elements of a discrete set $\mathbb{C}$. The choice of $\mathbb{C}$ can have a major impact on the learning process. The probably most obvious choice is to set $\mathbb{C} = \{1, 2, 3, \dots, L\}$:

$$C_1 = 1$$
$$C_2 = 3$$
$$\vdots$$
$$C_L = L$$

Problem

Assigning class identifiers from an ordered set implicitly also imposes a not existing order on the classes that might be captured by the learned model.

→ **Use one-hot encoding!**

One-Hot Encoding

To avoid introduction any latent information contained in ordered sets, a common scheme of encoding clusters and classes is one-hot encoding:

- For a data set with L clusters/classes set $\mathbb{C} = \mathbb{R}^L$
- Cluster/class C_k is assigned the vector $c_k = (0, \dots, 0,1,0, \dots, 0)^T$ where every except for the kth entry is 0

Example

Sample	House	Car	Cat	Flower
s_1	0	0	1	0
s_2	0	1	0	0
s_3	0	0	0	1
s_4	1	0	0	0
s_5	0	1	0	0

Least Squares for Classification

Encoding categorical class labels as one-hot vectors in $\mathbb{R}^n$ enables mapping the classification problem to a regression problem that can be solved with least squares. Recall that every class is described by its own linear model:

$$y_{C_k}(x_n, w_k) = w_k^T x_n$$

Grouping all functions y_{C_k} together yields:

$$y(x_n, W) = W^T x_n$$

W is a matrix that is comprised of the weight vectors w_k of the individual linear models y_{C_k}:

$$W = (w_0 \quad \dots \quad w_L)$$

Least Squares for Classification

Loss Function

With $X = (x_0^T, \dots, x_n^T)^T$ and $Y = (y_0^T, \dots, y_n^T)^T$, the loss function $\mathcal{L}_S$ of the least squares regression problem can be defined as follows:

$$\mathcal{L}_S(W) = \frac{1}{2}\mathrm{Tr}\{(XW - Y)^T(XW - Y)\}$$

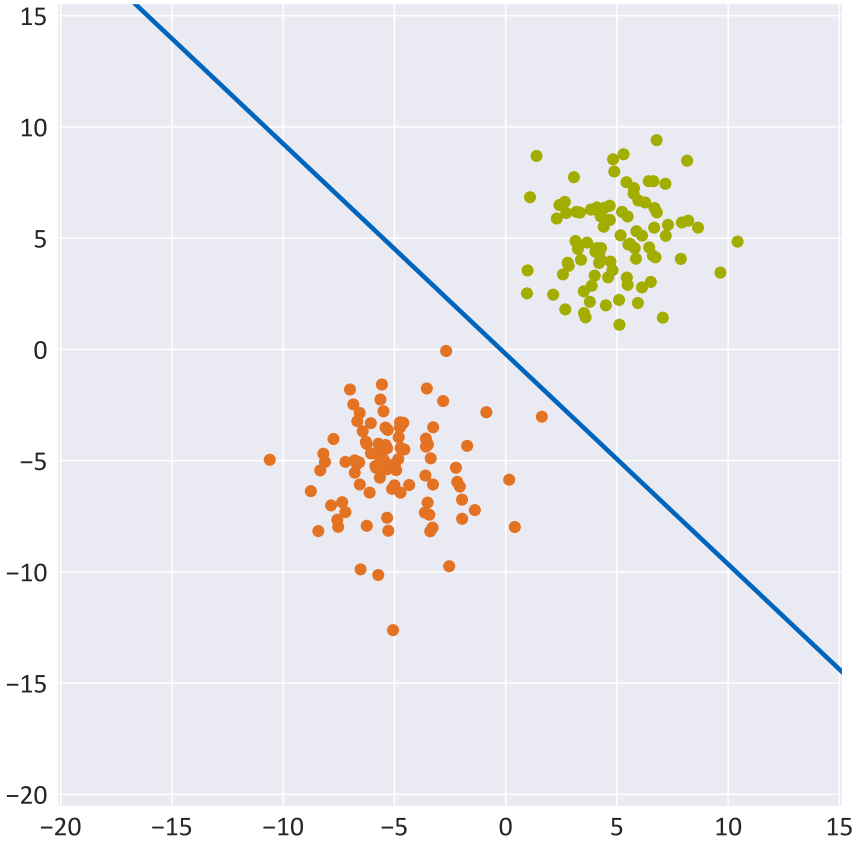
The trace operator ensures that only the quadratic terms on the diagonal of the product matrix account for the loss.

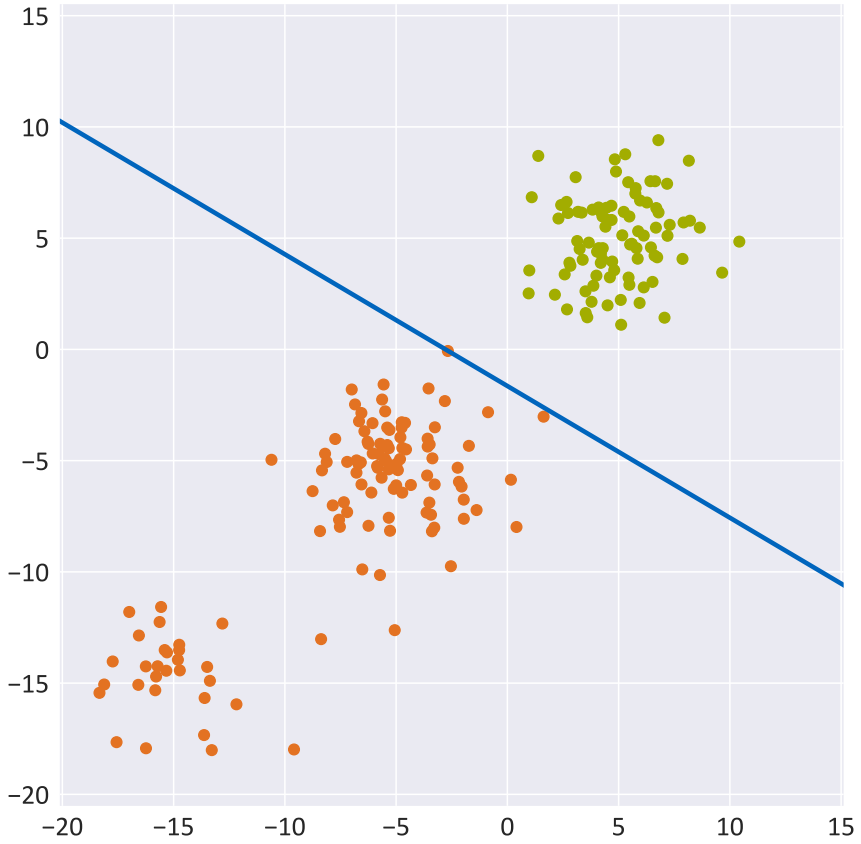
Least Squares Solution

Minimizing $\mathcal{L}_S(W)$ analogously to the standard regression case yields the weights values for the linear models y_{C_k}:

$$W = (X^T X)^{-1} X^T Y$$

Issues of Least Squares Linear Classification

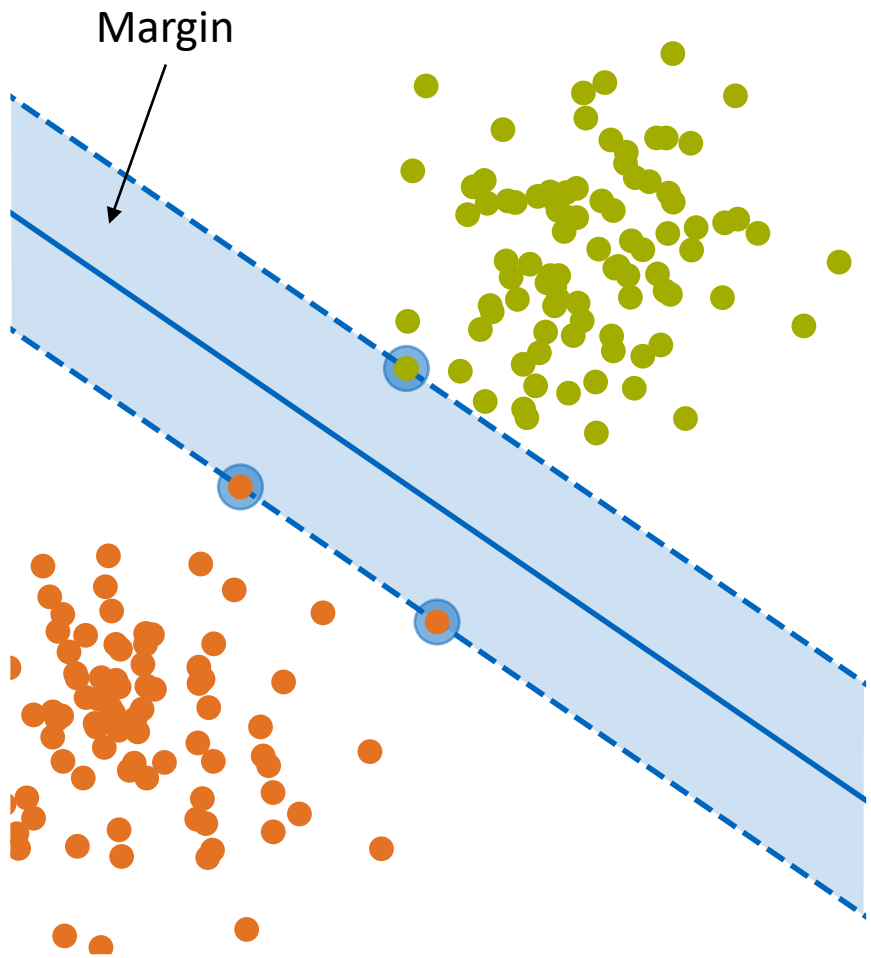




Least squares classification is sensitive to outliers!

Support Vector Machines

- When using least squares for linear classification, the resulting hyperplanes depend on all samples in the dataset. If the input is linearly separable, the algorithm computes one of infinitely many solutions.
- Support Vector Machines (SVMs) minimize the generalization error by computing a hyperplane that maximizes the margin of the classifier, i.e. the smallest distance between the decision boundary and the training samples.
- Maximizing the margin can be motivated using computational learning theory.



Support Vector Machines

Computing Distances from the Decision Boundary

Like in the standard classification task, the separating hyperplane of a SVM is defined as follows:

$$y(x) = w^T \phi(x) + b$$

By setting $\mathbb{C} = \{-1, 1\}$, all correctly classified examples fulfil the inequality

$$t_n y(x_n) > 0$$

The distance of a sample x_n from the decision boundary therefore can be computed as

$$d(x_n) = \frac{|y(x_n)|}{\|w\|} = \frac{t_n y(x_n)}{\|w\|} = \frac{t_n (w^T \phi(x) + b)}{\|w\|}$$

Support Vector Machines

Optimization Problem

Based on d, the optimization problem for computing the decision boundary can be formulated as follows:

$$\arg\max_{w,b}\left\{\frac{1}{\|w\|}\min_{n}[t_n(w^T\phi(x)+b)]\right\}$$

By scaling w and b, one can easily set $t_n(w^T\phi(x^*)+b)=1$ for the point x^* that lies closest to the decision surface. This results in the following optimization problem:

$$\arg\min_{w,b}\|w\|$$

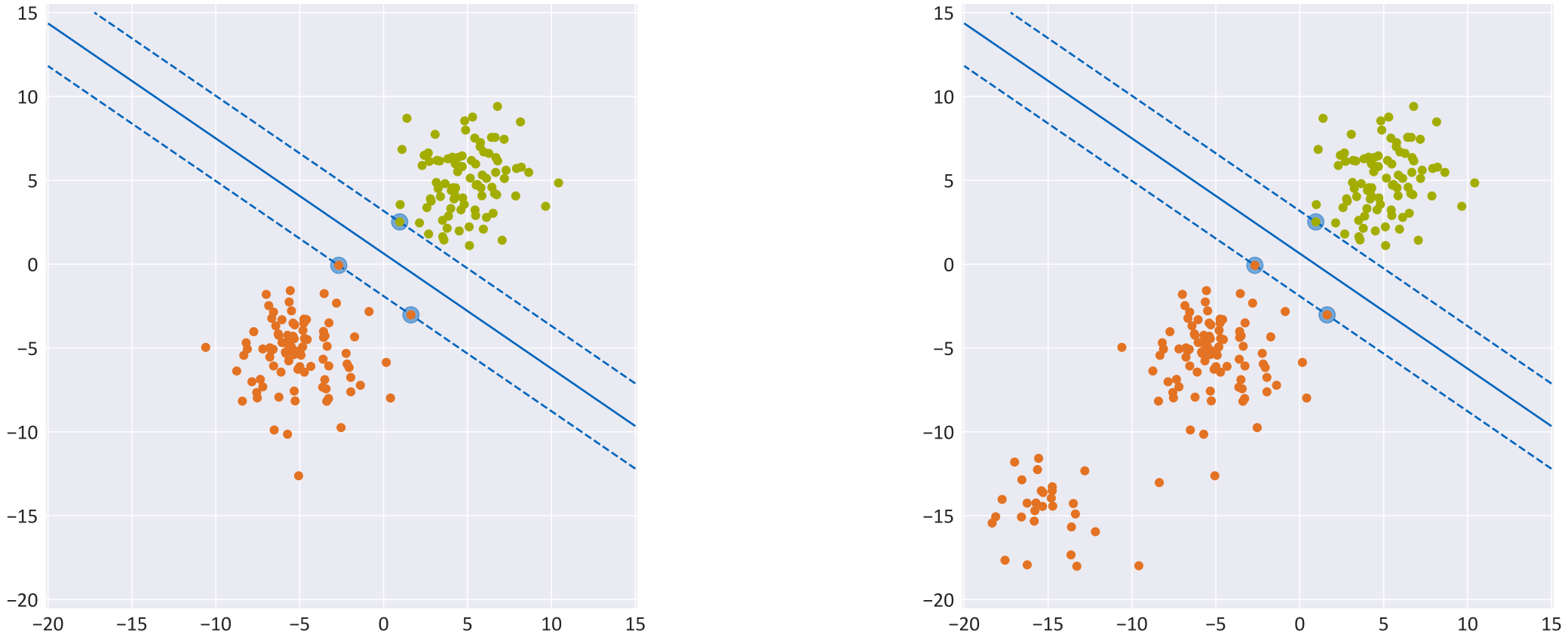
$$\text{with } t_n(w^T\phi(x)+b)\geq 1 \text{ for all samples } x \text{ in the dataset}$$

The above problem can be solved with quadratic programming (→ Machine Learning lecture).

Support Vector Machines

- Solving the optimization problem yields a set of support vectors that define the decision boundary of the SVM.
- The support vectors are samples from the training dataset.
- After training the SVM, all samples except for the support vectors can be discarded!
- In the general case, the optimization function also includes slack variables to allow for training samples that lie inside the margin. This ensures better generalization performance for overlapping classes.

Classification with SVMs



SVM classifier with support vectors and margins. Outliers do not affect the result.